e-Informatica

software engineering journal

2025 volume 19 issue 1

e-Informatica

e-Informatica

software engineering journal

2025 volume 19 issue 1

e-Informatica

Wrocław University
of Science and Technology

Editor-in-Chief
Lech Madeyski (Lech.Madeyski@pwr.edu.pl, https://madeyski.e-informatyka.pl)
Associate Editor
Mirosław Ochodek (Miroslaw.Ochodek@put.poznan.pl)
Editor-in-Chief Emeritus
Zbigniew Huzar (Zbigniew.Huzar@pwr.edu.pl)

Faculty of Information and Communication Technology, Department of Artificial Intelligence
Wrocław University of Science and Technology,
50-370 Wrocław, Wybrzeże Wyspiańskiego 27, Poland

e-Informatica Software Engineering Journal
www.e-informatyka.pl, DOI: 10.37190/e-inf

Editorial Office Manager: Wojciech Thomas

Typeset by Wojciech Myszka with the LaTeX 2_ε Documentation Preparation System

OFICYNA WYDAWNICZA POLITECHNIKI WROCŁAWSKIEJ
Wybrzeże Wyspiańskiego 27, 50-370 Wrocław
www.oficyna.pwr.edu.pl;
e-mail: oficwyd@pwr.edu.pl; zamawianie.ksiazek@pwr.edu.pl

ISSN 1897-7979

Editorial Board

Contents

e-Informatica Software Engineering Journal
Volume 19, Issue 1, pages: 1–29
DOI: 10.37190/e-Inf250101

BibTeX

Bug Report Analytics for Software Reliability Assessment using Hybrid Swarm – Evolutionary Algorithm

Sangeeta*, Sitender*, Rachna Jain*, Ankita Bansal*
*Corresponding authors: sangeeta@msit.in, sitender@msit.in, rachnajain@bpitindia.com, ankita.bansal06@gmail.com

Article info

Dataset link: https://zenodo.org/records/13147825

Keywords:
software development process
bug report analysis
optimization
swarm evolutionary algorithms
software reliability

Submitted: 30 Sep. 2023
Revised: 12 Aug. 2024
Accepted: 17 Aug. 2024
Available online: 3 Oct. 2024

Abstract

Background: With the growing advances in the digital world, software development demands are increasing at an exponential rate. To ensure reliability of the software, high-performance tools for bug report analysis are needed.
Aim: This paper proposes a new "Iterative Software Reliability" model based on one of the most recent Software Development Life Cycle (SDLC) approach.
Method: The proposed iterative failure rate model assumes that new functionality enhancement occurs in each iteration of software development and accordingly design modification is made at each stage of software development. In terms of defects, testing effort, and added functionality, these changing needs in each iteration are reflected in the proposed model using iterative factors. The proposed model has been tested on twelve Eclipse and six JDT software failure datasets. Proposed model parameters have been estimated using a hybrid swarm – evolutionary algorithm.
Results: The proposed model has about 32% and 55% improved efficiency on Eclipse and JDT datasets, respectively, as compared to other models like Jelinski Moranda Model, Shick–Wolverton Model, Goel Okumotto Imperfect Model, etc.
Conclusion: In each analysis done, the proposed model is found to be reaching acceptable performance and could be applied on other software failure datasets for further validation.

1. Introduction

With the growing advances in cutting-edge innovations in digital world, software development demand from software industries is increasing exponentially. Due to limited budget, high demand and lack of time, there is a high probability of fault occurrence in the newly developed software, making reliability a concern. The reliability of software is considered as the most crucial quality attribute [1]. At one end, software has made today's life much more comfortable but at other end failures in unreliable software may cause life threatening issues to humans. So unpredictable software failures make life challenging due to the decreasing reliability of the newly developed systems. Software reliability could be only measured using bug report analysis of the software. Using the latest tools and techniques at most care has been taken to develop reliable software, but practically a software developer cannot produce defect-free software [2]. Therefore, there should be a way to avoid software failures to further avoid severe losses to human life or any financial losses in industry. The prerequisite of software developers is to know whether developed software is reliable before they are dispatched to customers. In the competitive arcade of software development, software industries must ensure the reliability of their software to satisfy their customers and make an outlook in the global market [3–5]. As the size and complexity of software grow, so does the number of errors per 1000 lines of code. It becomes more difficult to obtain valuable data for reliability estimation. Software reliability models are only the way to estimate software reliability. Many software reliability estimation models have been developed or proposed in literature, but these developed models have been designed only for specific software applications, works in particular environment and on specific datasets and assumptions of the model. Thus, developing a new software reliability estimation model is a question behind the software developers. There are mainly three reasons that reflect the need to develop a new software reliability model.

- Existing models have been developed only based on traditional software development approaches and do not incorporate the latest SDLC approach [6–10].
- Available research publication has only 31% experimental research, and in this, only 13% pure experimental work has been found [11]. The reason behind this is the narrow failure dataset availability as software companies are not releasing their fault data for public research.
- Existing models are made application-specific, and these can precisely estimate the reliability of application-specific software only. These are not well proving their capacity if applied to other applications. The models are further developed based on the Non-homogeneous Poisson Process-based approach [12–23].

Software reliability models are divided into two categories: failure rate-based models and NHPP behavior-based models. The first category of software reliability models used in business and research were failure rate-based models [9]. However, these models need enhancement to incorporate the latest software development methods and estimate software reliability. Furthermore, more realistic assumptions need to be included in these models to be more suitable in the latest software development environments for reliability estimation. Keeping in mind the need to use the latest software development methods, the authors recommend the iterative method as a top-down refinement approach for software development. The more recent agile method, iterative enhancement method, and spiral method are well-known models supporting the iterative software development process [24, 25]. The authors in this paper propose a failure rate model known as Iterative Software Reliability (ISR) model which incorporates the behavior of these latest software development methods, mainly the iterative SDLC approach [26], for estimating the reliability of big data generating software. There is a necessity to add a factor that will reflect all shifting needs based on the defect analysis in each software development process model [27–29]. Identifying a varying need in each iteration using some factor could be very beneficial for iterative software development inappropriate resource usage. In this paper, varying conditions in each iteration are represented with an iterative factor. The iterative element's value is determined by the iterative parameter, which ranges from 0 to 1 based on the assumption that user acceptability improves with new capabilities in each iteration. The value of an iterative factor is obtained, representing all the requirements that have been changed from the previous iteration in current iteration releases using iterative parameters.

Value of iterative factor increases in some initial iterations because there is an abrupt change in requirements due to many defects found and added functionality in an iteration. Later, when the system has added almost all the functionalities for full implementation of the system, the value of the modulation factor represents that the change in requirements decreases, so its value changes from higher to lower one. When the value of the iterative parameter reaches again near to one, it is assumed that the system is reliable enough and has attained all the vital functionalities to fulfil the end-user needs. Thus, the iterative parameter reflects the level of user acceptance at each phase. Further in the cutting-edge software development environments, using the latest testing techniques, there is always a possibility of removing more than one fault at a time. Fault introduction and removal probabilities are well incorporated in the proposed ISR model development. To the best of authors' knowledge, no studies are available in the literature that includes such factor in the ensemble for reliability estimation via software reliability models. Further, in modeling software, reliability parameter estimation methods play an essential role. However, classical numerical optimization techniques are highly based on constraints, and they may not converge to maxima or minima in multimodal cases [7, 30]. To address non-differential, non-linear, and multimodal problems, new parameter optimization approaches based on nature-inspired optimization algorithms are available [31, 32]. This paper proposes model parameter optimization using a hybrid swarm evolutionary algorithm [33]. This algorithm incorporates the best feature of the artificial bee colony and differential evolution in parameter estimation. Swarm evolutionary algorithm is a new hybrid swarm-evolutionary algorithm for software reliability model parameter estimation is proposed in this paper. This algorithm is centered on the social behavior of artificial bee colonies (Yang 2010a; Abu-Mouti and El-Hawary 2012) and evolutionary behavior of DE algorithm (Storn and Price 1997). Here swarm intelligence of employee bee is enhanced for providing exploitation to provide a better local search of neighborhood positions using evolutionary principle based DE algorithm. Onlooker bee phase has been improved here by incorporating a new factor, showing the fitness probability of the ecological space.

The proposed ISR model is empirically validated using Eclipse and JDT software failure datasets. Furthermore, the Goodness-of-fit of the proposed work is estimated using Sum of Squared Errors (SSE), Mean Square Error (MSE), Accuracy of Estimate (AE), and Theil Statistics (TS). The proposed ISR model fits the estimation of software reliability under the iterative software development process. Summary of the significant contribution of our work is as follows:

1. Objective 1: To study existing software reliability estimation models and specially failure rate behavior-based models.
2. Objective 2: To propose a new model that considers the behavior of the recently used iterative software development life cycle processes.
3. Objective 3: To validate the proposed model (ISR) on twelve Eclipse common software failure datasets and six JDT software failure dataset iterations.
4. Objective 4: To compare the proposed model with five existing models named Jelinski Moranda Model, Shick–Wolverton Model, Goel Okumotto Imperfect debugging Model, GS Mahapatra and P Roy Model and Modified SW Model.

Proposed ISR model is found to be supporting software developers and end-users in estimating software reliability at each software development iteration and hence in software systems evolution. The rest of the paper is structured as follows: Section 2 discusses the existing literature in software reliability modelling and various parameter estimation algorithms used in software reliability model development. Then, in Section 3 proposed software reliability model is discussed. In Section 4, experimental setup and results are discussed. Section 5 discusses the threats to validity and finally Section 6 concludes the paper with future scope.

2. Literature review

A thorough literature has been done by authors which is elaborated in 2 subsections of this section. The first section focusses on development of software reliability models of the software that has been developed using latest software development life cycle processes (SDLC). The next subsection presents a survey of parameter optimization algorithm that is the major requirement for precise software reliability measurement using software reliability models. Already existing parameter optimization algorithms applied on software reliability modeling can only better estimate the reliability of software that has been developed with their specific assumptions and applications.

2.1. Software reliability models and their assumptions

About 300 software reliability models have been developed in the last four decades. The developed models have specific environments, assumptions, and applications [34–40]. Failure rate behavior-based models are used to analyze the program failure rate per fault and study how failure changes at the time of failure in an interval of time. The JM model [8] is the first to estimate software reliability. This behavior-based debugging approach assumes the program initially has a fixed, constant, and unknown number of defects. The duration between failures is considered to be independent and distributed exponentially in these models. Table 1 discusses software reliability models along with their assumptions.

Table 1. Survey of Software Reliability Models

Reference	Year	Application domain and assumptions Related to software reliability models
[9]	1972	The first Markov process-based model assumes that the total number of initial software defects is unknown and fixed. Furthermore, it assumes that the duration between failures is always a random variable that is distributed exponentially.
[41]	1973	A Bayesian reliability growth model is presented here and assumes that the program is complete to work for a continuous time-period between the failures. It also considers a repair rule for program developers at each failure. It does not take into account the program's internal structure.
[42]	1988	This model fits well into a general framework of the Bayes problem and assumes a Bayesian approach for inference by considering the conditions as an empirical Bayes problem.
[10]	1978	The Hazard function is proportional to the current number of total fault content and the time since the previous failure in this model originated from the JM model.
[43]	1977	This model extends the JM Model and SW Model; It allows more than one fault at each time interval.
[44]	1979	It follows a Markov process like the JM Model. This characterizes the transition between the modules while execution as following the Markov property.
[43]	1979	This is evolved using the JM and geometric model-based and follows Poisson distribution-based failure rate. It assumes that the number of faults occurring at intervals follows a Poisson distribution with an intensity rate of Dki-1.
[45]	1998	It has extended the JM geometric model by describing the behavior of software as having safe and unsafe states.

Table 1 continued

Reference	Year	Application domain and assumptions related to software reliability models
[46]	1979	This model considers the phenomenon of imperfect debugging for software development and testing.
[13]	1981	A variation of the JM model develops this model. And assumes that: – Time separations between error detections – Number of errors per written instruction – Failure rate of the software is considered as proportional to current error content in software.
[47]	1981	This model is presented as a particular case of the JM and NHPP models.
[48]	1985	This is evolved by considering the Bayesian process in the JM model and Mein-hold and Singpurwalla model. It also assumes that it is easy to calculate the distribution of undetected errors at the end of testing to see the relative effects of uncertainty in several errors and fault detection efficiency.
[49]	1985	In this model, an alternative formulation of the JM and Little-wood models is presented. Here a formulation in terms of failure rate rather than inter failure time is given.
[50]	1991	It is assumed that various defects contribute differently to the failure rate. In addition, the software's structure is taken into account in this method.
[43]	2000	This model is extended from the JM Model. The effect of environmental factors has been incorporated in the model development.
[51]	2003	A Moranda de-eutrophication model is proposed by assuming time between failures as a statistically independent exponential random variable with a given failure rate.
[37]	1991	It has evolved from the JM model's assumptions and formulates the total expected software costs with two different release policies.
[51]	2006	This model extends the JM model by using a negative binomial prior distribution for the number of remaining faults and a Gamma distribution for the rate at which each fault becomes disclosed.
[15]	2011	This is the modification of the famous JM model, and it is based on cloud theory.
[16]	2012	This model considers imperfect debugging in fault removal and considers that when a failure occurs, then the detected fault is assumed to be removed with probability p, and it is not removed perfectly with probability q. It also assumes that a new fault may be generated with probability r.
[52]	2016	They discussed that software reliability analysis can be done at various phases during the development of engineering software. JM and SW SRGMs are two exceptional cases of this general SRGM.
[53]	1985	This model proposes that the reliability of computer software may be evaluated holistically by taking a Bayesian perspective, and it provides an alternate justification for the widely utilized JM model.
[54]	2017	To estimate the parameters of the JM model, objective Bayesian inference was developed. The Bayesian estimators, credible intervals, and coverage probability of the parameters are obtained using Gibbs sampling.
[55]	2018	1) It takes into account the impact of software upgrades on subsequent releases. 2) It also implies that software problems are classified as soft or hard depending on the amount of work and time required to correct them.
[56]	2018	They are assumed as a special case of the famous inflection S-shaped model and generalized GO model. Special attention is given to non-existence issues of MLE.
[57]	2018	A variable η is considered as a random variable to represent the uncertainty of FDR in the operating environment.
[58]	2019	The fault Removal process for multi-release OSS systems is assumed by considering the concept of change point.
[22]	2018	Assumes that all corrected issues in a current software release are used to determine the next software release; entropy-based methods assess uncertainty concerns.
[59]	2019	Complexity issues like the debugging process, coverage factor, and delay time Function in a distributed computing environment are concerned.
[60]	2021	Uncertain factors are taken into account while developing a logistic growth model for software reliability estimates.

Table 1 continued

Reference	Year	Application domain and assumptions related to software reliability models
[61]	2021	The extension predicts the software reliability of the Jelinski Moranda model to a stack of feature-specific models.
[62]	2021	Application of EM algorithm is done to NHPP software reliability assessment with generalized failure count data.
[63]	2020	The importance and need of software reliability from an industry perspective have been discussed.
[64]	2020	A new statistical time series and ARIMA approach-based software reliability prediction is proposed. time series based technique is very flexible as it needs a very few assumptions.
[65]	2019	Quantitative Testing Effort based estimate of software Reliability for Multi Release Open Source Software Systems is proposed. Testing Effort is assumed to be impacting the number of faults generated in the software at a particular time interval.
[66]	2021	Green computing based software systems is discussed here with some new modulation parameters that are quantifying the amount of fault introduced in the software.
[67]	2021	New iterative failure rate behavior based model is proposed for iterative software development life cycle approach based softwares. In each iteration new fault may be introduces and it assumes that some faults may gete flow from the previous iterations.
[68]	2022	Grey-based approach for estimating software reliability under nonhomogeneous Poisson process is proposed for software reliability assessment using mean value function.
[69]	2022	Various Machine learning techniques can be used in software reliability prediction. Various ML techniques have been used here for reliability prediction. But the dataset required must be large enough here.
[70]	2023	During testing process optimal time management is a need and an approach using S-curve two-dimensional software reliability growth model is proposed here.
[71]	2023	Reliability Assessment for Open-Source Software using Probabilistic approach is proposed. For assessing reliability more accurately Imperfect debugging along with Marine Predators Algorithm with six probabilistic models is used here.
[72]	2023	A new Model for software reliability growth estimation based on generalized inflection S-shaped is proposed. This has considered fault reduction factor and optimal release time for proposed model development.
[73]	2023	On the basis of assuming latest Emerging trends and future directions in software reliability growth modeling. Engineering reliability and risk assessment is discussed.
[74]	2023	Testing coverage is used here for software reliability growth modeling and considers uncertainty of operating environment.

2.2. Parameter optimization processes

Parameter optimization processes play a pivotal role in estimating the reliability of software. Traditional techniques [30, 73, 74] have been found in use for the estimation of parameters of software reliability models. Fatefully, all model parameters usually have non-linear relationships. Because of this, traditional techniques for optimizing parameters suffer various problems in finding the optimum value of models to predict software reliability better. In recent years, meta-heuristic algorithms have become very popular due to their simplicity, flexibility, derivative-free nature, and capability to avoid the local optima problem. These algorithms explore the feasible solution space using some specified rules. Several nature-inspired algorithms have been created in the literature and may be used to address numerical optimization-based issues in various domains [75–83]. Mirjalili [79] created a hybrid PSOGSA method for mathematical function optimization in 2010 by combining PSO and PSOGSA. Furthermore, In 2014, Mirjalili [83] presented a binary optimization method based on a hybrid PSOGSA algorithm. Liu and Zhou [84] used a new QPSO (Quantum Particle Swarm Optimization) technique to solve high-dimensional complex issues in 2014. Li et al. [85] proposed ABC-assisted DE for ORPF (Optimal Reactive Power Flow) in 2013. They have used ABC in the DE algorithm to recover the shortcoming of large population requirements to avoid premature convergence. Tiwari et al. [86] presented a hybrid ABC method with DE, which he used to solve welded beam design issues. This method changed the employee bee phase position update equation and utilized DE for the onlooker bee phase position update. Sangeeta et al. [87] have suggested a magnetic navigation-based optimizer to analyze proper software reliability model parameters. Tripathi [88] has used the military Dog algorithm to estimate phone reviews and found it one of the most effective parameter estimation methods available. Sangeeta et al. [89] have created an ecological space-based hybrid swarm evolutionary method for parameter estimation in software reliability models. The authors [90] suggested a firefly-based multi-level picture thresholding method. Khan, Jabeen et al. [91] has proposed a new Metaheuristic algorithm in optimizing deep neural network model for software effort estimation. Kassaymeh et al. [92] has developed a new swarm optimizer for modeling software reliability prediction problems. Authors suggested that proposed algorithm can be applied in various optimization problems. Lakra and Chug [93] has given a study on Application of metaheuristic techniques in software quality prediction. Lilly, Jothi [94] has enhanced software reliability and fault detection process using hybrid brainstorm optimization-based LSTM Model. Kumar and Gopalan [95] have developed an efficient parameter optimization of software reliability growth model by using chaotic grey wolf optimization algorithm. Singh, Kumar et al. [96] has proposed a new Adaptive infinite impulse response system identification using teacher learner-based optimization algorithm and has analyzed good prediction on the basis of proposed algorithm. Rakhi and Pahuja [97] has developed a method for solving reliability redundancy allocation problem using grey wolf optimization algorithm. Kaushik et al. [98] has discussed about the role of neural networks and metaheuristics in agile software development effort estimation. Yadav N. and Yadav V. [99] proposed a software reliability prediction and optimization approach using machine learning algorithms.

2.3. Insights of the literature survey

From the literature in Table 1 it has been found that failure rate behavior-based models are the earliest models and these are the basis of newly proposed models. These models are studying the program failure rate per fault and are considering only the traditional fixed behavior-based waterfall approach for software development but this is a static and oldest approach. There is a need to propose new model that consider latest method of software development in software reliability analysis on the basis of Bug report analysis. Hybrid algorithms are proving to be effective in a variety of domains. These techniques can be used to estimate the parameters of software reliability models. These swarm evolutionary algorithms are found to be performing good when applied to specific application and dataset. To make their appropriate usage, a new algorithm specific to software reliability application is used in this paper. This research employed a hybrid swarm evolutionary feature-based ABC-DE method for the suggested model parameter estimation. The code for the algorithm is available in the replication package available at https://zenodo.org/records/13147825.

3. Proposed model

Failure rate behavior-based models are the oldest of the software reliability models. These have primarily been used in academics and industry to assess software product reliability. Considering the most recent software development environments and technology, a new model based on the iterative software development cycle process has been proposed. Traditional software development lifecycle methods focus on the models available in the literature. The waterfall software development lifecycle method underpins all existing models. However, modern software development approaches outperform the waterfall software development life cycle methodology. These are life cycle processes that are iterative, spiral, and adaptable. The suggested model uses the most up-to-date software development methods, which are based on the iterative software development approach. The suggested model, which is concerned with the iterative software development process, is developed from a generic collection of failure-rate based models. The proposed Iterative Software Reliability (ISR) model is derived from a general group of failure-rate models and is concerned with the iterative software development process. The failure rate model incorporates changing demands with additional capabilities in each software development iteration using the iterative factor γ that reflects varying requirements in each iteration. The changing needs in each software development iteration include a report of bugs, additional features, and the amount of testing effort. These parameters are used to identify the number of flaws in each iteration that have been either injected or eliminated in each subsequent software development iterations. The modulation factor (γ) represents all of these shifting requirements during the software development period. In future iterations, new mistakes or mutually dependent errors may be introduced. With each iteration of the software development life cycle, the suggested model implies that new defects will be inserted with a probability (p_i) and removed with (r_i) each iteration. As a result, the varied demands in each iteration are unique, and they change according to 1.

3.1. Modified reliability estimation model

3.1.1. Assumptions of proposed model

1. An iteration begins with an undetermined and constant number of software faults.
2. Each of the faults is either mutually dependent or independent, and they may equally cause a failure during testing.
3. The time period between fault occurrences is thought to be independent.
4. The program failure rate is assumed initially to be constant; however, it varies with each iteration at failure times.
5. The failure rate may increase or decrease, depending on the remaining faults in software and the modulation factor.
6. Whenever a failure occurs in an iteration, an immediately debugging effort will be initiated. Each iteration has the fault removal probability, not removed with a probability. Instead, the fault is introduced with a probability. Here, probability.
7. When a fault occurs in iteration, it may not be ideally removed. This may pass in successive iterations because the testing environment of each iteration is different

from the operating environment. Regenerated fault in subsequent iterations introduces imperfect debugging in the proposed model.

The initial iterative faults are fixed and constant but unknown for iteration. Later on, new faults may get injected from the previous iteration, or new faults may get introduced in the current iteration. Based on these faults, developers can change resource allocation in debugging iteration. The Iterative factor represents the modified need for resources and integrates the iterative SDLC process in software reliability modeling. Iterative element defined in Equation (1).

$$\gamma = \mu + (1-\mu)/\mu(0 < \mu \leq 10) \tag{1}$$

Here, μ the iterative parameter represents newly added functionality and level of user acceptance in an iteration. Its value is near zero initially and becomes almost one until the final software development iteration.

3.1.2. Model formulation

$\lambda(t_i)$, i.e., the failure rate function is modeled in Equation (2).

$$\lambda(t_i) = \phi\left[N - \frac{(n_{i-1})(\gamma)}{(i-1)}(p-r)\right], \quad i-1,2\ldots N \tag{2}$$

Here γ – iterative factor that represents varying needs in each iteration n^{i-1} – cumulative number of failures N – number of initial faults ϕ – constant of proportionality $F(t_i)$, i.e., cumulative density function and $R(t_i)$, i.e., reliability function is given in eqations (3) and (4)

$$F(t_i) = 1 - e^{\left[-\phi\left[N - \frac{(n_{i-1})\gamma}{i-1}(p-r)\right]t_i\right]} \tag{3}$$

$$R(t_i) = e^{\left[-\phi\left[N - \frac{(n_{i-1})\gamma}{i-1}(p-r)\right]t_i\right]} \tag{4}$$

3.1.3. Parameter estimation

There are three parameters N, n and ϕ, these unknown parameters are measured at different values of γ. MLE function is used to estimate the value of all parameters. The probability density function $f(t)$ for the Proposed Iterative Software Reliability (ISR) model is given in Equation (5).

$$f(t_i) = \phi\left[N - \frac{(n_{i-1})\gamma}{i-1}(p-r)\right]e^{\left[-\phi\left[N - \frac{(n_{i-1})\gamma}{i-1}(p-r)\right]t_i\right]} \tag{5}$$

Likelihood function $L(N)$ is calculated in Equation (6) using Equation (5).

$$\begin{aligned} \mathrm{L(N)} &= \prod f\left(t_i\right) \\ &= \prod_{i=1}^{n}\left[\phi\left[N-\frac{\left(n_{i-1}\right)\gamma}{i-1}(p-r)\right]e^{\left[-\phi\sum_{i=1}^{n}\left[N-\frac{\left(n_{i-1}\right)\gamma}{i-1}(p-r)\right]t_i\right]}\right. \end{aligned} \tag{6}$$

Log-likelihood function in parameter estimation needs calculation of partial derivatives w.r.t. N, n and ϕ, respectively. It then equates them to zero. Maximum likelihood estimates of parameters are obtained using (7), (8), and (9).

$$\frac{n}{\phi}-\sum_{i=1}^{n}\left[N-\frac{\left(n_{i-1}\right)y}{i-1}(p-r)\right]t \tag{7}$$

$$n=\phi\sum_{i=1}^{n}\left[N-\frac{\left(n_{i-1}\right)y}{i-1}(p-r)\right]t \tag{8}$$

$$\frac{1}{\sum_{i=1}^{n}\left[N-\frac{\left(n_{i-1}\right)y}{i-1}(p-r)\right]}-\phi\sum_{i=1}^{n}t_i \tag{9}$$

4. Experimental setup and results

4.1. Collection of bug data used in experimentation

The proposed ISR model is tested using a bug report taken from the Tera Promise repository. Datasets were preprocessed using various preprocessing techniques like outlier analysis, removal of redundant/missing values, etc. The datasets obtained after preprocessing are available at the repository https://zenodo.org/records/13147825 to encourage replication of the results. The bug report contains a table with *bug_id*, summary, description, time, associated commit, commit status, and saved files. Dataset is extracted from this file and structured in a required format. The multiple iteration dataset from this bug report is used in this paper to test the capability of proposed model to predict the remaining number of errors, reliability, and failure intensity of the OSS system.

4.2. Results

The proposed ISR model objective is to estimate the precise reliability growth of the software and includes the model parameters N, n, ϕ, γ, p and r. Probability p and r is dependent on the type of project and human skill involved in software testing. Based on these factors, the number of faults removed in testing is assumed to be 96%, and the number of faults introduced is considered as 2%. γ is defined as the new modulation factor showing the changing needs of the iterative software development process. γ values are calculated using the modulation parameter (μ). The calculated values of γ and μ with varying parameters N, n, and ϕ are estimated using the hybrid Artificial Bee Colony and Differential Evolution Algorithm (the code for the used algorithm is available in the replication package; link available at https://zenodo.org/records/13147825) to maximize the log-likelihood function value. Model parameter values are estimated using the MLE technique. 80% of data has been used to estimate parameters. The predicted number of remaining errors, i.e., the expectation, is calculated for each value of modulation factor (γ). Goodness-of-fit for the Proposed Iterative Software Reliability (ISR) model is measured using SSE, MSE, AE, and TS for each application dataset [33]. These statistics are used further for comparison between iterations of datasets. Initially, the system is assumed to be having minimum features. Later on, with time, the requirements of the end-users need to be fulfilled; they can add more and more features to the system. These additional changes in upcoming iterations may incorporate some new errors and reduce errors from the earlier iterations. Depending on these new conditions, there is a need to change the iterative system development. Accordingly, the proposed work the parameter changes its shape. In the proposed Iterative Software Reliability (ISR) model, estimated values of represent additional feature changes incorporated in the current iteration, differing from previous ones. For initial iteration releases of Eclipse and JDT open-source software systems, the value should be estimated so that a critical sample of requirements is implemented with the minor features in the system. As time passes and more and more iterations are added with changing functionality in each iteration, the value should fluctuate from lower to higher with the user acceptance and increase in functionality of the system with time. Depending on values, it is estimated for each iteration release, and the model is implemented.

4.3. Result analysis

In this section the analysis of Eclipse software and JDT software failure dataset is done using Proposed model and other failure rate behaviour models. Table 2 shows the failure rate based models used in this research work and Table 3 shows the parameter estimation using Eclipse software failure dataset on various existing models along with the proposed model.

Table 2. Summary of failure rate based software reliability models

Sr. no.	Model name	Failure intensity	No. of actual parameters
1	Jelinski–Moranda model	$2(t) = [N - (i-1)]$	2
2	Schick–Wolverton model	$2(t) = \phi[N - (i-1)]t$	2
3	Goel–Okumotto imperfect debugging model	$2(t) - \varphi[N - p(i-1)]$	3
4	G.S. Mahapatra and P. Roy model [16]	$2(t) - f[N - p(i-1) + r(i-1)]$	4
5	Modified S–W model	$2(t) = \rho\,[N - (n_{2-1})]$	3
6	Proposed Iterative Software Reliability (ISR) model	$2(t) = \phi\left[N - \dfrac{(n_{i-1})\,(p)}{(i-1)}(p-r)\right]$	6

Table 3. Parameter estimation using Eclipse software failure dataset

Model name	Parameter estimated values (fit)	Predicted number of remaining errors	
		expectation (fit)	remarks (predicted)
Iteration 1.0			
JM model	$\phi = 5.74\text{E}{-6}, N = 4$	2	over estimation
GOI model	$\phi = 1.64\text{E}{-5}, N = 3$	1	exact estimation
SW model	$\phi = 2.124\text{E}{-6}, N = 3$	1	exact estimation
Mahapatra model	$\phi = 7.77\text{E}{-7}, N = 3$	1	exact estimation
SWM model	$\phi = 2.23\text{E}{-5}, N = 2, n = 58$	0	under estimation
Proposed ISR model	$\phi = 2.86\text{E}{-5}, N = 5, n = 11, \gamma = 2.2666, \mu = 0.3419$	3	over estimation
Iteration 2.0			
JM model	$\phi = 2.68\text{E}{-5}, N = 8$	-11	under estimation
GOI model	$\phi = 2.89\text{E}{-6}, N = 28$	9	over estimation
SW model	$\phi = 4.70\text{E}{-6}, N = 28$	9	over estimation
Mahapatra model	$\phi = 2.53\text{E}{-5}, N = 23$	4	under estimation
SWM model	$\phi = 1.52\text{E}{-6}, N = 9, n = 209$	-10	under estimation
Proposed ISR model	$\phi = 4.70\text{E}{-6}, N = 27, n = 362, \gamma = 2.8763, \mu = 0.2779$	8	over estimation
Iteration 2.1			
JM model	$\phi = 5.039\text{E}{-6}, N = 27$	4	under estimation
GOI model	$\phi = 1.95\text{E}{-6}, N = 26$	3	under estimation
SW model	$\phi = 3.83\text{E}{-6}, N = 29$	6	exact estimation
Mahapatra model	$\phi = 3.20\text{E}{-6}, N = 28$	5	under estimation
SWM model	$\phi = 1.36\text{E}{-5}, N = 18, n = 316$	-5	under estimation
Proposed ISR model	$\phi = 1.67\text{E}{-6}, N = 30, n = 415, \gamma = 2.6343, \mu = 0.2999$	7	over estimation
Iteration 3.0			
JM model	$\phi = 2.40\text{E}{-6}, N = 124$	48	over estimation
GOI model	$\phi = 1.37\text{E}{-6}, N = 100$	24	over estimation
SW model	$\phi = 2.92\text{E}{-5}, N = 121$	45	over estimation
Mahapatra model	$\phi = 1.25\text{E}{-6}, N = 125$	49	over estimation
SWM model	$\phi = 1.29\text{E}{-6}, N = 123, n = 298$	47	over estimation
Proposed ISR model	$\phi = 2.98\text{E}{-5}, N = 102, n = 5690, \gamma = 3.7891, \mu = 0.2188$	26	over estimation
Iteration 3.1			
JM model	$\phi = 2.99\text{E}{-5}, N = 100$	-8	under estimation
GOI model	$\phi = 3.07\text{E}{-7}, N = 126$	18	under estimation
SW model	$\phi = 2.99\text{E}{-5}, N = 125$	17	under estimation
Mahapatra model	$\phi = 6.62\text{E}{-6}, N = 130$	22	under estimation
SWM model	$\phi = 1.36\text{E}{-5}, N = 135, n = 4979$	27	under estimation
Proposed ISR model	$\phi = 2.96\text{E}{-5}, N = 136, n = 6132, \gamma = 3.2217, \mu = 0.2519$	28	exact estimation
Iteration 3.2			
JM model	$\phi = 2.21\text{E}{-6}, N = 122$	27	over estimation
GOI model	$\phi = 8.61\text{E}{-6}, N = 106$	11	under estimation
SW model	$\phi = 5.40\text{E}{-6}, N = 90$	-5	under estimation
Mahapatra model	$\phi = 1.25\text{E}{-6}, N = 115$	20	under estimation
SWM model	$\phi = 2.08\text{E}{-6}, N = 132, n = 4108$	37	over estimation
Proposed ISR model	$\phi = 2.98\text{E}{-5}, N = 122, n = 6910, \gamma = 2.5788, \mu = 0.3055$	27	over estimation
Iteration 3.3			

Table 3 continued

JM model	$\phi = 2.95\text{E}{-5}, N = 117$	22	under estimation
GOI model	$\phi = 1.25\text{E}{-6}, N = 115$	20	under estimation
SW model	$\phi = 2.95\text{E}{-5}, N = 123$	28	over estimation
Mahapatra model	$\phi = 3.06\text{E}{-6}, N = 114$	19	under estimation
SWM model	$\phi = 5.31\text{E}{-6}, N = 121, n = 3129$	26	over estimation
Proposed ISR model	$\phi = 2.96\text{E}{-5}, N = 123, n = 7879, \gamma = 2.3121, \mu = 0.336$	28	over estimation
Iteration 3.4			
JM model	$\phi = 8.28\text{E}{-6}, N = 53$	13	over estimation
GOI model	$\phi = 3.06\text{E}{-6}, N = 50$	10	under estimation
SW model	$\phi = 2.93\text{E}{-5}, N = 54$	14	over estimation
Mahapatra model	$\phi = 1.63\text{E}{-5}, N = 55$	15	over estimation
SWM model	$\phi = 1.72\text{E}{-5}, N = 52, n = 2094$	12	over estimation
Proposed ISR model	$\phi = 2.92\text{E}{-5}, N = 53, n = 1411, \gamma = 2.7295, \mu = 0.2908$	13	over estimation
Iteration 3.5			
JM model	$\phi = 1.63\text{E}{-6}, N = 30$	10	over estimation
GOI model	$\phi = 3.81\text{E}{-6}, N = 29$	9	over estimation
SW model	$\phi = 1.70\text{E}{-7}, N = 29$	9	over estimation
Mahapatra model	$\phi = 3.81\text{E}{-6}, N = 29$	9	over estimation
SWM model	$\phi = 1.26\text{E}{-5}, N = 17, n = 2663$	-3	under estimation
Proposed ISR model	$\phi = 2.82\text{E}{-5}, N = 25, n = 416, \gamma = 2.6434, \mu = 0.2990$	5	under estimation
Iteration 3.6			
JM model	$\phi = 6.24\text{E}{-7}, N = 28$	6	exact estimation
GOI model	$\phi = 3.208\text{E}{-6}, N = 28$	6	exact estimation
SW model	$\phi = 3.74\text{E}{-6}, N = 27$	5	under estimation
Mahapatra model	$\phi = 6.65\text{E}{-7}, N = 28$	6	exact estimation
SWM model	$\phi = 2.97\text{E}{-5}, N = 17, n = 2624$	-5	under estimation
Proposed ISR model	$\phi = 2.94\text{E}{-5}, N = 30, n = 401, \gamma = 2.0435, \mu = 0.3747$	8	over estimation
Iteration 4.1			
JM model	$\phi = 2.20\text{E}{-5}, N = 19$	7	over estimation
GOI model	$\phi = 1.58\text{E}{-5}, N = 17$	5	over estimation
SW model	$\phi = 1.93\text{E}{-5}, N = 19$	7	over estimation
Mahapatra model	$\phi = 5.53\text{E}{-6}, N = 17$	5	over estimation
SWM model	$\phi = 2.68\text{E}{-5}, N = 8, n = 1529$	-4	under estimation
Proposed ISR model	$\phi = 2.87\text{E}{-5}, N = 16, n = 136, \gamma = 1.1258, \mu = 0.7026$	4	over estimation
Iteration 4.2			
JM model	$\phi = 1.92\text{E}{-5}, N = 31$	7	exact estimation
GOI model	$\phi = 3.20\text{E}{-6}, N = 30$	6	under estimation
SW model	$\phi = 6.50\text{E}{-6}, N = 31$	7	exact estimation
Mahapatra model	$\phi = 7.29\text{E}{-7}, N = 31$	7	exact estimation
SWM model	$\phi = 2.97\text{E}{-5}, N = 30, n = 3729$	6	under estimation
Proposed ISR model	$\phi = 2.82\text{E}{-5}, N = 32, n = 513, \gamma = 1.00009, \mu = 0.9904$	8	over estimation

4.3.1. Data analysis of Eclipse software

In the first released iteration of the Eclipse project, the estimated value of γ is 2.2666 and represents that the software has fewer features with the least number of bugs. A new iteration is released, more and more features are added, and value γ reflects all changing requirements incorporated in the next released iteration. The value γ reflects minute changes in minor iteration releases because it involves only a few feature addition and bug fixing. After iteration 2.0, iteration 2.1 has been released with minor feature addition and bug fixing. The value of γ this iteration is 2.8763 in iteration 2.0, but within a minute, it changes in iteration 2.1 γ, takes the value as 2.6343. This value is differing by only a small amount from the value in iteration 2.0. In major iteration release 3.0, the value is γ 3.7891, showing the major change in its value from the 2.1 version. Further in minor releases of iteration 3.0, values γ are changing as 3.2217, 2.5788, 2.3121, 2.7295, 2.6434, and 2.0435 for iteration numbers $3.1, 3.2, 3.3, 3.4, 3.5$, and 3.6, respectively. Similarly, in iteration 4.0, the γ parameter's value reflects all major iteration's requirement updates, the value γ is 1.1258, and in its minor iteration release 4.1, the value γ is 1.0009. Using estimated parameter values, prediction of the remaining number of errors is made, remarks are made whether the model overestimates, underestimates, or precisely estimates the number of faults remaining in the system.

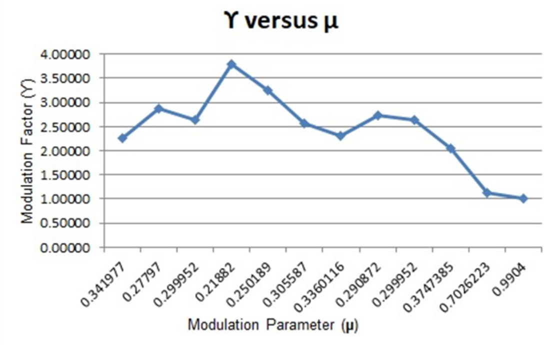


Figure 1. The plot of γ versus μ for Eclipse dataset

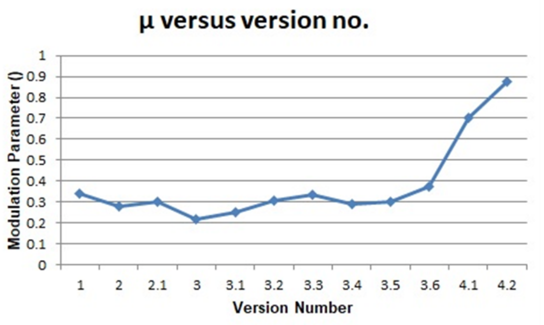


Figure 2. The plot of μ versus iteration for Eclipse dataset

Figure 1 represents how the modulation factor values change with respect to the modulation parameter for each iteration. Estimated values of the parameter are significantly reflecting all the changing requirements for each upcoming iteration numerically. These values meaningfully represent how much impact is of adding and removing new features with user acceptance in each upcoming iteration.

Figure 2 represent how the value of changes increases with the growth of the iterative software development process.

Table 4. Goodness-of-fit estimated using Eclipse software failure dataset

Sr. No.	Model	SSE	MSE	AE	TS
Iteration 1.0					
1	JM model	5.23	5	0.333	59.761
2	GOI model	1.51	1	0.0189	**26.721**

Table 4 continued

3	SW model	17.31	17	0.666	110.19
4	Mahapatra model	1.26	1	0.013	26.722
5	SWM model	5	4.9	0.297	65.94
6	**Proposed ISR model**	**1.24**	**0.666**	**0.012**	37.79
Iteration 2.0					
1	JM model	375.51	17.045	0.225	27.664
2	GOI model	92.005	4.181	**0.125**	13.702
3	SW model	218.81	9.909	0.266	21.092
4	Mahapatra model	55.46	2.51	0.166	12.371
5	SWM model	528.7	15.89	0.495	34.57
6	**Proposed ISR model**	**45.12**	**2.5**	**0.125**	**10.594**
Iteration 2.1					
1	JM model	452.2	16.74	0.054	22.985
2	GOI model	133.34	4.925	0.035	12.468
3	SW model	126.6	4.666	0.034	**12.136**
4	Mahapatra model	131.31	4.851	0.039	12.374
5	SWM model	118.67	**3.89**	0.028	13.9
6	**Proposed ISR model**	**101.02**	4.391	**0.023**	14.42
Iteration 3.0					
1	JM model	56948	605.82	0.302	43.602
2	GOI model	45510.1	559.22		38.978
3	SW model	39981	494.69	0.292	**36.507**
4	Mahapatra model	55122.2	586.4	0.395	42.898
5	SWM model	66890.4	679.32	**0.279**	54.89
6	**Proposed ISR model**	**39705.2**	**484.14**	0.584	52.069
Iteration 3.1					
1	JM model	**6091.1**	45.45	0.016	8.476
2	GOI model	1018.8	7.59	0.005	**3.465**
3	SW model	1123	8.38	0.0065	3.639
4	Mahapatra model	1015	**7.57**	**0.005**	3.56
5	SWM model	2000.7	15.9	0.006	5.69
6	**Proposed ISR model**	1589.82	12.22	0.007	4.551
Iteration 3.2					
1	JM model	13940.73	119.14	0.05	15.655
2	GOI model	7246.05	61.931	0.042	11.287
3	SW model	12585	107.564	0.111	14.871
4	Mahapatra model	7794.28	68.973	0.107	12.416
5	SWM model	9685.12	89.67	0.357	20.85
6	**Proposed ISR model**	**952.2**	**8.136**	**0.034**	**4.091**
Iteration 3.3					
1	JM model	6810.14	58.239	0.025	10.945
2	GOI model	9371.1	80.094	0.042	12.835
3	SW model	7318.81	62.547	0.034	11.342
4	Mahapatra model	2217.4	**18.948**	0.05	9.243
5	SWM model	2903.9	27.872	0.068	12.09
6	**Proposed ISR model**	**2203.05**	19.495	**0.025**	**7.099**
Iteration 3.4					
1	JM model	1153.32	23.53	0.058	50.122
2	GOI model	1545.41	1320.9	0.546	52.124
3	SW model	1153.73	23.53	0.049	15.914
4	Mahapatra model	905.4	18.46	0.058	14.099
5	SWM model	3589.96	29.79	0.089	34.956
6	**Proposed ISR model**	**520.23**	**11.55**	**0.039**	**12.809**

Table 4 continued

Iteration 3.5					
1	JM model	373.3	15.542	0.153	24.526
2	GOI model	137.74	5.7083	0.076	14.864
3	SW model	551.1	22.958	0.152	29.808
4	Mahapatra model	137.8	5.708	0.077	14.864
5	SWM model	489.74	18.24	0.0204	24.95
6	**Proposed ISR model**	**86.62**	**4.3**	**0.077**	**12.88**
Iteration 3.6					
1	JM model	199.91	7.654	0.036	16.064
2	GOI model	160.01	6.154	0.071	14.402
3	SW model	154.54	5.923	0.056	14.129
4	Mahapatra model	152.23	5.864	r|0.046	14.037
5	SWM model	315	14.92	0.184	23.68
6	**Proposed ISR model**	**91.15**	**4.136**	**0.035**	**14.669**
Iteration 4.1					
1	JM model	286.65	22	0.266	48.025
2	GOI model	111.21	8.538	0.256	29.91
3	SW model	284.86	22.1	0.266	48.025
4	Mahapatra model	129	9.923	0.133	32.254
5	SWM model	178.22	7.34	0.0836	**18.48**
6	**Proposed ISR model**	**49.95**	**5.444**	**0.246**	26.335
Iteration 4.2					
1	JM model	441.12	15.206	0.323	20.576
2	GOI model	121.21	4.172	0.24	10.778
3	SW model	193.34	6.655	0.289	13.612
4	Mahapatra model	166.65	5.724	0.224	12.624
5	SWM model	704.86	26.87	0.593	49.056
6	**Proposed ISR model**	**72.235**	**2.88**	**0.129**	**10.688**

The Goodness-of-fit of the models is shown in Table 4. From the analysis of results, it is found that the Proposed ISR model is fitted well in terms of SSE, MSE, and AE in iterations 1.0 and 2.1. But GOI model is beating the Proposed Iterative Software Reliability (ISR) model in terms of TS in iteration number 1.0 of the Eclipse software failure dataset. In iteration 2.0 Proposed ISR model is beating other models in terms of SSE and AE but also performing good. SW, and SWM are performing better than the proposed ISR model. In iteration 3.0, the Proposed Iterative Software Reliability (ISR) model performs better than other SSE and MSE, but its AE and TS values are exhausted by other models. In iteration number 3.1 Proposed Iterative Software Reliability (ISR) model is found to be less fitted than other models. Again, in further iterations of 3.2, 3.3, 3.4, 3.5, 3.6, and 4.2, the Proposed ISR) model estimates parameters more precisely than other models. In iteration number 3.3 and 4.1, the Proposed ISR model is not much good than the Mahapatra model in terms of MSE and TS. But they are performing a good fit than other models in terms of SSE, MSE and TS. From all the data analysis done in Table 4, it is found that the Proposed Iterative Software Reliability (ISR) model is performing fighting fit in terms of various Goodness-of-fit criteria and can be used for predicting the reliability of system software.

Optimal iteration releases can be selected based on reliability. The iterations 3.3, 3.6, and 4.1 are found to have higher reliability, and there are fewer falls in reliability values of these versions than other released iteration reliabilities. All iteration's reliability values are found to be decreasing up to 89%. Failure intensity graphs in Figure 4 show how

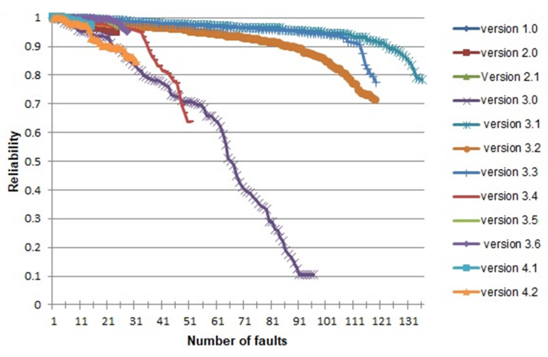


Figure 3. The plot of the reliability function for various iterations of the Eclipse failure dataset

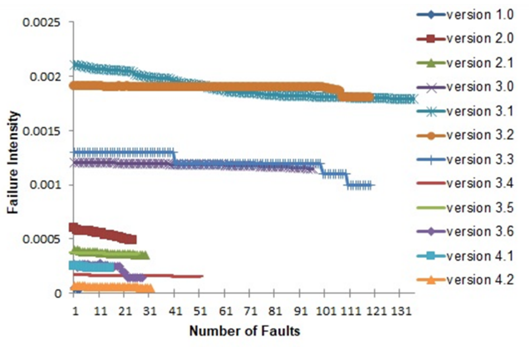


Figure 4. Plot of failure Intensity for various iterations for Eclipse failure dataset

much there is a decrease in failure behavior as the number of fault count increases. The iterations 3.3, 3.6, and 4.1 releases have a much-decreasing failure rate. There decrease in failure rate behavior is depicting that with time, the reliability of these systems increases. Reliability analysis has been done in Figure 3 and failure Intensity has been shown in Figure 4. Developers could decide the quality of the released iteration using estimated reliability. This is also helpful for end-users in choosing which iteration release will be most reliable in the future.

4.3.2. Result and data analysis of JDT dataset

In this section, analysis of the JDT software failure dataset is done using the proposed Iterative Software Reliability (ISR) model and other failure rate behavior-based models. Table 5 is showing the estimated values of model parameters.

Table 5. Parameter estimation using JDT software failure dataset

Model Name	Parameter estimated values (fit)	Predicted number of remaining errors	
		expectation (fit)	remarks (predicted)
Iteration 3.2.0			
JM model	$\phi = 2.35\text{E}{-6}, N = 50$	3	under estimation
GOI model	$\phi = 8.16\text{E}{-7}, N = 56$	9	under estimation
SW model	$\phi = 2.17\text{E}{-6}, N = 58$	11	under estimation
Mahapatra model	$\phi = 1.21\text{E}{-6}, N = 57$	10	under estimation
SWM model	$\phi = 2.91\text{E}{-5}, N = 52, b = 1723$	5	under estimation
Proposed ISR model	$\phi = 2.91\text{E}{-5}, N = 60, n = 1788, \gamma = 4.8479, \mu = 0.1710$	13	over estimation
Iteration 3.3.0			
JM model	$\phi = 2.78\text{E}{-5}, N = 9$	−5	under estimation
GOI model	$\phi = 1.48\text{E}{-5}, N = 16$	2	under estimation
SW model	$\phi = 4.70\text{E}{-6}, N = 8$	−6	under estimation
Mahapatra model	$\phi = 8.92\text{E}{-6}, N = 26$	12	over estimation
SWM model	$\phi = 2.99\text{E}{-5}, N = 9, n = 1028$	−5	under estimation
Proposed ISR model	$\phi = 2.97\text{E}{-5}, N = 19, n = 179, \gamma = 3.8869, \mu = 0.2140$	5	over estimation

Table 5 continued

Iteration 3.4.0			
JM model	$\phi = 8.86\text{E}{-6}, N = 18$	7	over estimation
GOI model	$\phi = 2.35\text{E}{-6}, N = 18$	7	over estimation
SW model	$\phi = 3.83\text{E}{-6}, N = 16$	5	over estimation
Mahapatra model	$\phi = 1.03\text{E}{-5}, N = 28$	17	over estimation
SWM model	$\phi = 2.91\text{E}{-5}, N = 13, n = 1124$	2	under estimation
Proposed ISR model	$\phi = 2.92\text{E}{-5}, N = 18, n = 119, \gamma = 2.9993,$ $\mu = 0.2680$	7	over estimation
Iteration 3.5.0			
JM model	$\phi = 2.49\text{E}{-5}, N = 8$	4	over estimation
GOI model	$\phi = 1.06\text{E}{-6}, N = 13$	9	over estimation
SW model	$\phi = 2.92\text{E}{-5}, N = 10$	6	over estimation
Mahapatra model	$\phi = 4.78\text{E}{-7}, N = 5$	1	exact estimation
SWM model	$\phi = 2.31\text{E}{-5}, N = 3, n = 137$	-1	under estimation
Proposed ISR model	$\phi = 2.96\text{E}{-5}, N = 7, n = 25, \gamma = 2.0918,$ $\mu = 0.3670$	3	over estimation
Iteration 3.6.0			
JM model	$\phi = 2.93\text{E}{-5}, N = 4$	-8	under estimation
GOI model	$\phi = 2.01\text{E}{-5}, N = 11$	-1	under estimation
SW model	$\phi = 2.99\text{E}{-5}, N = 29$		over estimation
Mahapatra model	$\phi = 1.26\text{E}{-5}, N = 85$	13	over estimation
SWM model	$\phi = 1.34\text{E}{-5}, N = 15, n = 134$	3	exact estimation
Proposed ISR model	$\phi = 2.67\text{E}{-5}, N = 18, n = 171, \gamma = 1.1529,$ $\mu = 0.678$	6	over estimation
Iteration 3.7.0			
JM model	$\phi = 1.50\text{E}{-6}, N = 16$	7	over estimation
GOI model	$\phi = 1.58\text{E}{-6}, N = 13$	4	over estimation
SW model	$\phi = 5.40\text{E}{-6}, N = 26$	17	over estimation
Mahapatra model	$\phi = 5.30\text{E}{-6}, N = 12$	3	exact estimation
SWM model	$\phi = 2.69\text{E}{-5}, N = 14, n = 1599$	5	over estimation
Proposed ISR model	$\phi = 2.74\text{E}{-5}, N = 15, n = 78, \gamma = 1.052,$ $\mu = 0.724$	6	over estimation

A predicted number of errors estimated using the Proposed Iterative Software Reliability (ISR) model is found to be representing accurate estimation when compared with the actual dataset values. There is a slight deviation only because of the poor debugging behavior in the iterative software development process. Remarks are made about whether the model underestimates, overestimates, or precisely estimates the number of remaining errors in the system.

Figures 5 and 6 are showing γ and μ behavior with respect to change in iterations. With each iteration, the value of functionality changes and corresponding to its requirements are made in upcoming iterations. Accordingly value of $\gamma\mu$ and changes their shape quantitatively, as shown in Figure 5. Figure 6 is showing the behavior of μ with change in iteration numbers. At initial iteration number 3.2, there are the least features and maximum changing needs from the end-users; the value γ μ are 4.8479 and 0.171, respectively. These values are varying in the same way in each iteration release. For example, in iteration 3.3, the value γ is 3.8869 and μ is 0.2140. For iteration 3.4 value γ is 2.9993, and the value μ is 0.2680 showing perfectly how the requirements and functionality change in each iteration with end-user acceptance. In the end, γ values move towards completion of end-user requirements, and acceptance level also increases, the value of γ the changes as 1.1052 and μ as 0.7240.

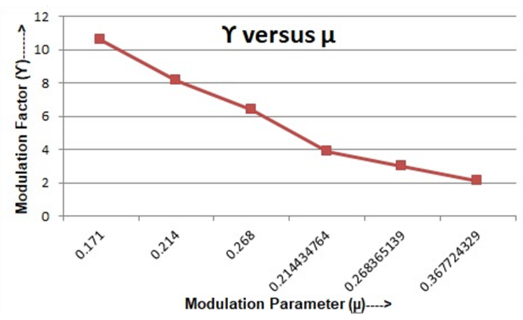


Figure 5. Plot of γ versus μ for JDT dataset

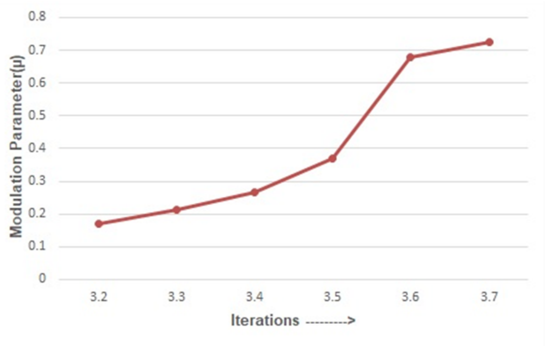


Figure 6. The plot ofμ versus Iteration for JDT dataset

Figures 5 and 6 are showing well, the iterative behavior of software development using γ and μ values.

Table 6. Goodness-of-fit of JDT software failure dataset

Sr. No.	model	SSE	MSE	AE	TS
1	JM model	179.79	11.187	0.0169	29.133
2	GOI model	317.67	5.561	**0.011**	6.719
3	SW model	1056	15.678	0.018	19.083
4	Mahapatra model	56500.09	991.228	0.915	89.706
5	SWM model	2989.06	29.689	0.0214	41.075
6	**Proposed ISR model**	**171.89**	**3.226**	0.0169	**5.585**
Iteration 3.3.0					
1	JM model	404	33.666	0.285	63.089
2	GOI model	989.9	61.825	0.666	68.479
3	SW model	589.1	40.78	0.39	69.93
4	Mahapatra model	604.01	37.75	0.388	53.515
5	SWM model	486.32	33.96	0.31	65.94
6	**Proposed ISR model**	**31**	**2.583**	**0.114**	**16.295**
Iteration 3.4.0					
1	JM model	37	12.33	0.6	82.02
2	GOI model	206.56	17.16	0.285	45.051
3	SW model	750.9	68	0.301	90.789
4	Mahapatra model	1241.01	103.41	**1.112**	110.574
5	SWM model	1839.56	146.92	1.947	130.05
6	**Proposed ISR model**	**91**	**11.37**	0.214	**3.365**
Iteration 3.5.0					
1	JM model	57	4.384	0.066	**21.44**
2	GOI model	76.53	25.33	1.6	117.551
3	SW model	3019.9	119.05	4.037	289.68
4	Mahapatra model	38225	318.54	8.5	613.678
5	SWM model	20687	156.94	5.923	493.8
6	**Proposed ISR model**	**18**	**-24.002**	**0.2**	66.057
Iteration 3.6.0					
1	JM model	76	7.6	0.333	**34.194**
2	GOI model	46721	359.39	6.6	613.8259
3	SW model	12089	367.83	17.89	156.9
4	Mahapatra model	11534	384.477	16.001	144.8
5	SWM model	34834.09	329.21	5.093	542.056

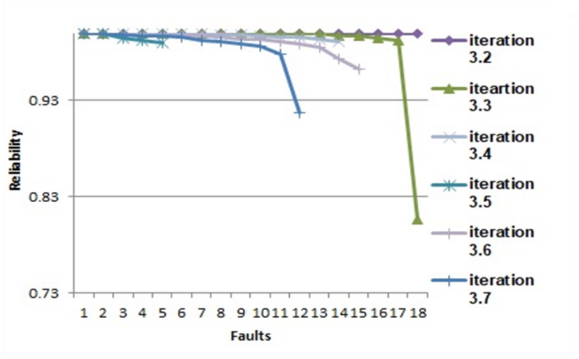


Figure 7. The plot of reliability versus number of faults for various iterations on JDT dataset

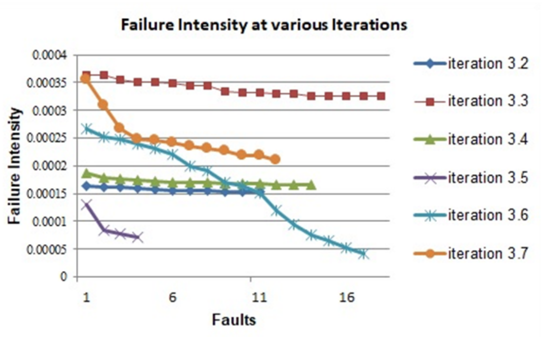


Figure 8. The plot of failure intensity versus faults for various iterations on JDT dataset

Table 6 continued

Sr. No.	model	SSE	MSE	AE	TS
6	**Proposed ISR model**	**192**	**21.3333**	**0.2**	44.45
Iteration 3.7.0					
1	JM model	280	45002	**0.071**	35.133
2	GOI model	90.011	9.001	0.166	37.21
3	SW model	200.9	11.08	0.189	49.057
4	Mahapatra model	952	**8.1368**	2.478	40.091
5	SWM model	1109.23	57.34	0.373	169.78
5	**Proposed ISR model**	**54**	9.0001	0.166	**25.5181**

Table 6 shows the Goodness-of-fit measures of the Proposed Iterative Software Reliability (ISR) model and other models. The iteration number 3.2 performs better than other models except in terms of AE compared to the GOI model. The Proposed Iterative Software Reliability (ISR) model is estimating values in a best fitted way than other models in iteration 3.3. In case of iteration 3.4 for proposed Iterative Software Reliability (ISR) model, AE value is comparatively higher than the Mahapatra model, but it performs well in other cases. For iteration number 3.5, the JM model beats the Proposed Iterative Software Reliability (ISR) model in terms of TS. In Iteration number 3.6, the proposed Iterative Software Reliability (ISR) model works well, compared to other models, except that it deviates well from the JM model at SSE, MSE, and TS values. For iteration number 3.7 proposed Iterative Software Reliability (ISR) model is fighting with all other models except at MSE and AE, where there is a bond in its performance with other models. Overall proposed Model is fitted well for estimating the parameters of the models and can be used for estimating the reliability of JDT systems.

The Proposed Iterative Software Reliability (ISR) model's reliability analysis using the JDT dataset is done in Figure 7. Failure intensity in Figure 8 is depicting the iteration's failure rate at each failure interval. There is a decreasing rate of failure at each iteration. In the initial iteration of JDT software, the failure rate decreases with low intensity; later on, when most bugs have been removed and almost all functionalities have been added, the iteration failure rate starts falling at high intensity. For example, in iteration numbers 3.6 and 3.7, there is a more decreasing failure rate than other iterations. This decreasing failure rate depicts that there is always an increase in end-user acceptance level and reliability with added functionalities and bug removal in each successive iteration. In the final iteration release, the software is at its full implementation of functionalities and ready for conclusive acceptance from the end-users.

4.3.3. Statistical evaluation

From the above discussion, we concluded that the proposed ISR model outperformed all the other models for both datasets, Eclipse and JDT. To statistically evaluate the results, we have used a non – parametric Friedman test. It is used to evaluate whether statistical difference also exist among various models when applied over multiple iterations. For Friedman test, the following hypotheses are created and then the value of χ^2 is used to test the null hypothesis:

Null Hypothesis (H_0): There is no statistical difference between the performances of the compared models (JM model, GOI model), SW model, Mahapatra model, SWM model and the proposed ISR model). Alternative Hypothesis (H1): There exists statistical difference between the performances of the compared Models (JM model, GOI model), SW model, Mahapatra model, SWM model and the proposed ISR model). Friedman test was applied on all the performance measures for both the datasets. The *p*-values and the chi-square values given by Friedman test in shown in Table 7. From the Table 7, we can observe that for all the cases except one, the *p*-values proved that the results are significant at the 0.05 level of significance over 5 degrees of freedom. Since, the *p*-value is less than the significance level of 0.05 (shown in bold), the null hypothesis is rejected and alternate hypothesis is accepted. Thus, we say that there is statistical significant difference in the performance of the compared models. The ranks obtained by each model when using SSE, MSE, AE and TS are shown in Tables 8 and 9 for Eclipse and JDT datasets, respectively. The rank demonstrates the performance of all the models (lowest numerical rank value shows the highest performance). From Tables 8 and 9, we can observe that the top rank (shown in bold) is obtained by the proposed ISR model is all the cases for both the datasets. This demonstrates that the proposed ISR model is significantly better than the other compared models with respect to all the performance measures.

Table 7. Friedman results (*p*-value and χ^2 value)

	SSE		MSE		AE		TS	
Dataset	*p* value	χ^2	*p* value	χ^2	*p* value	χ^2	*p* value	χ^2
Eclipse	**0**	35.048	**0**	30.431	0.063	10.483	**0**	22.464
JDT	**0.001**	20	**0.013**	14.476	**0.003**	18.341	**0.004**	17.524

Table 8. Mean ranks obtained using Friedman test on Eclipse dataset

Models	SSE	MSE	AE	TS
JM model	5	5.04	4.29	4.79
GOI model	3.25	3.46	3.17	2.88
SW model	4.17	4.21	4.13	3.63
Mahapatra model	2.58	2.54	3.29	2.38
SWM model	4.75	4.33	3.92	4.92
Proposed ISR model	**1.25**	**1.42**	**2.21**	**2.42**

Table 9. Mean ranks obtained using Friedman test on JDT dataset

Models	SSE	MSE	AE	TS
JM model	2	2.67	2.08	2.33
GOI model	3.83	3.5	3.08	3.5
SW model	3.83	4.33	4.33	4.33
Mahapatra model	5	4.67	5.33	4.33
SWM model	5	4.5	4.5	5.17
Proposed ISR model	**1.33**	**1.33**	**1.67**	**1.33**

4.4. Discussion

The suitability of the model is shown on time-domain data sets. The Proposed Iterative Software Reliability (ISR) model is more adaptive to observed time-domain failure data sets than other failure rate models. Adaptation has been made possible because of the modulation parameter μ used in the model. As the software development moves towards completion, this parameter changes its values according to added functionalities and user acceptance level in each successive iteration of software development. This parameter assumes that there is added functionality and user acceptance in each iterative development cycle. According to the functionality added in each iteration, there is a change in the requirements of iterative software development. The users' varying needs are reflected with modulation factors in the Proposed Iterative Software Reliability (ISR) model. The Proposed Iterative Software Reliability (ISR) model provides a good fit for the observed failure data. Values estimated in Tables 3 and 5 shows acceptable parameter values at all weights γ. With each iteration, functionality values and user acceptance increase (demonstrated by the value of modulation parameter that increases from lower to higher). There is a corresponding change in needs for iteration development (shown by modulation factor). Values of modulation factor and modulation parameter change suitably with each upcoming iteration in both the failure datasets, demonstrating well the iterative software development behavior. Depending on the values of the estimated parameters, the number of remaining faults in terms of expectation is calculated. In some cases, the prediction deviates by a small amount due to the introduction and removal of mutually dependent and independent faults in each iteration. This verifies the imperfect debugging phenomenon associated with each iterative software development. The Goodness-of-fit for models is shown in Tables 4 and 6 regarding SSE, MSE, AE, and TS values. The Proposed Iterative Software Reliability (ISR) model fits well in both the failure datasets used. The Proposed Iterative Software Reliability (ISR) model has outstanding performance in data analysis of both the failure datasets than JM, GOI, SW, Mahapatra et al. and SWM models in terms of Goodness values. In all iterations of datasets, Proposed Iterative Software Reliability (ISR) model reliability increases as several iterations proceeds. The last iteration is assumed to be the reliable iteration. This change in reliability is showing a value-added software development process. All models have been implemented using a hybrid PSO-GSA algorithm for parameter estimation. The Proposed Iterative Software Reliability (ISR) model shows considerable performance with a hybrid algorithm even with more parameters than other models.

5. Threats to validity

In this section, we discuss potential threats which may affect the findings of the study. Conclusion validity threat in the proposed work is mitigated after evaluating the statistical viability of results with the use of two statistical tests. Researchers in this work have proposed a new Software reliability estimation model for reliability estimation of Eclipse and JDT software's. Proposed work has been compared with most relevant models available in the literature. Although no models is well fitted to all the failure datasets, each model is having its own assumptions and it works significantly well mainly for that type of dataset only for which it has been developed. Although other model comparison is done in the proposed research here for Eclipse and JDT software datasets and selection of most optimal model was not the primary aim of the study, this threat exists. However, to reach to a useful set of parameters, all the possible combinations of various parameters need to be explored. Parameter estimation plays an important role in the performance model development. Model parameters have been estimated and well pruned using hybrid swarm evolutionary algorithm. Another important factor which we considered was the use of stable performance metrics as Sum of Squared Errors, Mean Square Error, Absolute Error and Theil statistics to deal with the imbalanced nature of the datasets and models. Finally, the use of different datasets and models based on different assumptions also helps to reduce the conclusion validity threat. It may be noted that the models developed in the study can be used to detect software reliability at different iterations of software development. However, further studies should be conducted for analyzing the suitability of proposed model for determining change and acceptance levels at levels of iterations in the software development. This acts as a limitation to the applicability of the reported results. External validity threat concerned with the generalizability and replicability of the results is reduced in the proposed work due to the use of popular Tera Promise repository datasets used in this study which are freely available for the researchers to replicate the result, its findings and to conduct additional research. However more datasets belonging to different domains and applications need to be considered to ensure generalizability of the results. Further varied percentage of change-prone parameters of proposed model using hybrid swarm evolutionary algorithm across different datasets guarantees generalizability of the results. However, the study investigated parameter estimation using only hybrid swarm evolutionary algorithm only. Future experiments which involve more parameter estimation techniques likely using Machine learning or deep learning should be conducted to enhance the generalizability of the obtained results.

6. Conclusion and future scope

The majority of existing failure rate models are based on the classic waterfall software development lifecycle process model. However, new software development techniques, such as iterative life cycle processes, have been created and shown to be more effective than waterfall software development lifecycle processes. The Proposed Iterative Software Reliability (ISR) model makes use of the iterative nature of software development. The Proposed Iterative Software Reliability (ISR) model uses a modulation factor to represent the changing demands in each iteration. Software development features reduce at the end; accordingly modulation factor changes its value. Debugging is never perfect, so imperfect debugging has been incorporated by assuming fault introduction and removal. A realistic iterative software development feature has been added to the Proposed Iterative Software Reliability (ISR) model perfectly and accurately. The Goodness-of-fit measure of the models is done using SSE, MSE, AE, and TS values. The Proposed Iterative Software Reliability (ISR) model fits all application datasets for each analysis and fulfils all the authentic assumptions during iterative software development. The proposed model has major beating SSE, MSE, AE, and TS values than JM, GO, SW, SWM, and Mahapatra et al. models when applied on both the datasets. According to various Goodness-of-fit measures, the proposed model is about 52% significant from other models in the case of Eclipse failure data analysis for iteration 1.0. It performs by about 30% using 2.0 to 2.1 and 3.0 to 3.6 iterations. Similarly proposed model has about 35% higher goodness than other used models. When the JDT software failure dataset is analyzed for Proposed Iterative Software Reliability (ISR) model fitness, the proposed model beats other failure rate models by about 55%. In each analysis done, the Proposed Iterative Software Reliability (ISR) model is found to be reaching acceptable performance and could be applied on other software failure datasets for further validation. Other reasonable criteria for finding Goodness-of-fit for the model can be developed, and a method for finding upper and lower bounds for estimation can be proposed in the future.

Data availability

Both the code for the algorithm and the datasets obtained after preprocessing are vailable in the replication package available at https://zenodo.org/records/13147825.

References

[1] S. Kumar and P. Ranjan, "A phase wise approach for fault identification," *Journal of Information and Optimization Sciences*, Vol. 39, No. 1, 2018, pp. 223–237.

[2] *IEEE standard glossary of software engineering terminology (IEEE Std 610.12-1990)*, IEEE Computer Society Std., 1990. [Online]. https://ieeexplore.ieee.org/document/159342

[3] P. Kapur, H. Pham, A.G. Aggarwal, and G. Kaur, "Two dimensional multi-release software reliability modeling and optimal release planning," *IEEE Transactions on Reliability*, Vol. 61, No. 3, 2012, pp. 758–768.

[4] C.Y. Huang and M.R. Lyu, "Optimal release time for software systems considering cost, testing-effort, and test efficiency," *IEEE Transactions on Reliability*, Vol. 54, No. 4, 2005, pp. 583–591.

[5] D. Greer and G. Ruhe, "Software release planning: An evolutionary and iterative approach," *Information and Software Technology*, Vol. 46, No. 4, 2004, pp. 243–253.

[6] A.L. Goel and K. Okumoto, "Time-dependent error-detection rate model for software reliability and other performance measures," *IEEE Transactions on Reliability*, Vol. 28, No. 3, 1979, pp. 206–211.

[7] H. Pham, *System software reliability.* Springer Science and Business Media, 2007.

[8] K. Sahu and R. Srivastava, "Revisiting software reliability," *Data Management, Analytics and Innovation*, 2019, pp. 221–235.

[9] Z. Jelinski and P. Moranda, "Software reliability research," in *Statistical computer performance evaluation.* Elsevier, 1972, pp. 465–484.

[10] G.J. Schick and R.W. Wolverton, "An analysis of competing software reliability models," *IEEE Transactions on Software Engineering*, Vol. 2, 1978, pp. 104–120.

[11] J. Xavier, A. Macêdo, R. Matias, and L. Borges, "A survey on research in software reliability engineering in the last decade," in *Proceedings of the 29th Annual ACM Symposium on Applied Computing*, 2014, pp. 1190–1191.

[12] A.N. Sukert, "An investigation of software reliability models," in *Annual Reliability and Maintainability Symposium, Philadelphia, Pa*, 1977, pp. 478–484.
[13] P.B. Moranda, "An error detection model for application during software development," *IEEE Transactions on Reliability*, Vol. 30, No. 4, 1981, pp. 309–312.
[14] B. Littlewood and A. Sofer, "A Bayesian modification to the Jelinski–Moranda software reliability growth model," *Software Engineering Journal*, Vol. 2, No. 2, 1987, pp. 30–41.
[15] Z. Luo, P. Cao, G. Tang, and L. Wu, "A modification to the Jelinski-Moranda software reliability growth model based on cloud model theory," in *Seventh International Conference on Computational Intelligence and Security*. IEEE, 2011, pp. 195–198.
[16] G. Mahapatra and P. Roy, "Modified Jelinski–Moranda software reliability model with imperfect debugging phenomenon," *International Journal of Computer Applications*, Vol. 48, No. 18, 2012, pp. 38–46.
[17] X. Zhang, X. Teng, and H. Pham, "Considering fault removal efficiency in software reliability assessment," *IEEE Transactions on Systems, Man, and Cybernetics* – Part A*: Systems and Humans*, Vol. 33, No. 1, 2003, pp. 114–120.
[18] J.D. Musa and K. Okumoto, "A logarithmic Poisson execution time model for software reliability measurement," in *Proceedings of the 7th International Conference on Software Engineering*. Citeseer, 1984, pp. 230–238.
[19] C.Y. Huang, M.R. Lyu, and S.Y. Kuo, "A unified scheme of some nonhomogenous poisson process models for software reliability estimation," *IEEE Transactions on Software Engineering*, Vol. 29, No. 3, 2003, pp. 261–269.
[20] N. Pavlov, G. Spasov, A. Rahnev, and N. Kyurkchiev, "A new class of Gompertz-type software reliability models," *International Electronic Journal of Pure and Applied Mathematics*, Vol. 12, No. 1, 2018, pp. 43–57.
[21] P. Roy, G. Mahapatra, and K. Dey, "An NHPP software reliability growth model with imperfect debugging and error generation," *International Journal of Reliability, Quality and Safety Engineering*, Vol. 21, No. 2, 2014, p. 1450008.
[22] V. Singh, M. Sharma, and H. Pham, "Entropy based software reliability analysis of multi-version open source software," *IEEE Transactions on Software Engineering*, Vol. 44, No. 12, 2017, pp. 1207–1223.
[23] X. Wei, Y. Dong, X. Li, and W.E. Wong, "Architecture-level hazard analysis using AADL," *Journal of Systems and Software*, Vol. 137, 2018, pp. 580–604.
[24] V.R. Basil and A.J. Turner, "Iterative enhancement: A practical technique for software development," *IEEE Transactions on Software Engineering*, Vol. 4, 1975, pp. 390–396.
[25] J. Erickson, K. Lyytinen, and K. Siau, "Agile modeling, agile software development, and extreme programming: The state of research," *Journal of Database Management*, Vol. 16, No. 4, 2005, pp. 88–100.
[26] C. Larman and V.R. Basili, "Iterative and incremental developments. A brief history," *Computer*, Vol. 36, No. 6, 2003, pp. 47–56.
[27] N. Kerzazi and F. Khomh, "Factors impacting software release engineering: A longitudinal study," in *2nd International Workshop on Release Engineering*, 2014, pp. 1–5.
[28] P. Kapur and R. Garg, "Optimal release policies for software systems with testing effort," *International Journal of Systems Science*, Vol. 22, No. 9, 1991, pp. 1563–1571.
[29] H. Pham and H. Pham, "Software reliability modeling," *System Software Reliability*, 2006, pp. 153–177.
[30] I.J. Myung, "Tutorial on maximum likelihood estimation," *Journal of Mathematical Psychology*, Vol. 47, No. 1, 2003, pp. 90–100.
[31] M.N. Ab Wahab, S. Nefti-Meziani, and A. Atyabi, "A comprehensive review of swarm optimization algorithms," *PLOS ONE*, Vol. 10, No. 5, 2015, p. e0122827.
[32] X.S. Yang, *Nature-inspired metaheuristic algorithms*. Luniver Press, 2010.
[33] K. Sharma, R. Garg, C. Nagpal, and R.K. Garg, "Selection of optimal software reliability growth models using a distance based approach," *IEEE Transactions on Reliability*, Vol. 59, No. 2, 2010, pp. 266–276.
[34] M. Xie, *Software reliability modelling*, Vol. 1. World Scientific, 1991.

[35] M. Ohba, "Inflection S-shaped software reliability growth model," in *Stochastic Models in Reliability Theory*. Springer, 1984, pp. 144–162.

[36] S. Yamada, M. Ohba, and S. Osaki, "S-shaped reliability growth modeling for software error detection," *IEEE Transactions on reliability*, Vol. 32, No. 5, 1983, pp. 475–484.

[37] S. Yamada, H. Ohtera, and H. Narihisa, "Software reliability growth models with testing-effort," *IEEE Transactions on Reliability*, Vol. 35, No. 1, 1986, pp. 19–23.

[38] S. Yamada, H. Ohtera, and M. Ohba, "Testing-domain dependent software reliability models," *Computers and Mathematics with Applications*, Vol. 24, No. 1–2, 1992, pp. 79–86.

[39] J. Xiang, F. Machida, K. Tadano, and Y. Maeno, "An imperfect fault coverage model with coverage of irrelevant components," *IEEE Transactions on Reliability*, Vol. 64, No. 1, 2014, pp. 320–332.

[40] P. Kapur and S. Younes, "Modelling an imperfect debugging phenomenon in software reliability," *Microelectronics Reliability*, Vol. 36, No. 5, 1996, pp. 645–650.

[41] B. Littlewood and J.L. Verrall, "A Bayesian reliability growth model for computer software," *Journal of the Royal Statistical Society: Series C (Applied Statistics)*, Vol. 22, No. 3, 1973, pp. 332–346.

[42] T. Mazzuchi and R. Soyer, "A Bayes empirical–Bayes model for software reliability," *IEEE Transactions on Reliability*, Vol. 37, No. 2, 1988, pp. 248–254.

[43] H. Pham, *Springer Handbook of Engineering Statistics*. Springer Nature, 2023.

[44] B. Littlewood, "Software reliability model for modular program structure," *IEEE Transactions on Reliability*, Vol. 28, No. 3, 1979, pp. 241–246.

[45] S. Yamada, K. Tokuno, and Y. Kasano, "Quantitative assessment models for software safety/reliability," *Electronics and Communications in Japan (*Part II*: Electronics)*, Vol. 81, No. 5, 1998, pp. 33–43.

[46] A.L. Goel and K. Okumoto, "A Markovian model for reliability and other performance measures of software systems," in *International Workshop on Managing Requirements Knowledge (MARK)*. IEEE, 1979, pp. 769–774.

[47] J. Shanthikumar, "A general software reliability model for performance prediction," *Microelectronics Reliability*, Vol. 21, No. 5, 1981, pp. 671–682.

[48] W.S. Jewell, "Bayesian extensions to a basic model of software reliability," *IEEE Transactions on Software Engineering*, Vol. 12, 1985, pp. 1465–1471.

[49] H. Joe and N. Reid, "On the software reliability models of Jelinski–Moranda and Littlewood," *IEEE transactions on Reliability*, Vol. 34, No. 3, 1985, pp. 216–218.

[50] T.F. Ho, W.C. Chan, and C.G. Chung, "A quantum modification to the Jelinski–Moranda software reliability model," in *Proceedings of the 33rd Midwest Symposium on Circuits and Systems*. IEEE, 1990, pp. 339–342.

[51] P.J. Boland and H. Singh, "A birth-process approach to Moranda's geometric software-reliability model," *IEEE Transactions on Reliability*, Vol. 52, No. 2, 2003, pp. 168–174.

[52] L.I. Al Turk and E.G. Alsolami, "Jelinski–Moranda software reliability growth model: A brief literature and modification," *International Journal of Software Engineering and Applications*, Vol. 7, No. 2, 2016.

[53] N. Langberg and N.D. Singpurwalla, "A unification of some software reliability models," *SIAM Journal on Scientific and Statistical Computing*, Vol. 6, No. 3, 1985, pp. 781–790.

[54] Y. Lian, Y. Tang, and Y. Wang, "Objective Bayesian analysis of JM model in software reliability," *Computational Statistics and Data Analysis*, Vol. 109, 2017, pp. 199–214.

[55] A.G. Aggarwal, P. Kapur, and N. Nijhawan, "A discrete SRGM for multi-release software system with faults of different severity," *International Journal of Operational Research*, Vol. 32, No. 2, 2018, pp. 156–168.

[56] P. Erto, M. Giorgio, and A. Lepore, "The generalized inflection S-shaped software reliability growth model," *IEEE Transactions on Reliability*, Vol. 69, No. 1, 2018, pp. 228–244.

[57] D.H. Lee, I.H. Chang, H. Pham, and K.Y. Song, "A software reliability model considering the syntax error in uncertainty environment, optimal release time, and sensitivity analysis," *Applied Sciences*, Vol. 8, No. 9, 2018, p. 1483.

[58] A.G. Aggarwal, V. Dhaka, N. Nijhawan, and A. Tandon, "Reliability growth analysis for multi-release open source software systems with change point," *System Performance and Management Analytics*, 2019, pp. 125–137.

[59] R. Gupta, M. Jain, and A. Jain, "Software reliability growth model in distributed environment subject to debugging time lag," *Performance Prediction and Analytics of Fuzzy, Reliability and Queuing Models: Theory and Applications*, 2019, pp. 105–118.

[60] M. Asraful Haque and N. Ahmad, "A logistic growth model for software reliability estimation considering uncertain factors," *International Journal of Reliability, Quality and Safety Engineering*, Vol. 28, No. 5, 2021, p. 2150032.

[61] W.D. van Driel, J. Bikker, and M. Tijink, "Prediction of software reliability," *Microelectronics Reliability*, Vol. 119, 2021, p. 114074.

[62] H. Okamura and T. Dohi, "Application of EM algorithm to NHPP-based software reliability assessment with generalized failure count data," *Mathematics*, Vol. 9, No. 9, 2021, p. 985.

[63] K. Sahu and R. Srivastava, "Needs and importance of reliability prediction: An industrial perspective," *Information Sciences Letters*, Vol. 9, No. 1, 2020, pp. 33–37.

[64] K. Kumaresan and P. Ganeshkumar, "Software reliability prediction model with realistic assumption using time series (S) ARIMA model," *Journal of Ambient Intelligence and Humanized Computing*, Vol. 11, 2020, pp. 5561–5568.

[65] K. Sharma, M. Bala et al., "A quantitative testing effort estimate for reliability assessment of multi release open source software systems," *Journal of Computational and Theoretical Nanoscience*, Vol. 16, No. 12, 2019, pp. 5089–5098.

[66] S. Malik, K. Sharma, and M. Bala, "Reliability analysis and modeling of green computing based software systems," *Recent Advances in Computer Science and Communications (Formerly: Recent Patents on Computer Science)*, Vol. 14, No. 4, 2021, pp. 1060–1071.

[67] Sangeeta, Sitender, K. Sharma, and M. Bala, "New failure rate model for iterative software development life cycle process," *Automated Software Engineering*, Vol. 28, No. 2, 2021, p. 9.

[68] X. Liu and N. Xie, "Grey-based approach for estimating software reliability under nonhomogeneous Poisson process," *Journal of Systems Engineering and Electronics*, Vol. 33, No. 2, 2022, pp. 360–369.

[69] B.V. Devi and R.K. Devi, "Software reliability models based on machine learning techniques: A review," in *AIP Conference Proceedings*, Vol. 2463. AIP Publishing, 2022.

[70] V. Verma, S. Anand, and A.G. Aggarwal, "Optimal time for management review during testing process: An approach using S-curve two-dimensional software reliability growth model," *International Journal of Quality and Reliability Management*, 2023.

[71] I. Ramadan, H.M. Harb, H. Mousa, and M. Malhat, "Assessment reliability for open-source software using probabilistic models and marine predators algorithm," *International Journal of Computers and Information*, Vol. 10, No. 1, 2023, pp. 18–35.

[72] V. Pradhan, A. Kumar, and J. Dhar, "Modelling software reliability growth through generalized inflection S-shaped fault reduction factor and optimal release time," *Proceedings of the Institution of Mechanical Engineers, em Part O: Journal of Risk and Reliability*, Vol. 236, No. 1, 2022, pp. 18–36.

[73] V. Pradhan, A. Kumar, and J. Dhar, "Emerging trends and future directions in software reliability growth modeling," *Engineering Reliability and Risk Assessment*, 2023, pp. 131–144.

[74] V. Pradhan, J. Dhar, and A. Kumar, "Testing coverage-based software reliability growth model considering uncertainty of operating environment," *Systems Engineering*, 2023.

[75] J.E. Dennis Jr and R.B. Schnabel, *Numerical methods for unconstrained optimization and nonlinear equations.* SIAM, 1996.

[76] J. Kennedy and R. Eberhart, "Particle swarm optimization," in *Proceedings of ICNN '95 – International Conference on Neural Networks*, Vol. 4. IEEE, 1995, pp. 1942–1948.

[77] J.H. Holland, "Genetic algorithms," *Scientific American*, Vol. 267, No. 1, 1992, pp. 66–73.

[78] A. Sheta and J. Al-Salt, "Parameter estimation of software reliability growth models by particle swarm optimization," *Management*, Vol. 7, 2007, p. 14.

[79] R. Malhotra and A. Negi, "Reliability modeling using particle swarm optimization," *International Journal of System Assurance Engineering and Management*, Vol. 4, 2013, pp. 275–283.

[80] C. Jin and S.W. Jin, “Parameter optimization of software reliability growth model with S-shaped testing-effort function using improved swarm intelligent optimization,” *Applied Soft Computing*, Vol. 40, 2016, pp. 283–291.

[81] S. Mirjalili and S.Z.M. Hashim, “A new hybrid PSOGSA algorithm for function optimization,” in *International Conference on Computer and Information Application*. IEEE, 2010, pp. 374–377.

[82] A. Abraham, R.K. Jatoth, and A. Rajasekhar, “Hybrid differential artificial bee colony algorithm,” *Journal of Computational and Theoretical Nanoscience*, Vol. 9, No. 2, 2012, pp. 249–257.

[83] S. Mirjalili, G.G. Wang, and L.d.S. Coelho, “Binary optimization using hybrid particle swarm optimization and gravitational search algorithm,” *Neural Computing and Applications*, Vol. 25, 2014, pp. 1423–1435.

[84] F. Liu and Z. Zhou, “An improved QPSO algorithm and its application in the high-dimensional complex problems,” *Chemometrics and Intelligent Laboratory Systems*, Vol. 132, 2014, pp. 82–90.

[85] Y. Li, Y. Wang, and B. Li, “A hybrid artificial bee colony assisted differential evolution algorithm for optimal reactive power flow,” *International Journal of Electrical Power and Energy Systems*, Vol. 52, 2013, pp. 25–33.

[86] S.S. Jadon, R. Tiwari, H. Sharma, and J.C. Bansal, “Hybrid artificial bee colony algorithm with differential evolution,” *Applied Soft Computing*, Vol. 58, 2017, pp. 11–24.

[87] K. Sharma, M. Bala et al., “Magnetic navigation based optimizer: A new optimization algorithm for software reliability model parameter estimation,” *Journal of Advanced Research in Dynamical and Control Systems*, 2018, pp. 1957–1968.

[88] A.K. Tripathi, K. Sharma, and M. Bala, “Military dog based optimizer and its application to fake review,” *arXiv preprint arXiv:1909.11890*, 2019.

[89] K. Sharma, M. Bala et al., “An ecological space based hybrid swarm-evolutionary algorithm for software reliability model parameter estimation,” *International Journal of System Assurance Engineering and Management*, Vol. 11, No. 1, 2020, pp. 77–92.

[90] A. Sharma, R. Chaturvedi, S. Kumar, and U.K. Dwivedi, “Multi-level image thresholding based on Kapur and Tsallis entropy using firefly algorithm,” *Journal of Interdisciplinary Mathematics*, Vol. 23, No. 2, 2020, pp. 563–571.

[91] M.S. Khan, F. Jabeen, S. Ghouzali, Z. Rehman, S. Naz et al., “Metaheuristic algorithms in optimizing deep neural network model for software effort estimation,” *IEEE Access*, Vol. 9, 2021, pp. 60 309–60 327.

[92] S. Kassaymeh, S. Abdullah, M. Al-Laham, M. Alweshah, M.A. Al-Betar et al., “Salp swarm optimizer for modeling software reliability prediction problems,” *Neural Processing Letters*, Vol. 53, 2021, pp. 4451–4487.

[93] K. Lakra and A. Chug, “Application of metaheuristic techniques in software quality prediction: A systematic mapping study,” *International Journal of Intelligent Engineering Informatics*, Vol. 9, No. 4, 2021, pp. 355–399.

[94] L. Raamesh, S. Jothi, and S. Radhika, “Enhancing software reliability and fault detection using hybrid brainstorm optimization-based LSTM model,” *IETE Journal of Research*, 2022, pp. 1–15.

[95] P. Dhavakumar and N. Gopalan, “An efficient parameter optimization of software reliability growth model by using chaotic grey wolf optimization algorithm,” *Journal of Ambient Intelligence and Humanized Computing*, Vol. 12, 2021, pp. 3177–3188.

[96] S. Singh, A. Ashok, M. Kumar, and T.K. Rawat, “Adaptive infinite impulse response system identification using teacher learner based optimization algorithm,” *Applied Intelligence*, Vol. 49, 2019, pp. 1785–1802.

[97] Rakhi and G.L. Pahuja, “Solving reliability redundancy allocation problem using grey wolf optimization algorithm,” *Journal of Physics: Conference Series*, Vol. 1706, 2020, p. 012155.

[98] A. Kaushik, D.K. Tayal, and K. Yadav, “The role of neural networks and metaheuristics in agile software development effort estimation,” in *Research anthology on artificial neural network applications*. IGI Global, 2022, pp. 306–328.

[99] N. Yadav and V. Yadav, "Software reliability prediction and optimization using machine learning algorithms: A review," *Journal of Integrated Science and Technology*, Vol. 11, No. 1, 2023, p. 457.

Authors and affiliations

Sangeeta
e-mail: sangeeta@msit.in
ORCID: https://orcid.org/0000-0002-8691-3892
Department of Computer Science and Engineering,
Maharaja Surajmal Institute of Technology, India

Sitender
e-mail: sitender@msit.in
ORCID: https://orcid.org/0000-0003-0341-2927
Department of Information Technology,
Maharaja Surajmal Institute of Technology, India

Rachna Jain
e-mail: rachnajain@bpitindia.com
Department of Information Technology,
Bhagwan Parshuram Institute of Technology, India

Ankita Bansal
e-mail: ankita.bansal06@gmail.com
Netaji Subhas University of Technology, India

e-Informatica Software Engineering Journal
Volume 19, Issue 1, pages: 1–36
DOI: 10.37190/e-Inf250102

BibTeX

ACoRA – A Platform for Automating Code Review Tasks

Mirosław Ochodek*, Miroslaw Staron
*Corresponding author: miroslaw.ochodek@put.poznan.pl

Article info

Dataset link: https://zenodo.org/records/13870908

Keywords:
code reviews
continous integration
BERT
machine learning

Submitted: 18 Jan. 2024
Revised: 5 Aug. 2024
Accepted: 2 Sep. 2024
Available online: 3 Oct. 2024

Abstract

Background: Modern Code Reviews (MCR) are frequently adopted when assuring code and design quality in continuous integration and deployment projects. Although tiresome, they serve a secondary purpose of learning about the software product.
Aim: Our objective is to design and evaluate a support tool to help software developers focus on the most important code fragments to review and provide them with suggestions on what should be reviewed in this code.
Method: We used design science research to develop and evaluate a tool for automating code reviews by providing recommendations for code reviewers. The tool is based on Transformer-based machine learning models for natural language processing, applied to both programming language code (patch content) and the review comments. We evaluate both the ability of the language model to match similar lines and the ability to correctly indicate the nature of the potential problems encoded in a set of categories. We evaluated the tool on two open-source projects and one industry project.
Results: The proposed tool was able to correctly annotate (only true positives) 35%–41% and partially correctly annotate 76%–84% of code fragments to be reviewed with labels corresponding to different aspects of code the reviewer should focus on.
Conclusion: By comparing our study to similar solutions, we conclude that indicating lines to be reviewed and suggesting the nature of the potential problems in the code allows us to achieve higher accuracy than suggesting entire changes in the code considered in other studies. Also, we have found that the differences depend more on the consistency of commenting rather than on the ability of the model to find similar lines.

1. Introduction

Modern Code Reviews (MCR) [1, 2] is a common practice in continuous integration and deployment companies. A modern code review is a practice that evolved from software inspections advocated by Fagan et al. [3] already in 1976, but it adapts to modern tools and the ability to peer review smaller segments of code (commits) pushed by developers to the main branch of the code. MCR is integrated into modern software development pipelines and all leading configuration management platforms enable this way of working. Git and

Gerrit [4,5] are two examples of such tools, where the developers can review other's code before it is integrated with the main branch.

Although MCR is a lightweight process compared to the original inspection process of Fagan, it is still a tiresome process and can result in delays when delivering the product [6]. It is also a process known to miss important quality issues [7]. To address the problem, the majority of current research efforts are targeted towards either eliminating this activity by automated code repairs [8], improving the tools used for code reviews [9], or even predicting which lines of code should be reviewed manually [10,11].

However, one of the main limitations of the automated code repair activities is the low success rate (ca. 30% at best, [8]). The major drawback of the automated suggestion for which code fragments to review is the lack of information on why and what should be reviewed exactly in that code fragment. Furthermore, MCR has secondary goals in addition to quality assurance. It is often perceived as a good way of onboarding developers into new projects and learning within the team [12,13]. Therefore, in this paper, we address the research problem of:

To which degree can we suggest relevant review guidance for a given code fragment based on historical data?

We address this question by designing and constructing an automated code review assistance platform – `ACoRA`. The platform is based on the idea that a programming language can be treated as a natural language from the perspective of machine learning language models [14]. It employs a Transformer-based language model [15] to search for lines of code similar to those under review that were previously commented on. Later, it analyzes the comments to highlight the aspects of code on which the reviewer should focus while reviewing a given code fragment. It performs a multi-class / multi-label classification according to the proposed taxonomy [16] and aggregates the results over the comments for similar lines to guide the reviewers' focus. `ACoRA` can be integrated with MCR tools to learn from the previous reviews to be able to suggest what should be reviewed in a given code fragment.

We use design science research as our methodology as prescribed by [17] and evaluate `ACoRA` on both open-source projects and together with an industrial partner. The results show that we can suggest completely correct recommendations (only true positives) in 35%–41% of the fragments and partially correct in 76%–84% of the fragments. The results are better than the results of similar studies (e.g., suggesting code that repairs a defect or suggesting a review text itself).

The paper is structured as follows. Section 2 summarizes the most relevant related research. Section 3 describes the `ACoRA` platform and Section 4 details our research methodology. Sections 5 and 6 present and discuss the results of evaluating `ACoRA` and Section 7 discusses the threats to validity. Finally, Section 8 presents the conclusions and outlines the further work in this area.

2. Related work

The field of using natural language processing models for programming tasks is developing rapidly. In this section, we provide the current and the most relevant related research in this area.

2.1. Natural Language Processing models applied to code

"On the naturalness of software" by [14] is a seminar work that started a number of research directions in using machine learning for programming tasks. This paper shows that there exist several approaches for using natural language processing machine learning models in software engineering, with a focus on such tasks as program repair or defect finding.

In fact, this field became very popular and a survey by [18] formalized a hypothesis about the naturalness of programming languages: *"The naturalness hypothesis. Software is a form of human communication; software corpora have similar statistical properties to natural language corpora; and these properties can be exploited to build better software engineering tools."* This hypothesis provided a foundation to classify approaches to processing programming languages, but the most important contribution of this work is the classification of the tasks where the hypothesis is used (at least initially, when it was formulated): code-generating models, representational models of code, and pattern mining models. The paper also reviewed each of these application areas and found a significant number of models and applications. For example, for the code-generating models, they found 49 studies, and the number is certainly higher today. Our work, however, is focused on studying the last area – pattern mining of code, although focused on applying these techniques to a specific task – code review support. That area showed a significantly smaller set of studies – 10 studies in 2018.

2.2. Using language models for code repair and generation tasks

One of the first models, which is used in several studies, is the representation of code using information about the program's abstract syntax tree – code2vec, as introduced by [19]. The underlying concept behind code2vec is that a program's code should be represented as a feature vector using information taken from the grammar of the programming language. The feature vector carries information about whether a code fragment is a definition of a function, a single statement, or even a specific type of token. The approach has been demonstrated to work well when translating programming languages or finding meaningful synonyms. However, it has one disadvantage. Namely, it requires the program to compile, which is quite problematic in industrial contexts as well as in the context of MCRs. Since the focus of MCR is, per definition, one commit, then the set-up of the entire code2vec alongside compilers can be problematic, as we found in our previous studies [6].

As opposed to the grammar-based language models, a new trend emerged when BERT models [20] showed significant progress in the area of natural language processing, in particular in translation between programming languages. One of the first BERT-based models in this area is CodeBERT designed by [21]. The model is pre-trained for programming languages like JavaScript or Python along with the English natural language. The model supports translations between a natural language and a programming language for such tasks as writing programs based on natural language commands or summarization of programs in natural languages (documentation generation).

Program generation and translation of natural language to programs are also the base tasks for the TransCoder models presented by Facebook Research. An example of such a task is the deobfuscation task, where the models change identifiers in the program to ones that are more meaningful for software developers, as presented by [22]. The performance of the model is impressive and in the best-case scenarios achieves an accuracy of 67.6%.

AlphaCode as presented by [23], is another prominent example of current state-of-the-art models in program generation. The model is trained to solve problems from programming competitions and uses the same technology. The application of the model demonstrates that it can generate programs based on natural language specifications and also ranks in the top 54.3% compared to ca. 5000 participants. The main difference of this model is that it is trained to solve "artificial" programming tasks, which are not the same as software engineering tasks in the industry. Our work is based on the same principles as AlphaCode (using transformer models) but aligned with the industrial needs and requirements.

However, the accuracy depends on the dataset and the task of the model. An example of a problem, which is significantly more relevant to the industrial context is program repair. There, the newest models, such as Review4Repair [24], can achieve an accuracy of above 30%, which is still a significant improvement from the previous models. Review4Repair solves the task of fixing a given defect based on finding and adapting code fragments from Git. The problem is significantly more difficult than deobfuscation or program generation since the new code fragment has to fit in the existing context. A similar approach was followed by Tufano et al. [25] who studied the possibility of adapting Text-To-Text Transfer Transformer (T5) to either automatically suggest changes in the code under review or to generate such changes based on the reviewer comments. These ways of using language models are the most similar to our approach, with the difference that we do not generate code fragments but guide the focus of the reviewers by indicating lines of code similar to the previously commented ones with hints on potential reasons for comments. Thus, we solve a modified version of this problem. As opposed to Review4Repair, we provide a recommendation for a software developer, who needs to react, rather than providing a solution that needs to be automatically approved (e.g., through testing).

Finally, the latest commercial achievement in this line of research is the GitHub Copilot[1], which is based on OpenAI's GPT-4 model. GitHub Copilot tool can provide both suggestions for new code fragments based on natural language comments of what the code should do, and based on the previous code that has been written (code completion). Our model solves a simpler problem but is trained on a codebase selected by the software developers, which does not pose any legal issues, as they can choose to use the model only on their previous code.

2.3. Modern code reviews

The focus of our work, i.e., code review processes in the continuous integration/deployment context, has industrial roots. The MCR process is effort-intensive and can vary in quality. One of the seminal papers about MCR, and its industrial role, is the study of code reviews at Google [26]. Among other findings, [26] presents identifying code ownership and readability as important factors in the process of code review. They have also found that code reviews should be done on smaller parts of the code to make the process faster, which has implications for the technology used to support the code reviews. The smaller fragments of code require the models to use non-grammar-based approaches. It also poses requirements for being specific when providing recommendations. These implications have also been identified in our previous studies [6] and therefore `ACoRA` generalizes the review suggestions as well as provides recommendations on arbitrary code fragments, e.g., individual lines.

[1]https://copilot.github.com

One of the fundamental challenges when applying machine learning to code reviews is our ability to understand what a good code review [27] is studied the concept of code review and source code change from the perspective of how a good change, or review, is defined in the literature. They verified their studies based on industrial practice. Small size of the change, clear context, and relevant suggestions are some of the identified factors. In `ACoRA`, we follow these findings and provide generalized suggestions about the potential nature of the problem and its context (examples of similar lines of code previously commented on and the comments) to help the developers make the most of the suggestions.

One of the challenges we encounter in our work is the ability to characterize and categorize code review comments. [28] studied a set of code review comments from over 2000 software developers. Their study used pre-defined, fine-grained categories of code reviews and achieved an accuracy of 63.9%. This shows that the accuracy of the `ACoRA`'s `BERT4Comments` model (above 86%) is in line with other models performing automatic classification of pull-request comments, e.g., [29–31].

Continuing in this area, the state-of-the-art models for review recommendation use ensembles and several matches to provide a good recommendation. For example, automating code review tool CORE, presented by [32], uses a corpus of 57 000 code review comments and obtains results at the level of 11% (recall for one suggestion) and 48% (recall for ten suggestions). The results are improvements of two orders of magnitude compared to the previous tools. The low recall for the first suggestion, however, can be linked to the fact that the CORE tool generates natural language suggestions, i.e., a text for the comment. In order to reduce the complexity, `ACoRA` suggests categories of problems rather than generating the text. Our results outperform CORE's recall for a single suggestion.

Instead of providing suggestions for code reviews in general, there are tools that focus on specific types of suggestions. An example of such a tool is the RAID tool, as presented by [33]. The tool focuses on identifying code refactoring opportunities based on analyzing code reviews in MCR. The tool results in significant improvements in the code base, e.g., by reducing the size of the codebase. It also, like `ACoRA`, uses software developers in the loop to make the assessment of the quality of the recommendations.

Tufano et al. [34] studied the strengths and weaknesses of contemporary code-review automation approaches. They identified three types of code-review automation tasks, i.e., code-to-comment, code and comment-to-code, and code-to-code. The first one, code-to-comment, is about generating review comment text for a piece of code under review that would match the comment made by a human expert reviewer. `ACoRA` partially fits into this category. However, instead of generating a comment, we aim to guide the focus of the human reviewers, first to the lines that might need their attention, secondly by suggesting what they should focus on when reviewing the code, and finally by showing examples of comments for similar lines. By doing so, we limit the weaknesses of similar tools that use historical code-review comments to generate review comments, such as CommentFinder [35], since we do not narrow the recommendation to a proposal of a single comment but rather guide the focus of the reviewer.

Finally, the field of MCR has been developing rapidly, and there are several systematic reviews on the topic, e.g., by Badampudi et al. [36, 37], by Davila and Nunes [38], and by Cetin et al. [39]. We can summarize the current state-of-the-art as being focused on either understanding the process of code reviews or providing tool support. Our work contributes to the latter, in particular by creating a support tool. We automate the process and support learning (onboarding new project members, solution discussions) rather than replacing core reviewers.

3. Automated Code Review Assistant platform

Automated Code Review Assistant[2] (ACoRA) is a platform for automated review recommendations based on historical code reviews. In general, it covers two processes: a *configuration process* and a *recommendation process* (see Figure 1).

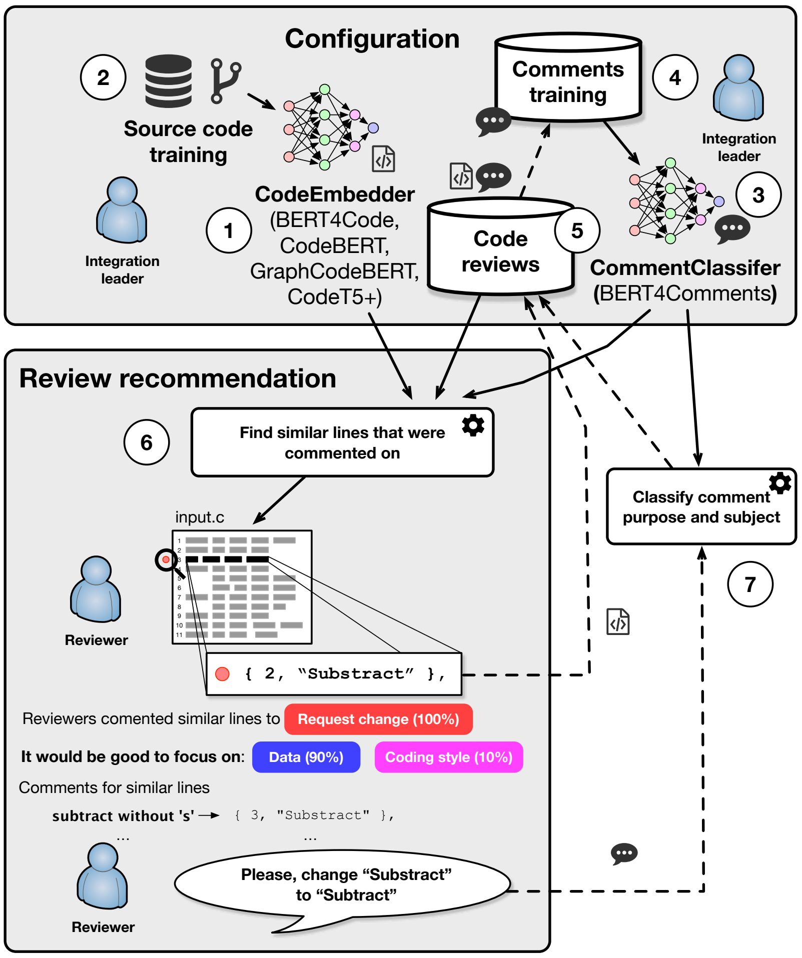

Configuration
2
Source code training
Integration leader
1
CodeEmbedder (BERT4Code, CodeBERT, GraphCodeBERT, CodeT5+)
Comments training
4
Integration leader
3
Code reviews
5
CommentClassifer (BERT4Comments)
Review recommendation
6
Find similar lines that were commented on
input.c
Reviewer
{ 2, "Substract" },
Classify comment purpose and subject
7
Reviewers comented similar lines to
Request change (100%)
It would be good to focus on:
Data (90%)
Coding style (10%)
Comments for similar lines
subtract without 's'
{ 3, "Substract" },
Please, change "Substract" to "Subtract"
Reviewer

Figure 1. Overview of the use of ACoRA. First, the integration leader configures ACoRA. Then, software developers use ACoRA to check their code before it is integrated with the entire product code

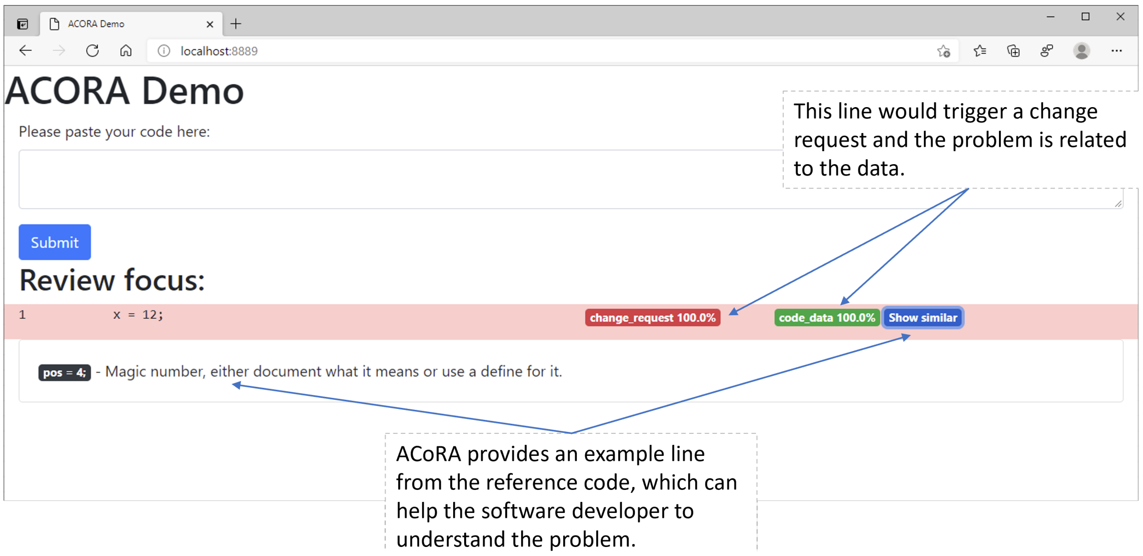
The integration leader, or another continuous integration specialist, configures ACoRA once when the system is being set up. Later, this process can be periodically repeated in order to update the recommendations. Software developers and architects use ACoRA to check the quality of their source code before it is integrated with the entire codebase of the product. Since ACoRA uses code fragments, it can be used as part of the continuous integration toolchain or as an add-on to a development environment.

The configuration process presented in Figure 1 consists of two independent subprocesses. The first one is to pre-train/fine-tune a neural network language model ① (CodeEmbedder) using a codebase ② that seems similar to the target codebase on which one wants to apply ACoRA. Alternatively, one can use publicly available pre-trained models (e.g., CodeBERT,

[2]ACoRA – https://github.com/mochodek/acora

CodeT5+, etc.). The second sub-process is to train/fine-tune a review-comment classifier ③ (`CommentClassifier`) using a dataset of manually labeled comments ④. Finally, one needs to establish a reference database of past code reviews ⑤ that includes commented code chunks and reviewer comments classified with `CommentClassifier`.

A configured `ACoRA` can provide recommendations ⑥ as shown in Figure 2 by searching for similar lines in the reference code reviews database ⑤ to those currently under review. As we can see in Figure 2, `ACoRA` suggested the reviewer to focus on line 3 by providing recommendations about the potential nature of the problem (`code_data`) and an example of a similar line from the reference code database. New comments provided by reviewers can be classified with `CommentClassifier` and stored in the database ⑤.



```
1            x = 12;
```

- Magic number, either document what it means or use a define for it.

This line would trigger a change request and the problem is related to the data.

ACoRA provides an example line from the reference code, which can help the software developer to understand the problem.

Figure 2. Demonstration of `ACoRA` as a stand-alone web service, to be used by software developers. After submitting a code fragment, different lines in this fragment are provided with recommendations on what to fix

3.1. `CodeEmbedder` language models

The core component of `ACoRA` is a language model used to transform programming language text to its vector representation. As the `ACoRA` design follows the pipe and filter architectural style, one can either use a built-in infrastructure to train such a model (`BERT4Code`) or employ one of the publicly available pre-trained models for generating code embedding vectors by adding a new filter component. For instance, in this study, we use three proven pre-trained models, i.e., `CodeBERT`, `GraphCodeBERT`, and `CodeT5+`, as well, as `BERT4Code` models pre-trained from scratch.

The `BERT4Code` model is based on the BERT (Bidirectional Encoder Representations from Transformers) language model [20], which is a deep artificial neural network implementing a multi-layer bidirectional Transformer architecture [15] (however, technically, it uses only the Transformer Encoder stack). The language model is trained and evaluated during the configuration phase of `ACoRA` and used in the recommendation phase. The pipeline for training follows the same principles as the established approaches like TransCoder and CodeGen [40]:

1. Tokenization: where each code fragment is transformed into a set of tokens. We use a modified version of WordPiece tokenizer [41] that split tokens not only based on whitespace characters but also on operators, brackets, etc. [42]. Optionally, `ACoRA` allows to convert the out-of-vocabulary tokens into their symbolic form [43], e.g., a variable identifier `number0` would be replaced by a signature `a0` (a sequence of small letters proceeded with a sequence of digits).
2. Padding: where each code fragment is transformed to a vector of the same size, 128 tokens in our case.
3. Embeddings extraction: a fragment of code is transformed into its embedding representation using four last hidden layers of `BERT4Code` (by following the recommendations for the original `BERT` model [20]).

In the training phase, we pre-train a `BERT4Code` model using the same procedure as for the original `BERT` model [20]. This process is used by other BERT-inspired models [44]. The inputs to the model are pairs of code fragments (in 50% of cases these are consecutive fragments). The model is simultaneously trained on two tasks – Masked Language Model (MLM) and Next Sentence Prediction (NSP). For the former, 15% of tokens in a sequence is being masked and the goal of the network is to guess the original ones. For the NSP task, the network needs to respond to whether the second provided code fragment directly proceeds from the first one. The `BERT4Code` models studied in this paper consist of four layers (384 neurons in each of the hidden layers; 8 attention heads). We use compact `BERT` models [45] since programming languages are more formal (and structured) than natural languages. Also, such networks can be pre-trained on commodity hardware affordable even for small organizations.

In the inference phase, each code fragment is inputted to the `BERT4Code` model, and the embedding vector output is used when calculating the similarity between code fragments.

3.2. `CommentClassifier` language model

`ACoRA` provides a default implementation of `CommentClassifier` called `BERT4Comments`[3]. It is a language mode structurally similar to the `BERT4Code` model, except that it is based on the official pre-trained `BERT` model (12-layer), which is further fine-tuned to classify review comments. The `BERT` model was pre-trained on a large corpus of plain text for the masked word prediction and next sentence prediction tasks. Such a pre-trained `BERT` model can be further fine-tuned to a specific downstream task. The input to `BERT4Comments` is one review comment and the output is a set of categories describing what the comment is about. The process is shown in Figure 3.

The taxonomy of the comments is taken from our previous work [16], and is shown in the top row of Figure 3. The categories of the taxonomy are as follows:

code_design – the comment is about a structural organization of code into modules, functions, classes, or similar, e.g., "code snippet inherited from original dissector. I have refactored the code to have the decompression in a single place now it should be a bit better". It is also about overriding, e.g., "this will not work for IA5. Why not simply override dataCoding before the switch?", and dead/unused code, e.g., "This is duplicated code, put outside the if-else.".

[3]`BERT4Comments` is is also available at the Huggingface repository: https://huggingface.co/mochodek/comment-bert-subject together with a complementary model to annotate comment purpose https://huggingface.co/mochodek/comment-bert-purpose

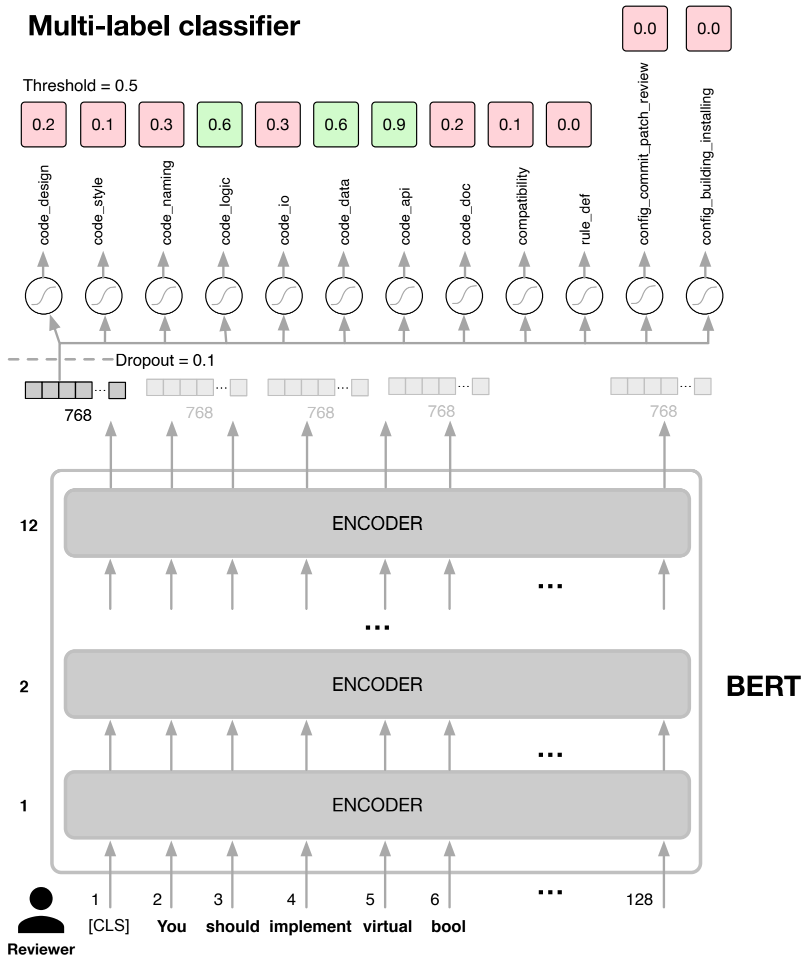


Figure 3. `BERT4Comments` architecture [16]

code__style – the comment is about the layout of the code/readability issues, for example: “add blank line” or “formatting: remove space after 4”.

code__naming – the comment is about issues related to naming code constructs, tables, for example “please use lowercase for field name => ’isakmp.sak.nextpayload’ ” or “Add name of dissector XXX: use custom…”

code__logic – the comment is about algorithms used, operations on data, calling functions, creating objects, and also the order the operations are performed, for example “missing validation of chunk size, potential buffer overflow?” or “should this be initialized with NULL or something?”

code__io – the comment is about input/output, GUI, for example: “What about showing the hub port, i.e., ’address:port’? So the normal endpoints would display as ’address.endpoint’ and split would display as ’address:port’ ” or “Debug output to be removed?”.

code__data – the comment is about data, variables, tables, pieces of information, and strings, for example: “You probably want encoding ENC__BIG__ENDIAN here. You could also use proto__tre__add__item-ret()int() here to avoid fetching the value twice. This is true for other places in the code too” or “Are these ports registered with IANA? If not, I am not sure if they should be used here”.

code__api – the comment is about an existing API or suggestions on how the API should evolve, for example: "This needs to remain the same as before. The dissection must continue therefore the latest offset must be updated after adding to the tree. offset += dissect__dsmcc__un__session__nsap(tvb, offset, pinfo, sub__sub__tree)" or "If they are non-standard and uncommon, I would replace them with: dissector__add__for__decode__as("udp.port", otrxd__handle)".

code__doc – the comment concerns the documentation or comments in the source code, for example: "Which 3GPP document specifies this AVP?" or "Maybe remove this comment now? We do not support older drafts anymore".

compatibility – the comment is related to the operating system compatibility, tools compatibility, versions, or issues that appear only on certain platforms, e.g.: "the Ubuntu failure is to the revert of my previous change (only on__btnImport__clicked() call must be guarded)" or "I can empirically confirm that /proc/self/exe somehow expands to the real path. So this code would probably have no effect on Linux".

rule__def – the comment can be used to elicit a definition of coding/style rules, note it has to explain the broader context, e.g., "'add blank line' is not a definition since we don't know why the blank line should be added here; on contrary, 'use space for indent (like rest of file)' states that spaces should be used for indentation (in general)" or "remove comment when it doesn't help understanding the code".

config__commit__patch__review – the comment is about patches, commits, or review comments, for example: "To be done in the next patch set" or "Right, if you decide to do a formatting patch, it is best to do that in a separate change".

config__building__installing – the comment is about a process of building, installing, and running the product, for example "This is not required. Already done by the install script" or "let's remove this example, installing binary packages across different distros is not supported and we should not recommend users to skip signature checking, etc".

Naturally, this taxonomy can be used manually to understand and classify each comment, but the best support is to use an automated classifier of these comments. The classifier needs to be based on techniques from natural language processing and has to utilize a pre-trained model as the number of comments in a typical repository is not in parity with the diversity of the natural language constructs available.

The proposed taxonomy consists of categories representing high-level problem areas to direct the focus of reviewers. However, it is possible to extend or even replace our taxonomy with other taxonomies like the one proposed by Tufano et al. [34] that defines low-level code issues that are raised during MCRs.

In our previous study [16], we trained and validated `BERT4Comments` models on the dataset of 2,672 MCR comments from three open-source projects, i.e., Wireshark, The Mono Framework, and Open Network Automation Platform (ONAP). The accuracy of the models ranged from 0.84 to 0.99 (mean = 0.94). The mean Matthews Correlation Coefficient (MCC) value was equal to 0.60, which can be considered a moderate to strong correlation [46]. Finally, the average Area under the ROC Curve (AUC) was equal to 0.76, which is an acceptable value for a classifier according to [47].

4. Research methodology

In this work, we use design science research (DSR) methodology as described by Wieringa [17]. The first step recommended by DSR is problem formulation, which usually entails the need to organize artifact development and treatment design into more distinct parts [48].

The evaluation is structured into the initial evaluation, where we study open-source projects in-depth, and the external evaluation, where we apply `ACoRA` on an industrial project with an industrial partner – Bosch Gmbh.

The reproduction package for the study containing datasets and scripts used to perform the analyses (with the exclusion of confidential data) is publicly available[4].

4.1. Problem formulation

In the context of our study, the question of *"To what degree can we suggest relevant review guidance for a given code fragment based on historical data?"* has two parts, two sub-questions:

RQ1: To what degree does the language embedding model find similar code fragments?

RQ2: How relevant are the review comment suggestions with respect to the nature of the problem identified by reviewers?

RQ1 addresses our need to understand how well language embedding models (`CodeEmbedders`) find similar code fragments. We need to know whether the lines that are matched as similar are relevant – whether two given lines can be judged as similar or not. The similarity is a concept that depends on the textual content of the line, its context, and the semantics of a line, and there is no good measure for that [49]. Therefore, in our work we approximate similarity by changes in code fragments – we can change a code fragment so that it is textually different but has a resemblance in terms of its purpose and context. We seed changes to code fragments and assess the fragments returned by the language model as similar. The changes are designed to exemplify types of problems that the reviewers comment on, i.e., the taxonomy presented in Subsection 3.2 and studied in [16].

RQ2 addresses the problem of how well the review suggestions correspond to the suggestions that a code reviewer would provide. Since the review comments are specific to the code fragments commented on, we need to generalize them, and therefore we use the comment taxonomy and `BERT4Comments` as the means to provide such suggestions. As learning-by-example has been identified as an important aspect of MCR, we provide a similar line and its comment as an example. **RQ2** also addresses the challenge related to the reproducibility of suggestions and their generalizability. Since the review comments are specific to the code fragment, they cannot be directly provided as suggestions for other code fragments (even similar ones). However, they can capture problems that can be generalized and these generalized suggestions can be used as guidance for the software developers reviewing the new code fragment.

4.2. Treatment design

To address the research questions and ensure that `ACoRA` supports both the automation of code reviews and the possibility of training new project members, it is integrated into a continuous integration pipeline, as shown in Figure 4.

[4]Reproduction package – https://zenodo.org/records/13870908.

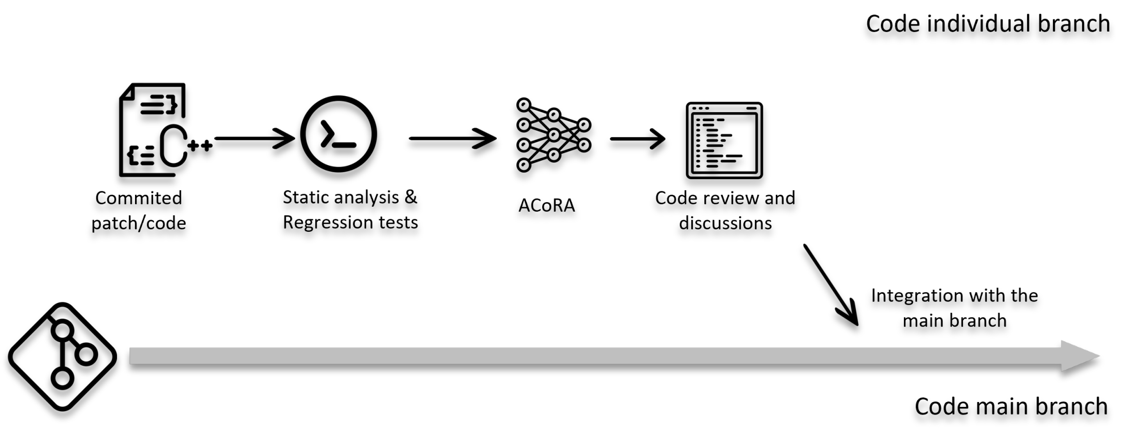

Code individual branch
Commited patch/code
Static analysis & Regression tests
ACoRA
Code review and discussions
Integration with the main branch
Code main branch

Figure 4. `ACoRA` integrated into a CI pipeline

The integration is done using docker containers, i.e., we designed `ACoRA` as a microservice that can be plugged into a continuous integration pipeline. It has several components designed according to the pipes and filters architectural style that allow for flexibility in implementing and replacing the components. The tool can be configured to provide feedback offline, as shown in Figure 2. The detailed design is described in Section 3.

4.3. Initial treatment evaluation and improvements

To evaluate the code review platform, we chose to use two separate open-source projects, which had an open Gerrit code review tool instance available for public access. For the initial treatment evaluation, we chose two open-source projects:

– Wireshark (https://www.wireshark.org) – an open-source network protocol analyzer, developed by professionals from leading telecommunication companies.
– Cloudera (https://www.cloudera.com) – an open-source cloud hosting solution, developed by a professional community.

Both projects are professionally developed and we chose them because of the size and quality of the code reviews in Gerrit. These two products are also rather specific, so the variety of use cases is not considered as a factor in assessing code reviews. In particular, when discussing the selection with our industrial partner, we wanted tools that were either in C/C++ or Java, developed by professional developers, and had a specific purpose. We considered using other tools like Android to study, but the diversity of the product (mix of C, C++, and Java), as well as the number of use cases (it is an operating system), made it not fit our requirements.

First, we prepared for the evaluation by performing the following steps:

Step 1: Clone a repository at a selected revision (#base_rev).

Step 2: Obtain a `CodeEmbedder` model either by pre-training `BERT4Code` on the code downloaded in Step 1 or by choosing one of the pre-trained language models, i.e., `CodeBERT` [21], `GraphCodeBERT` [50], or `CodeT5+` [51].

Step 3: Fetch the MCR review comments for the revisions following the #base_rev and the specific code fragments that were reviewed (*all-code dataset*).

Step 4: Use the `CommentClassifier` (i.e., `BERT4Comments` trained on the manually labeled dataset of MCR comments [16]) to classify the MCR comments in the *all-code dataset*.

Then, we performed the following steps to address **RQ1**:

Step 5: Select a sample of the code fragments from the *all-code dataset* (up to 11 fragments per each of the taxonomy categories).

Step 6: Introduce changes to the code fragments based on the taxonomy (*modified dataset*).

Step 7: Assess the similarity of the matched code fragments.

We decided to select and study four Transformer-based `CodeEmbedders` (Step 2). First, we pre-trained `BERT4Code` according to the process described in Subsection 3.1. This represents a scenario where a `CodeEmbedder` is pre-trained on a small but very representative codebase (an intra-organization usage). Then, we employed three language models pre-trained on large code corpora that are publicly available, i.e., `CodeBERT`, `GraphCodeBERT`, and `CodeT5+`. The former model is a BERT-based model pre-trained on combined inputs of natural language texts and code. `GraphCodeBERT` augments the inputs with the information about data flow. Finally, `CodeT5+` is an example of a large Transformer-based model. Although our goal was not to determine what is the best possible `CodeEmbedder`, the variety of selected models allowed us to gain early insights regarding the differences in how `ACoRA` performs depending on the embedding model it uses.

In Step 6, we modified lines to investigate how sensitive to such changes `CodeEmbedders`' embeddings are when they are used to find similar lines. We selected up to 10 lines for each of the comment categories from our taxonomy and introduced the following types of changes related to the category of comments they belong to:

– **config__commit__patch__review** – since most of the comments belonging to that category refer to lines of code that are commented, we decided to add/modify/replace particular parts of the text in a line, e.g., a person's name, commit identifier referred to in a line, a part of variable identifier, etc.,

Examples:

Original: ❝`* Copyright (C) 2010-2012 The Async HBase Authors. All rights reserved.`❞
Modified: ❝`* Copyright (C) 2010-2012 The another company Authors. All rights reserved.`❞
Original: ❝ `* scanners through the @link KuduScanToken API. Or use MapReduce.`❞
Modified: ❝ `* scanners through the @link AnotherPatch API. Or use MapReduce.`❞
Original: ❝`Change-Id: I1d6dfc4314091eb6f3eef418c5a17ed37f7a1200`❞
Modified: ❝`Change-Id: I2d6dfc4314091eb3d3eef418c5a17ed37f7a1200`❞

– **code__logic** – we changed the logic of the code by modifying the operators being used,

Examples:

Original: ❝ `return percent/4;`❞
Modified: ❝ `return percent*4;`❞
Original: ❝ else if (containers_total_ != 0) ❞
Modified: ❝ else if (containers_total_ == 0) ❞
Original: ❝ `offset += 4_`❞
Modified: ❝ `offset -= 4_`❞

– **code_data** – we changed either the values or the types of variables,

Examples:

Original: “ `int64_t time_elapsed = 0;`”
Modified: “ `int32_t time_elapsed = 0;`”
Original: “ `percent += 100;`”
Modified: “ `percent += 1000;`”

– **code_style** – we added/removed indentation/trailing whitespaces,

Examples:

Original: “`        int num_micro_batches = `”
Modified: “`int num_micro_batches = `”
Original: “`    if (packet_num > 0) {`”
Modified: “`        if (packet_num > 0) {`”

– **code_doc** – we commented/uncommented lines using inline or block comments,

Examples:

Original: “ `return 11_`”
Modified: “ `// return 11_`”
Original: “`/* packet_list.cpp`”
Modified: “`packet_list.cpp`”

– **code_io** – we modified sub-words in identifiers to their synonyms or changed abbreviations to full words,

Examples:

Original: “ `LOG(ERROR) << s;`”
Modified: “ `LOG(PROBLEM) << s;`”
Original: “ `ctx_menu_.addSeparator()_`”
Modified: “ `context_menu_.addSeparator()_`”

– **code_api** – we added/removed some parameters from function headers/calls,

Examples:

Original: “ `ascendlex_destroy(&scanner)_`”
Modified: “ `ascendlex_destroy(&scanner, force)_`”
Original: “ `DCHECK_EQ(out_length, cleartext[i].size());`”
Modified: “ `DCHECK_EQ(out_length, cleartext[i].size(out_length));`”

– **code_naming** – we changed the character case for some identifiers,

Examples:

Original: “`#define USBLL_POISON 0xFE`”
Modified: “`#define usbll_poison 0xFE`”
Original: “ `gint bytes_consumed_`”
Modified: “ `gint Bytes_Consumed_`”

– **code__design** – we modified/removed the keywords or names of identifiers,

Examples:
Original: ❝ `if ( skipped > 0 )`❞
Modified: ❝ `while( skipped > 0 )`❞
Original: ❝ `LZ4F_freeDecompressionContext(lz4_ctxt)_`❞
Modified: ❝ `LZ4F_freeCompressionContext(lz4_ctxt)_`❞

– **compatibility** – we added a suffix `_v2` to some identifiers,

Examples:
Original: ❝ `struct hf_tree tree = 0_`❞
Modified: ❝ `struct hf_tree tree_v2 = 0_`❞
Original: ❝`find_path(PCAP_INCLUDE_DIR`❞
Modified: ❝`find_path_v2(PCAP_INCLUDE_DIR`❞

To assess the matched code fragments in Step 7, we used the ranking of the recommendations. The fragment that is returned as the closest one (calculated as the Minkowski distance, used by `ACoRA`, between the embedding vectors) is ranked 1, the second closest is ranked 2, and so on. For practical reasons, we only report up to rank 10, and for higher ranks, we only report that the rank is higher than 10. In this case, we assess how well `CodeEmbedder` can match the modified line to the original line.

To address **RQ2**, we focused on evaluating the quality of `ACoRA`'s recommendations by performing the following steps:

Step 8: Divide the *all-code dataset* into *reference database* and *validation dataset* based on a selected revision (#split__rev) – remove duplicates so both datasets consist only of unique code fragments.

Step 9: Use `ACoRA` to provide recommendations for the code fragments in the *validation dataset* based on the *reference database*.

Step 10: Assess the relevance of the recommendations.

To evaluate the relevance of the suggestions, we start by dividing the code fragments in *all-code dataset* into the *reference database* and *validation dataset* to mimic how `ACoRA` would be used. We eliminate the duplicated code fragments so we do not bias the results by over-representing a given code fragment. We then use the *reference database* as a basis for providing recommendations for the code fragments in *validation dataset*. We apply the following filtering criteria to mitigate the impact of the following issues related to data quality:

– MCR comments attached to empty lines or code-block opening/closing – we remove lines that contain less than three characters – unfortunately, we observed that sometimes comments concerning fragments of code are attached by reviewers to lines such as closing or opening of code blocks or to empty lines proceeding or preceding a given code fragment. As a result, there is no logical association between the comment provided by the reviewer and the line of code in the data.
– Acknowledgment comments – we remove comments being classified by the `BERT4Comments` model as `acknowledgment` (e.g., "Thank you", "Done") since these are irrelevant from the perspective of providing suggestions to reviewers.
– Reviewers' discussion not resulting in change requests – we remove the lines for which there were no comments classified by `BERT4Comments` as `change_request`. Since not all comments provided by reviewers have to be relevant, we assume that the discussions

between reviewers that do not result in requesting a change to be made in code are not useful as the basis for automatic recommendations.

In order to evaluate the correctness of the suggestions, we calculate how many of the suggested categories overlap with the original categories, compared to the number of categories in the original line, according to Formula 1. In the formula, O is the overlap ratio, N is a set of categories of comments in the new line and R is the set of categories of comments for the reference line.

$$O = \frac{|N \cap R|}{|N|} \tag{1}$$

Formula 1 is designed so that the correct suggestion means that the overlap ratio is 1.0 while a suggestion that misses all categories is 0.0. A partial suggestion is between these values, with the better suggestions being closer to 1.0.

Finally, the quality of recommendations provided by automatic tools such as `ACoRA` depends on how similar the lines and comments in the *reference database* are with respect to the new code being targeted for review. Therefore, we perform the analyses for three scenarios. In the first scenario, we force `ACoRA` to provide recommendations for every line in *validation dataset*, no matter how similar or different are the reference and validation lines. In the following two scenarios, we use a maximum distance threshold between the line-embedding vectors generated by `CodeEmbedder`s to control whether a recommendation should be made or not. We calculate these thresholds based on the distribution of distances between the most similar lines in the *reference database* (10th and 50th percentile). We assume that lowering the distance threshold should result in increasing the relevance of comments with the cost of decreasing the number of cases for which the recommendation could be provided.

4.3.1. Wireshark and Cloudera

In the Wireshark and Cloudera projects, we focused on the subset of the projects written in C as our goal was to use the same base language model for all three evaluations. In the industrial validation, we limited the search to a smaller number as our visit time was limited. Project-specific parameters are presented in Table 1.

These three projects are selected since we validate slightly different aspects in each context, as prescribed by both design science research [17] and action research [48]. In Wireshark, we focused on the curation of the dataset for training. In Cloudera, we focused on the scalability of the approach and sensitivity to noise in the dataset, and in industry, we focused on the usability of the suggestions from the perspective of a software developer.

The `BERT4Comments` used in the study was trained on a subset of manually curated and labeled Wireshark codebase. The curation was done by:

– Removing the pairs `<line, comment>` which are wrong, e.g., when the review comment does not comment on anything specific to that line.
– Rewriting the comment to make the text more general and less specific to a particular line, e.g., by changing a comment about datatype `int` to a comment about a datatype in general.

Table 1. Project specific parameters used in our research design

Parameter	Wireshark	Cloudera	Industry
Revision (#base_rev) (Step 1)	5e34492a7e	10e3cec127	N/A
Pre-train `BERT4Code` (Step 2)	20 epochs, batch size 32, sequence length 128	the same	the same
Review comments (Step 3)	2015-02-23	2022-01-26	2021-11-24
Reference database (Step 8)			
– comments	40,430	74,000	N/A
– unique code fragments	9,930	29,599	15,000
Validation dataset (Step 8)			
– comments	46,850	174,630	N/A
– unique code fragments	12,789	51,034	10 commits
Filtered suggestions (Step 9)	1,582	4,738	10 commits

Note 1: The Wireshark community has migrated from Gerrit to GitLab and its old Gerrit instance is no longer available therefore we only use data up until 2019-12-31. Note 2: We are not allowed to publish some information regarding the industrial source code (N/A)

We used the same approach in our previous work and the accuracy and MCC for `BERT4Comments` were 0.86–0.98 (Accuracy) and 0.32–0.62 (MCC) [16]. Therefore, the evaluation of `BERT4Comments` is outside of the scope of this article.

4.4. Evaluation at the industrial partner

To understand the limitations of `ACoRA`, we design an evaluation of the tool at our industrial partner. The evaluation was done in the following way.

First, we pre-trained `ACoRA`'s `BERT4Code` on the source code from an open-source project suggested by the company employees as being similar to their code (19 218 513 lines of code). Next, we fetched code and comments from a Gerrit instance of a selected project at the company. For practical reasons, we limited the dataset to 500 patches, which included approximately 18 000 lines of code – 15 000 with comments and 3000 without comments. The commented lines were used as the *reference database* in this study.

Second, we tested `ACoRA` on 10 selected commits from another project as the *validation dataset*. We asked the company representatives to extract 10 commits containing code fragments that were commented on by a code reviewer, based on the method used to evaluate software measures [52]. We used `ACoRA` to provide suggestions for the code based on the comments in the *reference database*. The company representatives were asked to judge the correctness and actionability of each recommendation. We adjusted `ACoRA` to be over-sensitive, i.e., provide more suggestions for reviews, in order to challenge the industrial partners to check whether more lines should be commented on in the evaluated commits.

5. Results

5.1. Wireshark

5.1.1. RQ1: Finding similar lines – Wireshark

The recommendations provided by `ACoRA` are based on finding similarities between the lines under review and the lines previously commented on by reviewers. `ACoRA` performs that task by measuring the distance between line embeddings generated by CodeEmbedder. Unfortunately, the neural-network-based language models work as black boxes and we cannot tell exactly what relationships between tokens and code lines they capture. Therefore, we decided to study this phenomenon by investigating how sensitive the selected CodeEmbedders' code-line representations are to certain types of modifications introduced to lines when used to search for line similarities. We modify a line of code, ask ACoRA to search for similar lines to the modified one, and calculate the ranking position of the original line in the lines recommended by ACoRA. If the ranking position of the original line is greater than one, it means that the change introduced to the line caused it to be more "similar" to some other lines in the dataset than to the original line from which it was derived. The dataset included 35 000 Wireshark lines (including the 100 original lines that were modified), therefore, it was rich in examples of lines that could be identified as more similar to the modified lines than the original ones.

Figure 5 shows the distributions of ranking positions of original lines in the recommendations provided for their modified counterparts in the dataset when CodeT5+ was used as a CodeEmbedder. Although none of the CodeEmbedders appeared unanimously superior, ACoRA using CodeT5+ seems to provide the best overall results (the results for the remaining CodeEmbedders are presented in Table 2). For all CodeEmbedders, the changes made in the commented lines belonging to config_commit_patch_review and code_style categories appeared as difficult. Also, BERT4Code, CodeBERT, and GraphCodeBERT all had problems with detecting similarities for the code_doc category, which was not the case for CodeT5+. Finally, CodeT5+ performed slightly worse than BERT4Code for the code_data category.

We analyzed each of the cases where the original line was not provided as the first recommendation to study and hypothesize about what differences in code make the studied CodeEmbedders' embeddings recognize lines as similar or not.

For the changes made to the lines belonging tocode_io, the top suggestions made by ACoRA were the original lines. Therefore, the changes made to these lines were not significant enough (with regards to how the lines are represented by CodeEmbedders' embeddings) to make any other lines more similar to them than their original lines.

config_commit_patch_review – ACoRA using CodeT5+ found five perfect matches (the original line appears as the first recommendation). These lines were modified by changing the e-mail addresses of reviewers, changing the commit hash, or a website address. CodeBERT and GraphCodeBERT failed to match the lines with modified e-mail addresses, while BERT4Code matched only the lines with modified change hashes. Still, the recommendations for the remaining lines in this category seemed valid, despite the fact that the original lines were not provided as top recommendations. For instance, when we modified one of the lines having the structure of Petri-Dish: Name Surname <e-mail address> by changing the person's name and e-mail address, the original line appeared as the sixth recommendation, however, all ACoRA suggestions seemed equally valid since they all had exactly the same structure but contained names and addresses of other people. We made similar observations for other lines, e.g., for the line "Reviewed-on: https://code.wireshark.org/review/33705" that we modified by changing the review identifier to 52705. The original line appeared as the recommendation number 42, however, all proceeding lines in the ranking had exactly the same structure but different review identifiers (e.g., "Reviewed-on: https://code.wireshark.org/review/22515").

code_logic – the changes we made for commented lines belonging to this category were mostly simple operator overloading (e.g., "$<$" $\rightarrow$ "$>$"). ACoRA using CodeT5+ was able to

Table 2. Similarity of modified lines to their original counterparts for Wireshark (all `CodeEmbedders`)

`CodeEmbedder`/taxonomy category	Original line similarity rank										
config__commit__patch__review	1	2	3	4	5	6	7	8	9	10	11+
`BERT4Code`	2	1	2	–	–	1	–	–	–	–	4
`CodeBERT`	5	–	–	–	1	–	–	1	–	–	3
`GraphCodeBERT`	4	–	–	–	–	1	–	–	–	–	5
`CodeT5+`	5	1	1	–	–	–	–	–	1	–	2
code__logic	1	2	3	4	5	6	7	8	9	10	11+
`BERT4Code`	8	1	–	1	–	–	–	–	–	–	–
`CodeBERT`	9	–	–	1	–	–	–	–	–	–	–
`GraphCodeBERT`	10	–	–	–	–	–	–	–	–	–	–
`CodeT5+`	9	1	–	–	–	–	–	–	–	–	–
code__data	1	2	3	4	5	6	7	8	9	10	11+
`BERT4Code`	10	–	–	–	–	–	–	–	–	–	–
`CodeBERT`	9	–	–	–	–	–	–	–	–	–	1
`GraphCodeBERT`	9	–	–	–	–	–	–	–	–	–	1
`CodeT5+`	9	–	1	–	–	–	–	–	–	–	–
code__style	1	2	3	4	5	6	7	8	9	10	11+
`BERT4Code`	1	–	–	–	–	–	–	–	–	–	9
`CodeBERT`	4	–	–	–	1	–	–	–	–	–	5
`GraphCodeBERT`	8	–	–	–	1	–	–	–	–	–	1
`CodeT5+`	9	–	–	–	–	–	–	–	–	–	1
code__doc	1	2	3	4	5	6	7	8	9	10	11+
`BERT4Code`	1	–	2	–	–	–	–	–	–	–	8
`CodeBERT`	6	–	–	–	1	–	–	–	–	–	3
`GraphCodeBERT`	8	–	–	–	–	–	–	–	–	–	2
`CodeT5+`	10	–	–	–	–	–	–	–	–	–	–
code__io	1	2	3	4	5	6	7	8	9	10	11+
`BERT4Code`	10	–	–	–	–	–	–	–	–	–	–
`CodeBERT`	10	–	–	–	–	–	–	–	–	–	–
`GraphCodeBERT`	10	–	–	–	–	–	–	–	–	–	–
`CodeT5+`	10	–	–	–	–	–	–	–	–	–	–
code__api	1	2	3	4	5	6	7	8	9	10	11+
`BERT4Code`	10	–	–	–	–	–	–	–	–	–	–
`CodeBERT`	8	–	–	–	–	–	–	–	–	1	1
`GraphCodeBERT`	10	–	–	–	–	–	–	–	–	–	–
`CodeT5+`	10	–	–	–	–	–	–	–	–	–	–
code__naming	1	2	3	4	5	6	7	8	9	10	11+
`BERT4Code`	4	3	–	1	–	–	–	–	–	–	2
`CodeBERT`	8	–	1	–	–	–	–	–	–	–	1
`GraphCodeBERT`	9	–	–	–	–	–	–	–	–	–	1
`CodeT5+`	9	1	–	–	–	–	–	–	–	–	–
code__design	1	2	3	4	5	6	7	8	9	10	11+
`BERT4Code`	6	–	–	–	–	–	–	1	–	–	3
`CodeBERT`	9	–	–	–	–	–	–	–	–	–	1
`GraphCodeBERT`	10	–	–	–	–	–	–	–	–	–	–
`CodeT5+`	10	–	–	–	–	–	–	–	–	–	–
compatibility	1	2	3	4	5	6	7	8	9	10	11+
`BERT4Code`	10	–	–	–	–	–	–	–	–	–	–
`CodeBERT`	9	–	–	–	–	–	–	–	–	–	1
`GraphCodeBERT`	9	–	–	–	–	–	–	–	–	–	1
`CodeT5+`	10	–	–	–	–	–	–	–	–	–	–

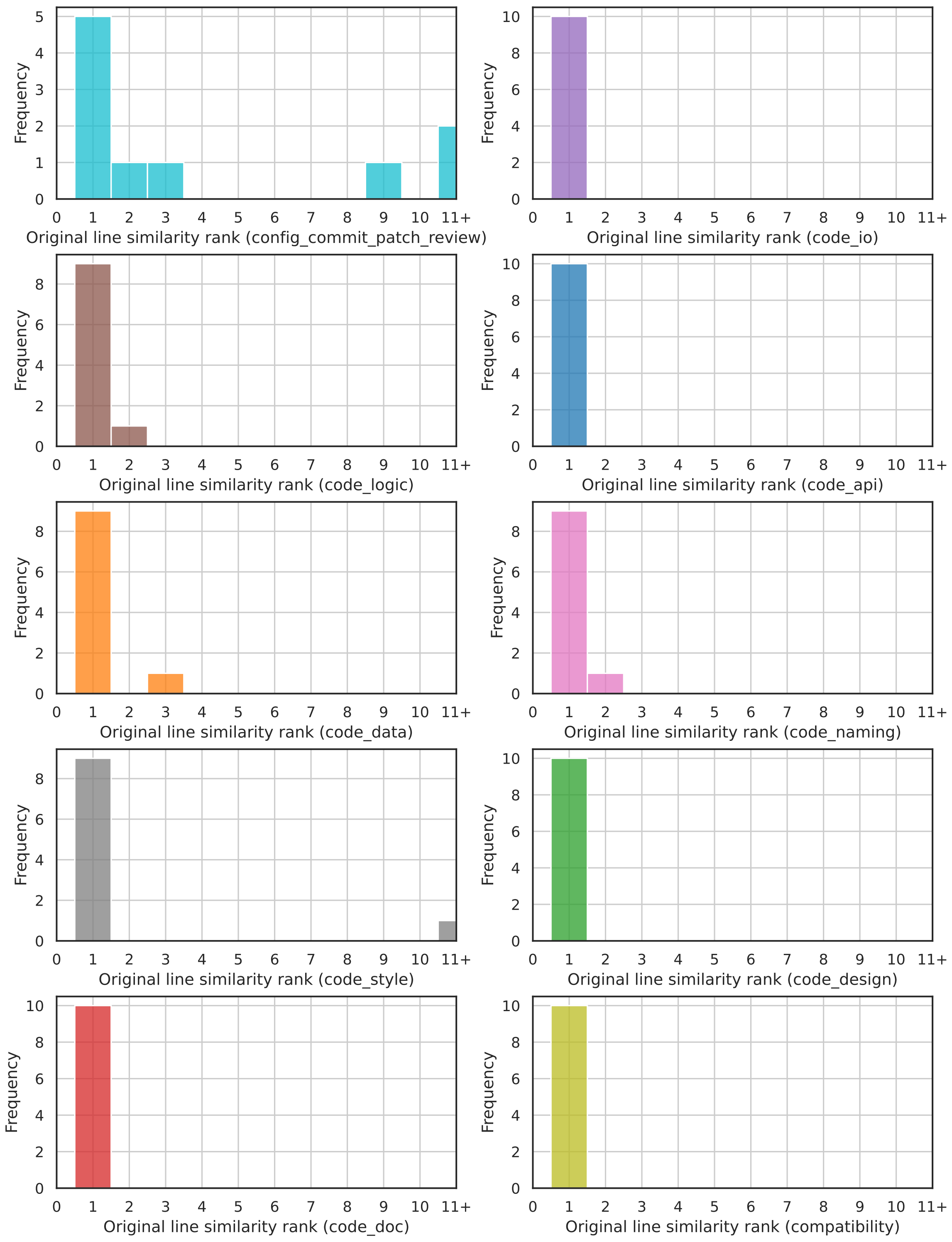


Figure 5. Similarity of modified lines to their original counterparts for Wireshark (`CodeT5+`)

indicate the modified line in the first place in nine out of ten cases. However, in the only case that the modified line was ranked second, the indicated similar line seemed more similar to the original line than its modified version. The original line was “ `*buf = '\0'_ /* NULL terminate */`” and was modified by removing the pointer operator `*`. The most similar line found by `ACoRA` was “ `*buf = '\0'; /* NULL terminate */`” which differs by a single character and has the pointer operator. Interestingly, `ACoRA` using `GraphCodeBERT` indicated all modified lines as the most similar to the original ones. However, taking into account the above, this might not necessarily be the best result. Finally, `BERT4Code` performed slightly worse and suggested eight out of ten modified lines. The two exceptions were the lines: “`if (tree) {`” modified to “`if ( ! tree) {`” and “`DISSECTOR_ASSERT(pos+chunk_size <= length)_`” changed to “`DISSECTOR_ASSERT(pos+chunk_size > length)_`”. In the case of the former, the line that was considered more similar by `ACoRA` was the line “`if (len > length) {`” – a conditional expression that has the same number of indenting whitespace characters but uses a different operator (“>” instead of negation). The original line has no operator at all, which might be the cause of recognizing this line as more similar. For the latter line, the recommendations were less clear to us. The top recommended line was “`g_free(temp_name)`”, which is also a function call, but in contrast to the modified line, there are no operators used to calculate the values of function parameters being passed.

`code_data` – only `ACoRA` using `BERT4Code` provided all ten modified lines as top suggestions, however, the remaining `CodeEmbedder`s were also very accurate with nine out of ten correct suggestions.

`code_style` – we observed mixed results for this category. `ACoRA` using `CodeT5+` or `GraphCodeBERT` correctly indicated nearly all modified lines (nine and eight, respectively), while for `BERT4Code` and `CodeBERT` in nine and five cases the modified line was ranked beyond top ten suggestions. After analyzing the worst cases, we could observe that tab (⎵) was often “recognized” as a visibly different token than space (␣). For instance, when we removed indentation spaces from the line “`␣␣␣␣if (api_version >= 4 ) {`”, the most similar line suggested by `ACoRA` was “`⎵if (tokenlen >= 1) {`”. Similarly, the top match for the line “`⎵⎵if(is_encrypted && !docsis_didssect_encrypted_frames)`” modified by changing two indentation tabs (⎵⎵) to four spaces (␣␣␣␣) was “`␣␣␣␣ if (is_from_server && session->is_session_resumed)`”. Therefore, the presence of indentation spaces influenced, to a large degree, the embedding vector for these lines. These observations were consistent among the other studied examples. Therefore, it seems that using `BERT4Code` or `CodeBERT` embeddings for finding similar lines is sensitive to the number and type of indentation whitespaces. We hypothesize that it might be a consequence of how `BERT` models are trained. One of the tasks used to pre-train `BERT` is next sentence prediction (i.e., next line of code prediction in the case of `BERT4Code`). In programming languages, such as C/C++, whitespaces are used by programmers to indent code blocks to improve code readability. Therefore, it seems that the number of trailing whitespaces could be an important code-feature that a `BERT` model uses when it is pre-trained on source lines of code. Also, using spaces for indentation (typically 2 or 4) is preferred over tabs within the Wireshark community, therefore, using tabs for that purpose could be considered as an anomaly. Finally, this property of `BERT4Code` does not have to be necessarily perceived as its weakness, since it could help identify issues related to wrong indentation in the code.

`code_doc` – again, we observed that `ACoRA` using `CodeT5+` or `GraphCodeBERT` was visibly better in finding modified lines than the variants using `BERT4Code` or `CodeBERT`. `CodeT5+`-based `ACoRA` correctly indicated all of the modified lines and provided them as top-ranked suggestions. At the same time, it seems that the toggle between a commented/non-commented line seemed to visibly influence the embeddings generated by `BERT4Code` Only for one of the modified lines, its original counterpart was returned as the top recommendation. However, since the line was a long (113 characters), inline comment, by removing the "//" we made it an invalid C/C++ line ("`// Adapted from sample code in https://raw...`") and difficult to match by any other line in the dataset. For the remaining lines, when a non-commented line was turned into a commented line, the suggestions provided by `ACoRA` became mostly commented lines. For instance, for the line "`return 11_`" modified to "`// return 11_`", the top recommendation was "`// Hex dump -x`" (both lines have four proceeding spaces and one space between "//" and the following token). The original line was ranked as the 30,903rd suggestion. Similarly, when we converted a commented line into a non-commented line, the suggested lines were also non-commented lines. For instance, for the line "`/* packet_list.cpp`" changed to "`packet_list.cpp`", the top suggestion was "`cfutils.h`" (the following suggestions were also mainly names of files). Interestingly, only at position 22 of the ranking was the first valid C/C++ code line consisting of an include statement ("`#include "packet-ssl-utils.h"`") that also contained the name of a file (with the word "packet"). The original line was returned as the 5464th recommendation.

`code_api` – only the variant of `ACoRA` using `CodeBERT` did not top-ranked all the modified lines. It struggled with finding modified lines for "`col_append_fstr(pinfo->cinfo, COL_INFO, "[zstd decompression failed]")_`" and "`decompress_lz4(tvbuff_t *tvb, packet_info *pinfo _U_, int offset, int length, tvbuff_t **decompressed_tvb, int *decompressed_offset)`". In the case of the second line the modified line appeared at the 75th position of the ranking.

`code_naming` – `ACoRA` using `CodeT5+` suggested nine out of ten modified lines in the first place while the remaining one was the 2nd recommendation. For other `CodeEmbedders`, in most cases, the original lines were either provided as a top suggestion or within the top 2 or 4 lines. All the perfect matches were lines that did not include method/function call, e.g., "`#define USBLL_-POISON 0xFE`." For method calls, we observed that changing the casing was against the convention used by the Wireshark community. The suggested lines had similar structures and often included calls to function with similar identifiers. For instance, for the line "`prefs_register_-enum_preference(smpp_module, "decode_sms_over_smpp",`" that we modified to `Prefs_-Register_Enum_Preference(smpp_module, "decode_sms_over_smpp",`", the top suggestion was "`prefs_register_uat_preference(someip_module, "_udf_someip_parameter_-list", "SOME/IP Parameter List",`".

`code_design` – `ACoRA` using `CodeT5+` or `GraphCodeBERT` correctly suggested all ten modified lines as the most similar to the original ones. The variant using `CodeBERT` missed one of the modified lines and ranked it at 173rd place. For `BERT4Code`, six out of ten lines a change made to an identifier or adding/removing a keyword (adding `else` to an `if` statement; removing a `static` keyword) resulted in suggesting the original line at the top of the ranking. The lowest ranking position (219) was observed for an `if` statement "`if ( skipped > 0 )`" converted to a `while` statement "`while ( skipped > 0 )`" (the same as the one missed by `CodeBERT`). However, the top suggestion seemed more adequate than the original line since it was also a `while` statement "`while (cert_list_length > 0)`." Therefore, once again, it is not clear whether, in this case, the most similar line is the modified line or the one suggested by `ACoRA`.

`code_compatibility` – `ACoRA` variants using `BERT4Code` and `CodeT5+` were able to correctly indicate the modified lines. However, the two remaining `CodeEmbedders` missed only one line each that were ranked at positions 101 and 373 by `CodeBERT` and `GraphCodeBERT`, respectively.

5.1.2. RQ2: Relevance of recommendations – Wireshark

We based the evaluation for Wireshark on the dataset containing lines from 3,475 revisions. We wanted to balance the number of entries in *reference* and *validation* databases. Therefore, we added the first 1,760 revisions (40,430 comments) to the former database and the remaining 1,715 revisions (46,850 comments) in the latter one. We made the split timewise (2 years each).

We performed three analyses using different thresholds for matching lines belonging to the *reference* and *validation* databases. The thresholds were calculated based on the distance measured between the `CodeEmbedder`'s embedding vectors representing the lines within the *reference database* (the distance to the closest line in the *reference database* other than the line itself). Such a strategy allowed us to avoid biasing the observations by choosing such thresholds arbitrarily. Finally, we applied the filtering criteria described in Section 4. As a result, we obtained 1,582 recommendations to be analyzed.

Figure 6 shows the quality of `ACoRA`'s recommendations (measured using the O measure) for `ACoRA` using `CodeT5+` (plots for the remaining `CodeEmbedder`s are available in the reproduction package) depending on the maximum distance threshold between the recommended and original lines, while the mean O values for all `CodeEmbedder`s are presented in Table 3. When we set the threshold to the 10th percentile of the distances in the *reference database*, 99.6% of suggestions were relevant (only true-positive suggestions), however, the threshold limited the number of recommendations to 232 lines only (ca. 15% of all recommendations). As we increased the distance threshold to the 50th and 100th percentile, the number of relevant recommendations decreased to 82.8% and 40.7% while the number of irrelevant recommendations increased to 5.4% and 15.9%, respectively. This shows, unsurprisingly, that the possibility of providing correct recommendations for reviewers strongly depends on the contents of the *reference database*. However, even for the worst-case scenario (i.e., always recommending the comment of best-matching line in the *reference database*), `ACoRA` was able to provide at least partially relevant recommendations for 84.1% of the cases. These observations were consistent among other `CodeEmbedder`s, with only minor differences in their accuracy of recommendations or the number of recommended lines depending on the threshold. We observed a trade-off between the number of recommended lines and the number of correct suggestions. Therefore, we cannot firmly state that either of the models is unanimously superior.

Table 3. Evaluation of the overlap of recommended vs. actual categories for Wireshark (all `CodeEmbedder`s)

	$\leq$10th perc. dist.		$\leq$50th perc. dist.		$\leq$100th perc. dist.	
`CodeEmbedder`	n	mean O	n	mean O	n	mean O
`BERT4Code`	233	0.98	465	0.78	1582	0.61
`CodeBERT`	233	0.99	366	0.83	1553	0.60
`GraphCodeBERT`	237	0.99	324	0.85	1609	0.61
`CodeT5+`	232	0.998	297	0.88	1474	0.61

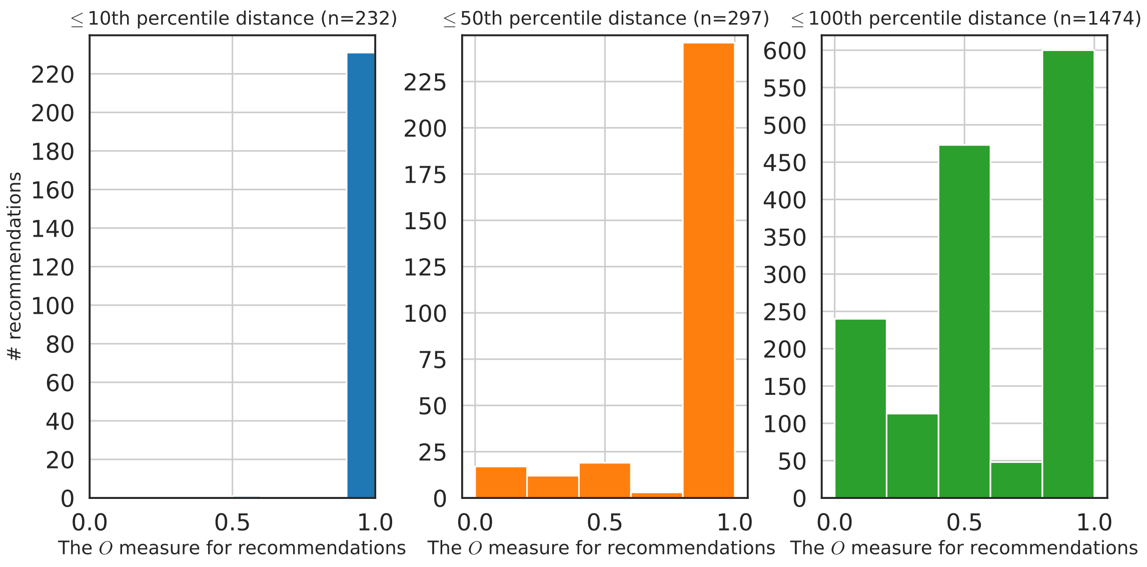


Figure 6. Evaluation of the overlap of recommended vs. actual categories for Wireshark (`CodeT5+`)

5.2. Cloudera

5.2.1. RQ1: Finding similar lines – Cloudera

We modified up to 10 lines for each of the categories in our taxonomy and searched for similar lines in the dataset of 35 000 lines of code coming from Cloudera. Figure 7 presents the distributions of ranking positions of the original lines recommended by `ACoRA` using `CodeT5+` while Table 4 summarizes the results for all `CodeEmbedders`. Similar to Wireshark, most of the original lines were found as the most similar to their modified counterparts.

`config_commit_patch_review` – most of the lines belonging to this category were comments. Even for the recommendations having the original line ranked at 11+ position, the top suggestions seemed justifiable. For instance, the top recommendation for the line "`// initial transaction.`" modified by adding an e-mail address at the end was "`/// @note The replication factor should be an odd number and range in`" – the presence of `@` that was in the added e-mail address could have made the line more similar than the original line. The second case was the line "`ASSERT_FALSE(empty has_user());`" modified by negating the parameter of the call (`!empty...`). Although the original line was ranked at the 16th position, the other 13 top suggestions were also the assert functions calls.

`code_logic` – `ACoRA` variants using `CodeT5+`, `CodeBERT`, and `GraphCodeBERT` correctly indicated all modified lines as top suggestions, while `ACoRA` using `BERT4Code` missed two lines. The first one was the line "`if (schema_elem.__isset.field_id) {`" modified by negating the condition (`!`). The first six recommendations were also if statements with negations, e.g., the top recommendation was the line"`if (!step.has_add_column()) {`". The second line was "`return percent/4;`" with the operator changed to multiplication (`*`). All recommendations were return statements. However, the top recommendation contained a pointer reference instead of the multiplication "`return *this;`"

`code_data` – `ACoRA` variants using `CodeT5+` and `GraphCodeBERT` correctly indicated modified lines, however, the two remaining `CodeEmbedders` missed only up to two lines. For

Table 4. Similarity of modified lines to their original counterparts for Cloudera (all `CodeEmbedders`)

`CodeEmbedder`/taxonomy category	original line similarity rank										
config_commit_patch_review	1	2	3	4	5	6	7	8	9	10	11+
`BERT4Code`	7	1	1	–	–	–	–	–	–	–	2
`CodeBERT`	9	–	–	–	–	–	–	–	–	–	2
`GraphCodeBERT`	8	–	–	–	–	–	–	–	–	–	3
`CodeT5+`	8	1	1	–	–	–	–	–	–	–	1
code_logic	1	2	3	4	5	6	7	8	9	10	11+
`BERT4Code`	7	–	–	–	–	–	1	1	–	–	–
`CodeBERT`	9	–	–	–	–	–	–	–	–	–	–
`GraphCodeBERT`	9	–	–	–	–	–	–	–	–	–	–
`CodeT5+`	9	–	–	–	–	–	–	–	–	–	–
code_data	1	2	3	4	5	6	7	8	9	10	11+
`BERT4Code`	8	–	–	–	–	–	–	–	–	–	2
`CodeBERT`	9	–	–	–	–	–	–	–	–	–	1
`GraphCodeBERT`	10	–	–	–	–	–	–	–	–	–	–
`CodeT5+`	10	–	–	–	–	–	–	–	–	–	–
code_style	1	2	3	4	5	6	7	8	9	10	11+
`BERT4Code`	4	2	1	–	–	–	–	–	–	–	2
`CodeBERT`	9	–	–	–	–	–	–	–	–	–	–
`GraphCodeBERT`	8	–	–	–	–	–	–	–	–	–	1
`CodeT5+`	9	–	–	–	–	–	–	–	–	–	–
code_doc	1	2	3	4	5	6	7	8	9	10	11+
`BERT4Code`	2	–	–	–	–	–	–	–	–	–	7
`CodeBERT`	5	–	–	–	–	–	1	–	–	1	2
`GraphCodeBERT`	8	–	1	–	–	–	–	–	–	–	–
`CodeT5+`	9	–	–	–	–	–	–	–	–	–	–
code_io	1	2	3	4	5	6	7	8	9	10	11+
`BERT4Code`	6	1	–	1	–	–	–	–	–	–	2
`CodeBERT`	10	–	–	–	–	–	–	–	–	–	–
`GraphCodeBERT`	10	–	–	–	–	–	–	–	–	–	–
`CodeT5+`	9	–	–	–	1	–	–	–	–	–	–
code_api	1	2	3	4	5	6	7	8	9	10	11+
`BERT4Code`	10	–	–	–	–	–	–	–	–	–	–
`CodeBERT`	6	–	1	–	–	–	–	–	–	–	3
`GraphCodeBERT`	10	–	–	–	–	–	–	–	–	–	–
`CodeT5+`	10	–	–	–	–	–	–	–	–	–	–
code_naming	1	2	3	4	5	6	7	8	9	10	11+
`BERT4Code`	5	–	1	–	–	–	–	–	–	1	2
`CodeBERT`	6	–	2	–	–	–	–	–	–	–	1
`GraphCodeBERT`	7	–	2	–	–	–	–	–	–	–	–
`CodeT5+`	9	–	–	–	–	–	–	–	–	–	–
code_design	1	2	3	4	5	6	7	8	9	10	11+
`BERT4Code`	8	–	–	–	–	–	–	–	–	–	1
`CodeBERT`	8	1	–	–	–	–	–	–	–	–	–
`GraphCodeBERT`	9	–	–	–	–	–	–	–	–	–	–
`CodeT5+`	9	–	–	–	–	–	–	–	–	–	–
compatibility	1	2	3	4	5	6	7	8	9	10	11+
`BERT4Code`	6	1	–	2	–	1	–	–	–	–	–
`CodeBERT`	6	2	–	–	1	–	–	–	–	–	1
`GraphCodeBERT`	10	–	–	–	–	–	–	–	–	–	–
`CodeT5+`	10	–	–	–	–	–	–	–	–	–	–

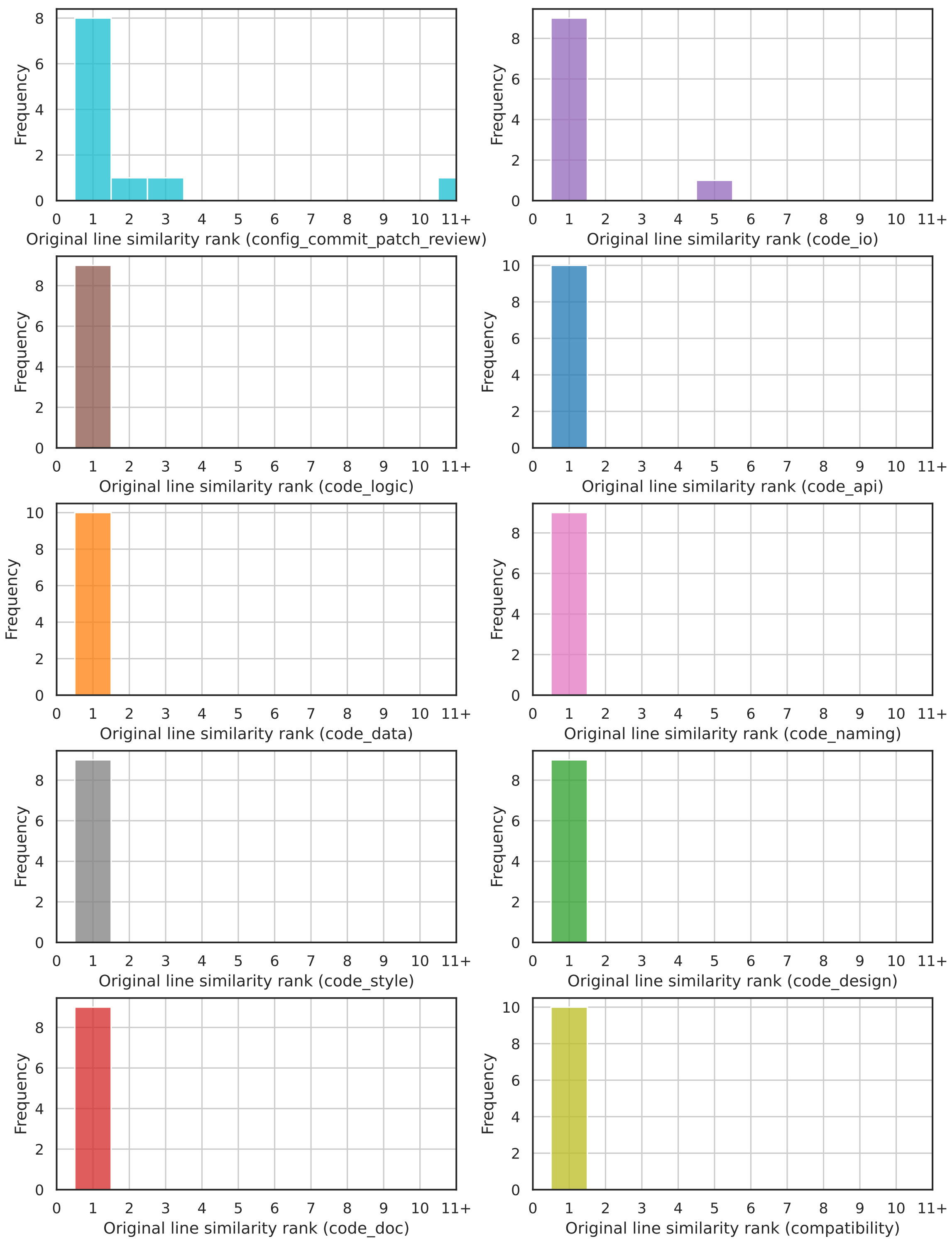


Figure 7. Similarity of modified lines to their original counterparts for Cloudera (`CodeT5+`)

`BERT4Code` two missed cases were the same line “`int percent = 0;`” modified by changing the declared type to `double` and `string`.

`code_style` – `ACoRA` variants using `CodeT5+` and `CodeBERT` correctly suggested all modified lines, while `GraphCodeBERT` missed one of the lines and recommended it as the 39th similar line. For ACoRA using BERT4Code, we observed more original lines recommended at high-ranking positions than for Wireshark, however, also the nature of changes made to the lines was slightly different. Two lines for which the top recommendations were the original lines were modified by removing a single indenting space (from six to five). Another one was adding an additional space between the and operator (&&) and function call ("&&␣ std::find..." to "&&␣␣std::find..."). Our general observation was that the the bigger the difference in the number of indention spaces the less similar the original and modified lines were. The extreme case was the line "␣␣␣␣␣␣␣␣int num_micro_batches = " with all indention spaces removed (the original line was provided as the 282nd recommendation).

code_doc – ACoRA variants using CodeT5+ and GraphCodeBERT correctly indicated ten and nine out of ten modified lines and performed visibly better than the remaining variants of ACoRA using CodeBERT and BERT4Code. For these two models, we made a similar observation to Wireshark that converting between commented and non-commented lines made them perceived as non-similar. For BERT4Code, two exceptions were (1) an inline comment changed to a single-line block comment and (2) an inline comment with preceding // modified to /// for which the original lines were provided as the first suggestions.

code_io – ACoRA variants using CodeBERT and GraphCodeBERT correctly indicated all ten modified lines as the most similar to the original ones. ACoRA using CodeT5+ made only one mistake, however, the top-suggested line was a similar logging statement to the original one. For ACoRA using BERT4Code changing the names to synonyms led to most modified lines being recognized as the most similar to their original counterparts. The two extreme cases were the lines "LOG(ERROR) << s;" and "return server_->Init();" modified by changing ERROR to PROBLEM and server to computer. Interestingly, for the latter, one of the top suggestions was the line "return client->DeleteTable(p.table_name);" containing a word client used in a similar context, which could mean that the word computer was perceived by the model to be more similar to the word client than to the word server.

code_api – three ACoRA variants using BERT4Code, CodeT5+, and GraphCodeBERT correctly indicated all ten modified lines as the most similar to the original ones. The exception was the ACoRA variant using CodeBERT which correctly suggested six lines but in three cases the modified line appeared beyond the first ten positions of the ranking (592, 328, and 943).

code_naming – only ACoRA variant using CodeT5+ was able to correctly indicate all modified lines as the most similar to the original ones. The variants using GraphCodeBERT and CodeBERT were slightly worse. For the variant of ACoRA using BERT4Code, we made similar observations as for Wireshark. However, we observed three lines for which the original lines were recommended as a 10+ option. The first one was changing the casing in the name of struct "struct TestData {" (TestData to test_data). The second one was changing "LOG.debug(..." to "log.debug(...". Interestingly, it turned out that a line "log.info(..." was found as one of the top three recommendations. The third line contained a call to a function "QUERY_OPT_FN(..." which identifier was changed to "query_opt_fn(...." These changes had the biggest impact on similarity among the three lines and the original line was ranked at the 307th position.

code_design – ACoRA variants using CodeT5+ and GraphCodeBERT correctly provided all the modified lines as top-suggestions. Both BERT4Code and CodeBERT did not provide correct suggestions for one line – "FragmentState* fragment_state;" modified to "RuntimeState* fragment_state;". Interestingly, the top suggestions were nearly identical lines "RuntimeState* state;" and "Server State* server;.

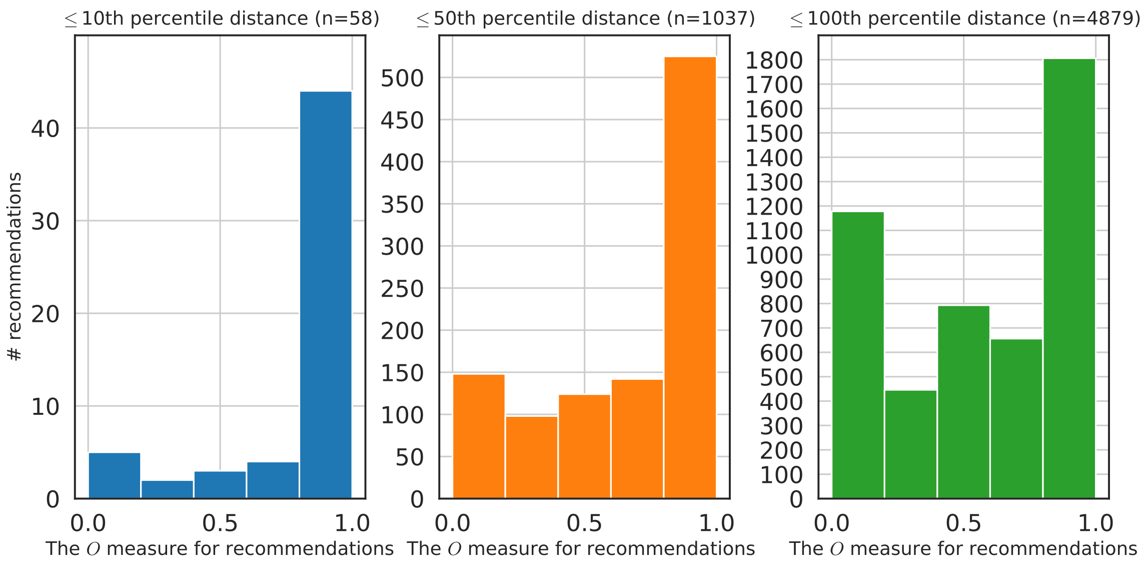


Figure 8. Evaluation of the overlap of recommended vs. actual categories for lines for Cloudera (`CodeT5+`)

`compatibility` – again, `ACoRA` variants using `CodeT5+` and `GraphCodeBERT` correctly provided all the modified lines as top-suggestions. For `BERT4Code`, we made a similar observation as for Wireshark that adding a suffix `_v2` to identifiers allowed for recognizing the original lines as very similar. Even for the cases where the top suggestion was not the original line, all of the recommended lines had a very similar structure. For instance, for the modified line "`import java.io.IOException_v2;`" all six suggestions were imports with "`import java.io.InputStreamReader;`" as the top recommendation.

5.2.2. RQ2: Relevance of recommendations – Cloudera

We followed the procedure described in Section 4 to select a sample of 29 599 lines as a *reference database* and 51 024 lines as the *validation database*. For Cloudera, we focused on scaling up the size of the validation dataset. The *reference database* was extracted from the 74K review comments that we initially fetched from the Gerrit instance, while the lines included in the *validation database* were extracted from the remaining (ca. 174K) comments fetched in later runs. Next, we applied the filtering criteria, which resulted in 4738 recommendations to be analyzed.

Table 5 presents the mean O values for all `CodeEmbedder`s depending on the similarity distance thresholds. The mean O value for the 10th percentile ranges between 0.69 and 0.89, dropping to 0.53–0.57 for the 100th percentile. The plot in Figure 8 shows that when all the recommendations were considered, the `ACoRA` variant using `CodeT5+` provided irrelevant suggestions in 23.7% of the lines (1158 lines), i.e., the overlap between the actual categories and the recommended categories was 0%. For 76.3% of the lines (3721 lines), `ACoRA` recommended at least one of the categories correctly. For 35.0% of the lines (1707 lines), the match was fully correct, i.e., the recommended categories were the same as the actual categories of the comment. Once we lowered the distance threshold (a minimum distance between the embedding vectors), we observed that the relevance of the recommendations increased. For the threshold equal to the 50th percentile of the distances in the *reference database*, the percentage of relevant recommendations increased to 46.5%, and for the most strict threshold corresponding to the 10th percentile, 72.4% of suggestions were relevant. Again, we observed a trade-off between the accuracy of the `ACoRA` and the number of recommended lines depending on the choice of a `CodeEmbedder` model.

Table 5. Evaluation of the overlap of recommended vs. actual categories for Cloudera (all `CodeEmbedders`)

	$\leq$10th perc. dist.		$\leq$50th perc. dist.		$\leq$100th perc. dist.	
`CodeEmbedder`	n	mean O	n	mean O	n	mean O
`BERT4Code`	139	0.89	1997	0.57	4738	0.53
`CodeBERT`	85	0.69	1816	0.61	4343	0.54
`GraphCodeBERT`	62	0.73	1599	0.59	4513	0.55
`CodeT5+`	58	0.83	1037	0.68	4879	0.57

We can conclude that the recommendations provided by `ACoRA` are relevant for the majority of cases (76.3%) and that the differences are often observed in a few categories (1–3), even when no similarity threshold is used.

5.3. Evaluation at the industrial partner

When evaluating the tool at the industrial partner, we analyzed a number of commits:
1. Comment on keyword `const` in a function parameter list. The reviewer asked for removing the keyword, i.e., changing the parameter to a non-constant variable. `ACoRA` identified this line as well, with the recommendation that `code_logic` should be investigated. In the database of examples, there was only one similar comment, but it was used in another context; therefore most of the examples were not relevant.
2. Reviewer questioned the re-location of a code fragment – he/she asked whether the code was moved correctly. `ACoRA` could not isolate the code fragment which was relevant, instead identified most lines in the commit as problematic, with different suggestions. The examples were mostly not relevant.
3. A reviewer commented on the name of a function, asking for a change of the name (not the entire signature). `ACoRA` identified the line with the name of the function as problematic, but the suggestion was to fix the **code_logic** instead of `code_naming`.
4. Reviewer asked to change a number of `#define` statements to `const`. `ACoRA` identified all such statements correctly, e.g., `#define x 1` and omit statements which should be omitted, e.g., `#define (x | y)`, which cannot be changed to a `const`. There were no relevant examples of the comments database and therefore the provided examples were not relevant.

During the discussion, we identified two major challenges for making this type of tool usable at the company.

The first challenge was the ability to capture the context of the code – not the lines before or after the commented line, but the ability to trace what has been done to the line in its context. For example:
- whether the line was newly added as part of the entire block or just a single line,
- whether the line was added as part of a large commit (e.g., more than three files were changed),
- whether the code block where the line is located has been in the code-base from the beginning or was added in a recent few commits (if it was not added in the commit-under-review).

– what was previously discussed in this code block, e.g., whether there was a discussion about a design solution for this block, naming conventions, etc.

Understanding the context of a reviewed line in this way would mimic the understanding of the context of the code reviews by human reviewers.

The second challenge was the ability to use meta-data in model training and prediction. The meta-data could contain the information of the context as in the first challenge, and the meta-information about the committed code fragment – its complexity, size, type of changed code (e.g., the role of the class/module), type of the system (e.g., safety-critical vs. web back-end). This information is available to the code reviewers as they know the system, but it is not available to tools like `ACoRA` (or even systems like CodeX [53]).

6. Implications for practitioners

The key takeaway from our study for practitioners is that a complex problem of automatic code review and repair can be simplified to a simpler problem of searching for similar lines to those under review and providing a summary of issues previously raised by reviewers to increase the accuracy of automatic code review support under the cost of increasing human involvement in the review process. Another general lesson from our study is that setting a similarity threshold while searching for similar lines of code based on language model embeddings is a critical success factor. Our study shows that such a threshold should not be set arbitrarily but should come from understanding how similar or different the code is in the considered codebase. To tackle this problem, we propose determining the threshold by sampling a codebase and using a percentile-based approach that allows for balancing recall (higher percentile) and precision (lower percentile). Finally, we show that the concept proposed by `ACoRA` can be implementedre using different `CodeEmbedder` models. Although neither of the studied Transformer-based models turned out to be unanimously superior, we suggest using large pre-trained models (e.g., `CodeT5+`) as a default option, however, in the scenario where a codebase is unique (e.g., contains only in-house developed components or developers use unique, company-specific coding guidelines), one might consider pre-training a `BERT4Code` model from scratch, as it can be easily done even using standard graphics processing units (GPU).

7. Validity evaluation

In our validity analysis, we use the framework advocated by Wohlin et al. [54], complemented with the threats specific to studies embedded in external context as prescribed by Weringa [17].

In the category of *construct validity*, our major threat is the use of machine-learning language models to extract features. Although it is modern technology, using word embedding networks does not allow us to construct a vocabulary manually. This means that we do not know whether language constructs important for programmers (e.g., keywords) are important for the neural network too. Our mitigation strategy is to study different techniques for feature extraction (presented in [55] and [56]). We have also examined the results of the similarity of lines, manually in this paper, in order to understand whether this is a threat in our case.

Another construct validity threat is our classification of comments. Although it is based on our previous studies [16] and the systematic review by [38], it is a generalization of a natural language in a specific context. To minimize the risk of bias towards a project-specific language, we used pre-trained models that provide the same classifications based on several projects.

The most important threat to *conclusion validity* is measuring the relevance of the suggestions provided by `ACoRA`. A single comment can relate to several issues belonging to different taxonomy categories. This means that there is a risk that this is a multi-label classification problem. Unfortunately, the multi-label classification makes the evaluation of `ACoRA` suggestions challenging, as they must be assessed across multiple categories simultaneously. Simplifying this assessment to a binary evaluation for individual categories might seem straightforward, but it fails to capture the nuanced reality of multi-faceted feedback. Therefore, we used a measure that captures the overlap between categories to tackle this issue. Also, to minimize the threat of making wrong conclusions, we manually analyzed a sample of suggestions with non-overlapping categories between suggested and actual comments.

When conducting the study, we chose to evaluate it at a company. We've selected one of our collaboration partners based on the domain – embedded systems, long experience with programming, and access to experienced architects (>10 years). However, there could be a threat to *internal validity* associated with how we worked with the company. Since `ACoRA` uses source code from a company to operate, we set it up to connect directly to the company's code repository at their premises. We extracted code changes and the associated comments, as prescribed by the process of using `ACoRA`. The time for that was limited due to access to the premises and experts, and therefore, we could not conduct this evaluation for an extensive period of time. We collected and analyzed the data on-premises while presenting the results off-site (via MS Teams). This could mean that there is a risk that we missed an important aspect of the evaluation or that we did not manage to select the most optimal code fragments to discuss (the selection was random).

Finally, since our evaluation is performed on two open source projects and one industrial project, there is a risk of being too specific, i.e., an *external validity* risk. We have considered this and therefore asked the company about this specific aspect, as well as we manually examined the results (random samples of results). We concluded that the company or project is not the decisive factor but the availability and quality of the data. We applied `ACoRA` on two other projects (one industrial and one open source), where we extracted a handful of comments only. The low number of data points did not provide any results, and therefore, they are not included, but we've learned about the limitations and, therefore, claim that the results are generalizable. They are generalizable to contexts where the number of commented code fragments is >1000 and when the comments are linked (in the tool) to code fragments and not to entire commits/patches. Also, we assume that the comments and lines of code in the historical database are similar to the lines in the code under review – thus, we generalize our findings to an intra-organization/process application of the proposed approach. More studies are needed to find if we can generalize findings across different projects. Such studies are planned for our future work. Also, we suspect that the accuracy and usefulness of the proposed approach might decrease even in an intra-project application scenario if the context changes visibly over time, making the historical database of comments invalid (e.g., the nature of the project changes significantly, and quality standards evolve). Another threat to external validity concerns the number of code lines selected per comment-taxonomy category while investigating the tool's ability to find similar lines for each category. We randomly queried the dataset to obtain representative samples of lines of code and comments belonging to particular taxonomy categories (within our dataset); however, we are not able to assess how well they cover the whole spectrum of lines/comments in these categories. Finally, we identified one more threat to external validity that regards the selection of "mutation" operations we applied to modify lines of code. We made these choices subjectively to ensure that they regard those code constructs that are decisive while categorizing a given line of code into a particular taxonomy category. However, we are aware that the variety of such code constructs that seem valid in the context of each category goes beyond the examples we provide.

8. Conclusions and future work

Code reviews are an integral part of modern software development, usually being integrated with the continuous integration/deployment pipelines. Although there are tools that automatically check the quality of source code, code reviews are still needed – they provide the ability to comment on design aspects that cannot be formalized into checking rules, they provide the ability to discuss design choices, and, not least important, they are a way of onboarding new project members.

Although it has already been found that we can pinpoint which code fragments (even down to a single line of code) would attract attention from reviewers, there is little support for providing suggestions on what to focus on. In this work, we address this problem by designing and evaluating a tool for automatically providing these suggestions based on the previous review discussions available in code repositories. The tool uses a machine-learning based language model and searches for similar lines of code to those under review that were previously commented on. It analyzes the previous reviewers' comments to indicate the aspects of the code the reviewer should focus on while reviewing a given fragment of code. The suggestions are based on company/community-specific culture and provide a way to speed up the review process while allowing new team members to understand the code and participate in the code review discussions.

By using two open-source projects and one industry project, we studied to which degree it is possible to provide code reviewers with a suggestion on what they should focus on when reviewing a given code fragment. The results show that the tool can give fully correct suggestions (only true positives) in 35%–41% of the fragments and partially correct suggestions in 76.3%–84.1% of the fragments. Compared to the state-of-the-art tools for code repair, this is higher but requires human intervention (it is the reviewer who has to review the code in the end). Also, we showed that one can control the recall and precision of such recommendations by changing a similarity threshold between the code fragments (for the distance between the line-embedding vectors). By setting the threshold to the 10th percentile of the distances in the reference dataset, we were able to increase the correctness of recommendations even to 72%–99%, however, at the cost of sacrificing the number of recommendations provided by `ACoRA`. Therefore, a key takeaway from our study for practitioners is that a complex problem of automatic code review and repair can be simplified to the problem of finding previously commented-on lines of code that are similar to those under review and summarizing the issues raised by reviewers. This approach allows for increasing the accuracy of automatic code review support, however, at the cost of increasing human involvement in the review process.

In future work, we plan to conduct a deeper analysis and comparison of machine-learning language models and code representations to study their impact on the accuracy of `ACoRA`. We plan to expand this study to design a pre-configured tool set-up to identify specific aspects, e.g., security vulnerabilities in source code and SQL injection checks, and evaluate them in an industrial context. In particular, we want to create portable (between contexts) reference databases designed to detect certain types of issues in the code.

CRediT authorship contribution statement

Mirosław Ochodek – Conceptualization, Methodology, Software, Data curation, Investigation, Visualization, Writing – original draft, Writing – review and editing.
Miroslaw Staron – Conceptualization, Methodology, Software, Data curation, Investigation, Visualization, Writing – original draft, Writing – review and editing.

Declaration of competing interest

The authors have no competing interests to declare that are relevant to the content of this article.

Data availability

The reproduction package for the study containing datasets and scripts used to perform the analyses (with the exclusion of confidential data) is publicly available. Reproduction package – https://zenodo.org/records/13870908.

Funding

This research has been supported by Software Center (www.software-center.se), Chalmers | University of Gothenburg and by National Science Centre, Poland within the research project "Source code representations for machine-learning-based identification of defective code fragments" (OPUS 21), registered with the no. 2021/41/B/ST6/02510.

References

[1] P.C. Rigby and C. Bird, "Convergent contemporary software peer review practices," in *Proceedings of the 9th Joint Meeting on Foundations of Software Engineering*, 2013, pp. 202–212.
[2] L. MacLeod, M. Greiler, M.A. Storey, C. Bird, and J. Czerwonka, "Code reviewing in the trenches: Challenges and best practices," *IEEE Software*, Vol. 35, No. 4, 2017, pp. 34–42.
[3] M.E. Fagan, "Design and code inspections to reduce errors in program development," *IBM Systems Journal*, Vol. 15, No. 3, 1976, pp. 182–211.
[4] L. Milanesio, *Learning Gerrit Code Review*. Packt Publishing, Ltd., 2013.
[5] M. Meyer, "Continuous integration and its tools," *IEEE Software*, Vol. 31, No. 3, 2014, pp. 14–16.

[6] M. Staron, W. Meding, O. Söder, and M. Bäck, “Measurement and impact factors of speed of reviews and integration in continuous software engineering,” *Foundations of Computing and Decision Sciences*, Vol. 43, No. 4, 2018, pp. 281–303.
[7] J. Czerwonka, M. Greiler, and J. Tilford, “Code reviews do not find bugs. How the current code review best practice slows us down,” in *IEEE/ACM 37th International Conference on Software Engineering*, Vol. 2. IEEE, 2015, pp. 27, 28.
[8] F. Huq, M. Hasan, M.M.A. Haque, S. Mahbub, A. Iqbal et al., “Review4Repair: Code review aided automatic program repairing,” *Information and Software Technology*, Vol. 143, 2022, p. 106765.
[9] M. Hasan, A. Iqbal, M.R.U. Islam, A. Rahman, and A. Bosu, “Using a balanced scorecard to identify opportunities to improve code review effectiveness: An industrial experience report,” *Empirical Software Engineering*, Vol. 26, No. 6, 2021, pp. 1–34.
[10] M. Staron, M. Ochodek, W. Meding, and O. Söder, “Using machine learning to identify code fragments for manual review,” in *46th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*. IEEE, 2020, pp. 513–516.
[11] D.S. Mendonça and M. Kalinowski, “An empirical investigation on the challenges of creating custom static analysis rules for defect localization,” *Software Quality Journal*, 2022, pp. 1–28.
[12] A. Bacchelli and C. Bird, “Expectations, outcomes, and challenges of modern code review,” in *2013 35th International Conference on Software Engineering (ICSE)*. IEEE, 2013, pp. 712–721.
[13] N. Fatima, S. Nazir, and S. Chuprat, “Knowledge sharing, a key sustainable practice is on risk: An insight from modern code review,” in *IEEE 6th International Conference on Engineering Technologies and Applied Sciences (ICETAS)*. IEEE, 2019, pp. 1–6.
[14] A. Hindle, E.T. Barr, M. Gabel, Z. Su, and P. Devanbu, “On the naturalness of software,” *Communications of the ACM*, Vol. 59, No. 5, 2016, pp. 122–131.
[15] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones et al., “Attention is all you need,” *arXiv preprint arXiv:1706.03762*, 2017.
[16] M. Ochodek, M. Staron, W. Meding, and O. Söder, “Automated code review comment classification to improve modern code reviews,” in *International Conference on Software Quality*. Springer, 2022, pp. 23–40.
[17] R. Wieringa, *Design Science Methodology for Information Systems and Software Engineering*, 2014. [Online]. http://portal.acm.org/citation.cfm?doid=1810295.1810446
[18] M. Allamanis, E.T. Barr, P. Devanbu, and C. Sutton, “A survey of machine learning for big code and naturalness,” *ACM Computing Surveys (CSUR)*, Vol. 51, No. 4, 2018, pp. 1–37.
[19] U. Alon, M. Zilberstein, O. Levy, and E. Yahav, “code2vec: Learning distributed representations of code,” *Proceedings of the ACM on Programming Languages*, Vol. 3, No. POPL, 2019, pp. 1–29.
[20] J. Devlin, M.W. Chang, K. Lee, and K. Toutanova, “Bert: Pre-training of deep bidirectional transformers for language understanding,” *arXiv preprint arXiv:1810.04805*, 2018.
[21] Z. Feng, D. Guo, D. Tang, N. Duan, X. Feng et al., “Codebert: A pre-trained model for programming and natural languages,” *arXiv preprint arXiv:2002.08155*, 2020.
[22] B. Roziere, M.A. Lachaux, M. Szafraniec, and G. Lample, “DOBF: A deobfuscation pre-training objective for programming languages,” *arXiv preprint arXiv:2102.07492*, 2021.
[23] Y. Li, D. Choi, J. Chung, N. Kushman, J. Schrittwieser et al., “Competition-level code generation with alphacode,” *arXiv preprint arXiv:2203.07814*, 2022.
[24] F. Huq, M. Hasan, M.M.A. Haque, S. Mahbub, A. Iqbal et al., “Review4repair: Code review aided automatic program repairing,” *Information and Software Technology*, Vol. 143, 2022, p. 106765. [Online]. https://www.sciencedirect.com/science/article/pii/S0950584921002111
[25] R. Tufano, S. Masiero, A. Mastropaolo, L. Pascarella, D. Poshyvanyk et al., “Using pre-trained models to boost code review automation,” in *Proceedings of the 44th International Conference on Software Engineering*, 2022, pp. 2291–2302.
[26] C. Sadowski, E. Söderberg, L. Church, M. Sipko, and A. Bacchelli, “Modern code review: a case study at google,” in *Proceedings of the 40th International Conference on Software Engineering: Software Engineering in Practice*, 2018, pp. 181–190.
[27] A. Ram, A.A. Sawant, M. Castelluccio, and A. Bacchelli, “What makes a code change easier to review: An empirical investigation on code change reviewability,” in *Proceedings of the 26th*

ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, 2018, pp. 201–212.

[28] Y. Arafat, S. Sumbul, and H. Shamma, "Categorizing code review comments using machine learning," in *Proceedings of Sixth International Congress on Information and Communication Technology*. Springer, 2022, pp. 195–206.

[29] Z. Li, Y. Yu, G. Yin, T. Wang, Q. Fan et al., "Automatic classification of review comments in pull-based development model," in *International Conferences on Software Engineering and Knowledge Engineering*. KSI Research Inc. and Knowledge Systems Institute Graduate School, 2017.

[30] Z.X. Li, Y. Yu, G. Yin, T. Wang, and H.M. Wang, "What are they talking about? Analyzing code reviews in pull-based development model," *Journal of Computer Science and Technology*, Vol. 32, 2017, pp. 1060–1075.

[31] L. Yang, J. Xu, Y. Zhang, H. Zhang, and A. Bacchelli, "EvaCRC: evaluating code review comments," in *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2023, pp. 275–287.

[32] J.K. Siow, C. Gao, L. Fan, S. Chen, and Y. Liu, "Core: Automating review recommendation for code changes," in *IEEE 27th International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 2020, pp. 284–295.

[33] R. Brito and M.T. Valente, "RAID: Tool support for refactoring-aware code reviews," in *IEEE/ACM 29th International Conference on Program Comprehension (ICPC)*. IEEE, 2021, pp. 265–275.

[34] R. Tufano, O. Dabić, A. Mastropaolo, M. Ciniselli, and G. Bavota, "Code review automation: Strengths and weaknesses of the state of the art," *IEEE Transactions on Software Engineering*, 2024.

[35] Y. Hong, C. Tantithamthavorn, P. Thongtanunam, and A. Aleti, "Commentfinder: A simpler, faster, more accurate code review comments recommendation," in *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2022, pp. 507–519.

[36] D. Badampudi, R. Britto, and M. Unterkalmsteiner, "Modern code reviews – Preliminary results of a systematic mapping study," *Proceedings of the Evaluation and Assessment on Software Engineering*, 2019, pp. 340–345.

[37] D. Badampudi, M. Unterkalmsteiner, and R. Britto, "Modern code reviews – Survey of literature and practice," *ACM Transactions on Software Engineering and Methodology*, Vol. 32, No. 4, 2023, pp. 1–61.

[38] N. Davila and I. Nunes, "A systematic literature review and taxonomy of modern code review," *Journal of Systems and Software*, Vol. 177, 2021, p. 110951.

[39] H.A. Çetin, E. Doğan, and E. Tüzün, "A review of code reviewer recommendation studies: Challenges and future directions," *Science of Computer Programming*, Vol. 208, 2021, p. 102652.

[40] B. Roziere, M.A. Lachaux, L. Chanussot, and G. Lample, "Unsupervised translation of programming languages," *Advances in Neural Information Processing Systems*, Vol. 33, 2020, pp. 20 601–20 611.

[41] Y. Wu, M. Schuster, Z. Chen, Q.V. Le, M. Norouzi et al., "Google's neural machine translation system: Bridging the gap between human and machine translation," *arXiv preprint arXiv:1609.08144*, 2016.

[42] M. Ochodek, M. Staron, D. Bargowski, W. Meding, and R. Hebig, "Using machine learning to design a flexible loc counter," in *2017 IEEE Workshop on Machine Learning Techniques for Software Quality Evaluation (MaLTeSQuE)*. IEEE, 2017, pp. 14–20.

[43] M. Ochodek, R. Hebig, W. Meding, G. Frost, and M. Staron, "Recognizing lines of code violating company-specific coding guidelines using machine learning," *Empirical Software Engineering*, Vol. 25, No. 1, 2020, pp. 220–265.

[44] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi et al., "Roberta: A robustly optimized bert pretraining approach," *arXiv preprint arXiv:1907.11692*, 2019.

[45] I. Turc, M.W. Chang, K. Lee, and K. Toutanova, "Well-read students learn better: On the importance of pre-training compact models," *arXiv preprint arXiv:1908.08962*, 2019.

[46] H. Akoglu, "User's guide to correlation coefficients," *Turkish journal of emergency medicine*, Vol. 18, No. 3, 2018, pp. 91–93.
[47] J.N. Mandrekar, "Receiver operating characteristic curve in diagnostic test assessment," *Journal of Thoracic Oncology*, Vol. 5, No. 9, 2010, pp. 1315–1316.
[48] M. Staron, *Action research in software engineering*. Springer, 2020.
[49] S.K. Pandey, M. Staron, J. Horkoff, M. Ochodek, N. Mucci et al., "TransDPR: design pattern recognition using programming language models," in *ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*. IEEE, 2023, pp. 1–7.
[50] D. Guo, S. Ren, S. Lu, Z. Feng, D. Tang et al., "Graphcodebert: Pre-training code representations with data flow," *arXiv preprint arXiv:2009.08366*, 2020.
[51] Y. Wang, H. Le, A.D. Gotmare, N.D. Bui, J. Li et al., "CodeT5+: Open code large language models for code understanding and generation," *arXiv preprint arXiv:2305.07922*, 2023.
[52] V. Antinyan, M. Staron, A. Sandberg, and J. Hansson, "Validating software measures using action research a method and industrial experiences," in *Proceedings of the 20th International Conference on Evaluation and Assessment in Software Engineering*, 2016, pp. 1–10.
[53] M. Chen, J. Tworek, H. Jun, Q. Yuan, H.P.d.O. Pinto et al., "Evaluating large language models trained on code," *arXiv preprint arXiv:2107.03374*, 2021.
[54] C. Wohlin, P. Runeson, M. Höst, M.C. Ohlsson, B. Regnell et al., *Experimentation in software engineering*. Springer Science and Business Media, 2012.
[55] K.W. Al-Sabbagh, M. Staron, M. Ochodek, R. Hebig, and W. Meding, "Selective regression testing based on big data: Comparing feature extraction techniques," in *IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*. IEEE, 2020, pp. 322–329.
[56] M. Staron, W. Meding, O. Söder, and M. Ochodek, "Improving quality of code review datasets – Token-based feature extraction method," in *International Conference on Software Quality*. Springer, 2021, pp. 81–93.

Authors and affiliations

Mirosław Ochodek
e-mail: miroslaw.ochodek@put.poznan.pl
ORCID: https://orcid.org/0000-0002-9103-717X
Institute of Computing Science,
Poznań University of Technology, Poland

Miroslaw Staron
e-mail: miroslaw.staron@gu.se
ORCID: https://orcid.org/0000-0002-9052-0864
IT Faculty, Chalmers University of Technology |
University of Gothenburg, Sweden

e-Informatica Software Engineering Journal
Volume 19, Issue 1, pages: 1–44
DOI: 10.37190/e-Inf250103

BibTeX

A Comparative Analysis of Metaheuristic Feature Selection Methods in Software Vulnerability Prediction

Deepali Bassi*, Hardeep Singh*
*Corresponding authors: deepalics.rsh@gndu.ac.in, hardeep.dcse@gndu.ac.in

Article info

Dataset link:
https://drive.google.com/file/d/14qoY98ZPiWalupRUquMALR_HHChuSaPB/view?usp=sharing

Keywords:
software vulnerability prediction
metaheuristic feature selection
SMOTE
machine learning algorithms

Submitted: 6 Jun. 2024
Revised: 21 Sep. 2024
Accepted: 2 Oct. 2024
Available online: 18 Nov. 2024

Abstract

Background: Early identification of software vulnerabilities is an intrinsic step in achieving software security. In the era of artificial intelligence, software vulnerability prediction models (VPMs) are created using machine learning and deep learning approaches. The effectiveness of these models aids in increasing the quality of the software. The handling of imbalanced datasets and dimensionality reduction are important aspects that affect the performance of VPMs.
Aim: The current study applies novel metaheuristic approaches for feature subset selection.
Method: This paper performs a comparative analysis of forty-eight combinations of eight machine learning techniques and six metaheuristic feature selection methods on four public datasets.
Results: The experimental results reveal that VPMs productivity is upgraded after the application of the feature selection methods for both metrics-based and text-mining-based datasets. Additionally, the study has applied Wilcoxon signed-rank test to the results of metrics-based and text-features-based VPMs to evaluate which outperformed the other. Furthermore, it discovers the best-performing feature selection algorithm based on AUC for each dataset. Finally, this paper has performed better than the benchmark studies in terms of F_1-score.
Conclusion: The results conclude that GWO has performed satisfactorily for all the datasets.

1. Introduction

The prediction of software vulnerabilities composes an essential step to provide software quality and security. Vulnerability, according to ISO/IEC 27000:2018, is a flaw in a control or asset that one or more threats could exploit. A few instances that illustrate the harm caused by software vulnerabilities are the open-source programs Heartbleed, ShellShock, and Apache Commons; well-known web browser plugins like Adobe Flash Player and Oracle Java. Millions of internet users have had their security jeopardized by browser plugins, and thousands of businesses and clients worldwide have been put in danger by open-source software. Furthermore, 1.7 million USD in financial damages have also been reported [1] as a result of software malfunction. Organizations had to pay 1.4 million USD in 2017 and 1.3 million USD in 2018 to cope with cyberattacks as a result of cybercrimes [2]. The National Institute of Standards and Technology (NIST) documented an exponential growth in software vulnerabilities since 2016 [3]. Software developers' negligence regarding the security facets during the initial phases of development causes security issues in later stages. Vulnerabilities provide possibilities for attackers to perform criminal activities and their sales at a very high price [4]. They are known to be the subgroup of faults, as are less in number than faults [5]. A fault in the software specification, development, or configuration is considered a vulnerability if the security policy is violated during its execution [6]. Detection and fixing of vulnerabilities before the deployment stage aids in reducing testing costs and maintaining their market reputation.

The vulnerability prediction models (VPMs) are devised to predict vulnerable and non-vulnerable components and therefore, the quality of these models is essential for the security of the software systems. Researchers and engineers are striving to build accurate VPMs, thus ensuring the quality and security of the systems. Research studies have previously used software assurance techniques such as static analysis, dynamic analysis, fuzz-testing, code inspections, etc. to identify security vulnerabilities [7]. Due to the huge time consumption and high false positive rate problems in conventional techniques, machine learning and deep learning-based VPMs gained interest. Commonly, VPMs are executed in the testing phase of the overall Software Development Life-Cycle (SDLC) in order to prioritize the inspection efforts (e.g., static analysis testing, dynamic testing, etc.). The model will identify which files are likely to have vulnerabilities if their features are collected at the file level, while vulnerable methods may be detected if the dataset is generated at the method level. Some studies on vulnerability prediction used text mining data, while others used software measures similar to those used in fault prediction models [8].

Hence, VPMs can be modeled as metrics-based, text-mining-based, or a combination of both datasets. In metrics-based VPMs, the components are determined using software metrics, e.g., cohesion, coupling, and complexity metrics that predict the vulnerability [9]. In text-mining-based VPMs, source code is converted into tokens and these tokens or text features predict the vulnerable components [10]. Vulnerability datasets are imbalanced which leads to biases in the prediction models as the majority class is favored over the minority class [11]. Most studies have used data balancing methods to handle class imbalance problems [8,12] and hyperparameter optimization (HPO) methods for choosing the optimal hyperparameters of classifiers [13–17].

Machine learning techniques also face the problem of the high dimensionality of the dataset. The performance of classifiers degrades when the classification parameters are increased. Therefore, to improve the efficacy of the model, the feature subset size should be decreased. Dimensionality reduction is an effective method to remove redundant or irrelevant features and thereby upgrade the performance, lowering computational complexity, constructing generalized models, and reducing storage [18,19]. Two major approaches have been proposed for dimensionality reduction: feature synthesis and feature selection [20].

In the case of feature synthesis, dimensional space is transformed from high to low whereas, in feature selection, a subset of given features is chosen by removing redundant or features with no predictive information. Feature selection methods are categorized as a filter, wrapper, hybrid, and embedded. Further two methods exist to describe how the features can be evaluated such as feature ranking and subset selection. In the feature ranking method, each feature is given a score based on some criteria, and the features with insufficient scores are removed [21]. In the subset selection method, an optimal subset is found out of all possible subsets. If there are n initial features, the search space for the best subset comprises all feature subsets, which is equal to 2n different states. In other words, the value of each property is evaluated independently in the property ranking algorithms, and the relationship between characteristics is not taken into account.

Feature selection algorithms based on metaheuristics are emerging in the field of vulnerability prediction [22–24]. These methods presume that the features are independent of one another and that any potential relationship between the features is ignored. Although this basic assumption decreases the computational complexity of the feature selection approach, it may reduce its performance in many circumstances. Choosing a feature subset is an NP-Hard task. The best subset can be identified simply by assessing all feasible subsets using an exhaustive search approach. Although this method guarantees an optimal feature subset, even for medium-sized datasets, finding the optimal answer is time-consuming and even impractical. Because evaluating all feasible subsets is prohibitively expensive, a feature subset must be searched that is acceptable in terms of both computing complexity and suitability. Metaheuristic algorithms are one technique to solve complex optimization and NP-Hard issues. Metaheuristic approaches are categorized as evolutionary algorithms and swarm intelligence (SI). SI algorithms used approximate and non-deterministic strategies to explore and exploit the search space to obtain near-optimal solutions. Swarm-based approaches are the most prevalent type of nature-inspired metaheuristic group [22].

1.1. Motivation

Efficient VPMs are important for ensuring the security and quality of the software [8]. Their performance is affected by the imbalanced datasets, the selection of optimal hyperparameters for machine learning algorithms, and the dimensionality of the datasets. Recent studies have worked on improving it by incorporating data balancing methods, HPO, reducing the dimensionality through feature synthesis, filter-based feature ranking, and also using metaheuristic algorithms for feature subset selection. In [23], the researchers have applied the dual HPO, where the problem of imbalanced datasets is handled and the selection of appropriate hyperparameters is done, to optimize VPMs. Research studies related to feature selection or dimensionality reduction have been explored for the past few years but metaheuristic feature selection methods have not been explored much.

To the best of our knowledge, we have come across two papers [24] and [25] that use such techniques. So, exploring such an area could be beneficial for the researchers to know the impact of the combination of metaheuristic feature selection and machine learning techniques on VPMs. The research paper [24] uses the grey-wolf optimization (GWO), particle swarm optimization (PSO), and genetic algorithm (GA) metaheuristic feature subset selection methods, random forest (RF) machine learner, and SMOTE resampling technique on metrics PHP dataset. In [25], Diploid Genetic algorithms with deep learning networks (Long Short Term Memory and Gradient Recurrent Unit) on software vulnerability prediction are applied. The deep SYMbiotic-based genetic algorithm model (DNN-SYMbiotic GAs) is used in this suggested method to solve challenges involving the prediction of software vulnerabilities. The current study is motivated to extend the paper [24] where it performs a comparative analysis of the various combinations of three other metaheuristic algorithms and seven other machine learning techniques. Moreover, it has not only worked on metrics datasets but also on text-features-based datasets.

1.2. Contributions

The contributions are as follows:

- This paper performs the experiments using two datasets; one is in PHP language and other in JavaScript. The PHP dataset [10] consists of three projects (Drupal, PHPMyAdmin and Moodle) and is released in two forms, i.e., software metrics and text features. The JavaScript dataset [12, 26] contains software components (methods) from several projects and contains also both metrics and text features. Since all the datasets are imbalanced therefore SMOTE resampling technique is applied to balance the datasets.
- The current study performs a comparative analysis of six metaheuristic algorithms such as PSO, GA, GWO, salp swarm algorithm (SSA), harris hawk optimization (HHO), and whale optimization algorithm (WOA) combined with eight machine learning algorithms namely random forest (RF), naïve bayes (NB), adaboost (AB), support vector machine (SVM), *k*-nearest neighbor (KNN), decision tree (DT), logistic regression (LR), and multilayer perceptron (MLP).

The paper has framed the following research questions:

RQ 1. Has all the metaheuristic feature selection and machine learning combinations improved the efficacy of VPMs?
Previous studies have experimented with improving the efficiency of VPMs using optimal hyperparameters settings [23], applying data balancing techniques and dimensionality reduction methods [20]. The research study [24] used feature selection algorithms namely PSO, GA, and GWO on only metrics-based PHP datasets with random forest machine learner and SMOTE technique. Second [25] has proposed a new VPM based on the SYMbiotic genetic algorithm and deep learning techniques on metrics-based PHP datasets. Through this question, the current study will explain whether all metaheuristic feature selection and machine learning combinations have improved the performance of VPMs.

RQ 2. Which one statistically performed better metrics-based or text-mining-based VPMs in the context of feature selection?
Walden et al. [10] have mentioned in their paper that text-mining-based VPMs have performed better than metrics-based VPMs for within-project prediction. In the case of cross-project prediction, metrics-based VPMs performed slightly better. The motive of this research question is to observe that on applying a metaheuristic feature selection method which VPM (metrics and text-mining) has significantly performed better. For the significant comparison, the paper applied Wilcoxon signed rank test.

RQ3. Which metaheuristic feature selection algorithm has performed the best?
Rhmann [24] has shown that PSO-RF has performed better than other benchmark studies. It has concluded that GA, PSO, and GWO-based RF outperformed the existing machine-learning algorithms. The current study applied all the possible combinations of eight machine learners with the six metaheuristic feature selection methods. The research question aims to find out which feature selection method has performed the best to help the researchers in improving the efficiency of VPMs.

1.3. Paper organization

The rest of the paper is structured as: Section 2 represents the research works related to the current study, Section 3 explains the background knowledge, Section 4 provides the research methodology, Section 5 provides the results, Section 6 discusses the results, Section 7 defines the threats to validity, and Section 8 concludes the paper. The appendix includes the detailed results tables and metrics description of the datasets.

2. Related work

To build effective VPMs research studies have emphasized the early prediction of vulnerable components using machine learning algorithms. Therefore, handling the factors affecting the performance of VPMs such as optimal hyperparameters, imbalanced datasets, feature selection, etc. is essential. In this section, we have discussed the research works that include VPMs and feature selection techniques.

Ghaffarian and Shahriari [7] have reviewed machine learning and data-mining techniques to curb the effect of software vulnerability. It has stated various software vulnerabilities. In addition, it has been mentioned that the vulnerability datasets are imbalanced and affect the efficiency of machine learning algorithms. Kaya et al. [8] show the impact of feature types, classifiers, and data-balancing techniques on VPMs. It has covered three feature types such as metrics, text, and a combination of metrics and text features. Experiments are performed using seven machine classifiers and four data sampling techniques on the PHP datasets namely Drupal, Moodle, and PHPMyAdmin. Evaluation is done through the performance metrics precision, recall, AUC, F_1-score, and specificity. The conclusion states that for smaller datasets Drupal and PHPMyAdmin, the random forest has outperformed other classifiers, and for larger datasets, i.e., Moodle, Rusboost has outshined.

Walden et al. [10] have created vulnerability datasets based on PHP open-source projects as most of the vulnerabilities are observed in web applications. The datasets include two feature types such as software metrics and text features. Random forest and under-sampling techniques are used for classification and balancing the dataset, respectively. The performance metrics used are recall and inspection ratio. The paper has experimented on both software metrics-based and text mining-based VPMs. In addition, it has been found that text-mining-based models outperformed metrics-based models. Ferenc et al. [12] predict the vulnerabilities in JavaScript programs using static code metrics. It has proposed the JavaScript dataset by extracting the vulnerability information from public databases such as Node Security Project, GitHub code fixing patches, and the Snyk platform. The paper has applied a grid search algorithm for parameter tuning, resampling techniques to balance the data, and eight machine learning algorithms including deep learning algorithms to find the best-performing VPMs. Finally, it concludes that the k-nearest neighbor has performed best with an F-measure of 0.76, over-sampling has improved recall but diminishes precision, and under-sampling increases precision and decreases recall.

Stuckman et al. [20] analyzed the impact of dimensionality reduction techniques (feature selection, principal component analysis, and confirmatory factor synthesis) on the productivity of VPMs. It has used Smote and under-sampling techniques for balancing the data and resulted in smote showing low recall, low inspection rate, and high f-measure therefore it is better than the under-sampling method. In addition, dimensionality reduction techniques performed well for cross-project prediction rather than within-project prediction. Chen et al. [21] have empirically analyzed the effect of feature selection methods on machine learning. It has used filter-based ranking methods due to the high cost of computation of other methods. The paper has applied ChiSquared, F-score, GainRatio, InfoGain, GiniIndex, FisherScore, and ReliefF filter-based ranking methods on three PHP datasets using a random forest machine classifier. The paper has shown an increase in the performance of VPMs.

Bassi and Singh [23] examine the effect of dual HPO on metrics-based VPMs. This study proposes an approach for optimizing hyperparameters for machine learners and data balancing strategies using the Python framework Optuna. It compared six combinations of five machine learners and five resampling approaches using default values and optimized hyperparameters for experimentation. The article discovered that dual HPO outperforms HPO on learners and HPO on data balancers. Furthermore, it investigated the impact of data complexity measures and concluded that HPO did not increase the performance of datasets with substantial overlap.

Rhmann [24] has proposed a new technique by combining the grey-wolf metaheuristic technique and random forest. The paper has emphasized finding the best subset of relevant features. It has shown that metaheuristic algorithms combined with random forest performed better than other machine learning algorithms. Particle swarm optimization with random forest outperforms the baseline methods. Sahin et al. [25] suggest a unique deep learning method and SYMbiotic Genetic algorithm for software vulnerability prediction. They have applied Diploid Genetic algorithms with deep learning networks on software vulnerability prediction. The deep SYMbiotic-based genetic algorithm model (DNN-SYMbiotic GAs) is employed in this suggested method to solve challenges involving the prediction of software vulnerabilities. On many benchmark datasets from the PHPMyAdmin, Moodle, and Drupal projects, extensive experiments are carried out. According to the results, the suggested approach (DNN-SYMbiotic GAs) improved vulnerability prediction, which implies better software quality prediction.

Viszkok et al. [26] have worked on the dataset produced by [12] by including the process metrics and observed that F-measure has improved by 8.4%, precision by 3.5%, and recall by 6.3%. Kalouptsoglou et al. [27] have used three feature types of metrics, text-tokens, and a combination of both to model the VPMs. It has proposed a text token-based dataset of JavaScript programs used in [12] and a new metric F_2-score. The paper concludes that text-tokens-based models perform better than metrics-based models in terms of F_2-score and the combination has not made much difference in the predictive performance.

Wang and Yao [28] tackled the class imbalance problem in defect prediction models by using under-sampling techniques, ensemble-learning techniques, and threshold moving techniques on naive bayes and random forest classifiers. The paper concludes that balanced random under-sampling shows a better defect prediction rate. Adaboost has performed best in increasing the efficacy of SDP models. The overall performance is measured using G-mean, AUC, and balance metrics. Shin et al. [29] evaluated the VPMs using code churn, complexity, and developer activity metrics on Linux and Firefox projects. The model has an inspection rate of less than 30% and a recall of 70%. Shin and Williams [30] analyze whether fault prediction models also work for vulnerability prediction by performing experiments on Mozilla Firefox which contains 21% of files with faults and 3% of files with vulnerabilities. It concluded that fault prediction models are equally capable as VPMs in predicting vulnerabilities. Furthermore, it suggested the models attain better recall and low false positives.

Lagerstrom et al. [31] experimented on the Google Chromium project to inspect the relationship that software vulnerabilities have with two types of metrics namely architecture coupling metrics and component-level metrics. The results reveal that vulnerable files are in correlation with both types of metrics. Zhang et al. [32] proposed a VULPREDICTOR that works on the combination of software metrics and text features. The paper performed experiments on three PHP datasets Drupal, Moodle, and PHPMyAdmin. It has achieved the F_1-score of 0.683 and EffectivenessRatio@20% to 75%. Abunadi et al. [33] explain how cross-project prediction techniques are useful in software vulnerability prediction. This paper results in the high recall, precision, and F-measure of J48 and random forest but has not applied data balancing techniques.

Khalid et al. [34] proposed NMPREDICTOR which consists of two tiers. The first tier contains 6 classifiers that are built on the training set of labeled metrics and text features files. In the second tier, a meta-classifier combining all 6 classifiers random forest, J48, and naïve bayes (both metrics and text) is built. This paper has experimented on PHP datasets and resulted in the highest F_1-score of 0.848. Catal et al. [35] implemented a web service for software vulnerability prediction. The paper uses the Azure machine learning platform to build the web service. Several machine learning algorithms are applied to the PHP datasets. For the performance evaluation, the Area under the ROC (AUC) metric is used. This paper concludes that the multilayer perceptron model has produced the best results.

Li and Shao [36] surveyed the prediction of software vulnerabilities using feature selection-based machine learning. This paper has classified the existing research works into 4 different feature types such as metrics, text mining features, graph, and taint analysis. It has discussed the advantages and challenges of machine learning in software vulnerability prediction. Solutions to the three main challenges namely selection of relevant features, class imbalance, and label data high cost are illustrated and further work has been discussed for the future. Rostami et al. [22] compares and categorizes various feature selection methods. The paper focuses on increasing the accuracy of prediction models through the use of metaheuristic algorithms. It has analyzed the performance of eleven swarm intelligence-based feature selection methods on six medical datasets from the UCI repository and three machine learning algorithms namely support vector machine, naïve bayes, and adaboost. In addition, it has discussed the pros and cons of each metaheuristic algorithm.

Apart from the above research studies, there are deep learning methods that are less sensitive to feature selection methods. Sonekalb et al. [37] provide an SLR which aims to do a detailed analysis and comparison of 32 primary works on DL-based vulnerability analysis of program code. It discovered a wide range of proposed analysis methods, code embeddings, and network topologies. They go over these strategies and alternatives in great depth and identify the current level of research in this field and suggest future directions by collating commonalities and contrasts in the techniques. To facilitate a stronger benchmarking of approaches, it also presents an overview of publicly available datasets. This SLR serves as an overview and jumping-off point for researchers interested in performing deep vulnerability analysis on program code.

Li et al. [38] have introduced VulDeePecker, the first deep learning-based vulnerability detection system, to relieve human experts from the tiresome and subjective labor of manually defining features and lowering the false negatives that are experienced by previous vulnerability detection systems. To assess the performance of VulDeePecker and other deep learning-based vulnerability detection systems that will be created in the future, they have gathered and made publicly available a relevant dataset. According to experimental findings, VulDeePecker can yield significantly fewer false negatives (with acceptable false positives) than other methods. They also applied VulDeePecker to 3 software products (Xen, Seamonkey, and Libav) and find 4 vulnerabilities that were "silently" patched by the vendors when they released later versions of these products but are not listed in the National Vulnerability Database; in contrast, these vulnerabilities are almost entirely missed by the other vulnerability detection systems they tested.

Zhou et al. [39] propose Devign, a general graph neural network-based model for graph-level classification through learning on a rich set of code semantic representations, which is inspired by the work on manually-defined patterns of vulnerabilities from various code representation graphs and the most recent development in graph neural networks. To effectively extract meaningful features from the learned rich node representations for graph-level classification, it contains a unique Conv module. The model is trained on manually labeled datasets constructed from four diverse, large-scale open-source C projects that use real source code with high levels of complexity and variation rather than the synthesis code employed in earlier efforts.

Li et al. [40] introduced a vulnerability detector, which can simultaneously achieve a high detection capability and a high locating precision, powered by deep learning called VulDeeLocator. The challenges while designing VulDeeLocator include how to support accurate control flows and variable define use relations, how to achieve high locating precision, and how to support semantic relations between the definitions of types as well as macros and their uses across files. They overcome these challenges by utilizing two novel concepts: (i) using intermediate code to accept more semantic information, and (ii) using the idea of granularity refinement to identify vulnerable areas. VulDeeLocator finds 18 verified vulnerabilities (also known as true-positives) when applied to 200 files randomly chosen from three different real-world software applications. Sixteen of these correspond to known vulnerabilities; the other two are not documented in the National Vulnerability Database (NVD) but have been "silently" corrected by Libav's manufacturer when fresh versions are released.

Kalouptsoglou et al. [41] explores if combining deep learning and software metrics might improve cross-project vulnerability prediction. Several machine learning models, including deep learning, are assessed and contrasted using a dataset of prominent real-world PHP software applications. Feature selection is evaluated for its impact on cross-project prediction. There investigation suggests that using software metrics and deep learning can improve vulnerability prediction models' performance across projects. The study found that cross-project prediction models perform better when projects share similar software metrics.

2.1. Comparisons with existing works

Table 1 compares our work with the existing works where PHP and JavaScript datasets are used. It describes the feature selection techniques, performance metrics applied, machine learning techniques used, whether data balancing is performed or not, features for modeling VPMs, i.e., metrics, text features, or combination of both. The current work is highly inspired by Rhmann et al. [24] and tried to work on different combinations of machine learning methods and metaheuristic feature selection methods. It is observed that the proposed work has tried to incorporate maximum performance metrics which suit the imbalanced datasets.

Previous works have included the PHP dataset and JavaScript datasets separately but this paper includes both to validate the work. In addition to this, it includes four performance metrics, eight machine learning algorithms, six feature selection methods, SMOTE resampling technique, two feature types, and finally two different language datasets with distinct granularity levels.

Table 1: Comparisons with existing works

Research papers	Feature selection techniques	Performance metrics	Features	Machine learning techniques used	Resampling techniques	Datasets used
Kaya et al. [8]	NO	AUC, Precision, Recall, F_1-score, Specificity	Software Metrics, Text Features, Combination of software metrics and text features	RF, AB, Linear Discriminant, Linear SVM, Weighted KNN, Subspace Discriminant, Rusboost	Smote, Adasyn, ClusterSmote, BLSmote	Drupal, Moodle, PHPMyAdmin
Walden et al. [10]	NO	Recall, Precision	Software Metrics, Text Features	RF	Under-sampling technique used (Weka SpreadSubsample)	Drupal, Moodle, PHPMyAdmin
Ferenc et al. [12]	NO	F-Measure	Software Metrics	Simple Deep Neural Network (DNNs), Complex Deep Neural Network (DNNc), KNN, SVM, RF, LR, Linear Regression, Gaussian NB(GNB), DT	Over-sampling with ratios (25%, 50%, 75%, 100%) and Under-sampling with ratios (25%, 50%, 75%, 100%)	JavaScript Dataset
Kudjo et al. [13]	NO	Precision, Recall, Accuracy	Software Metrics	RF, KNN, SVM, DT	No Resampling	Drupal, Moodle, PHPMyAdmin
Stuckman et al. [20]	Feature subset selection, Entropy Reduction, Principal Component Analysis (PCA), Sparse PCA, Confirmatory Factor Analysis (CFA)	Recall, F_1-score, Inspection Ratio	Software Metrics, Text Features	RF	Under-sampling, Smote	Drupal, Moodle, PHPMyAdmin
Rhmann et al. [24]	PSO, GWO, GA	Precision, Recall, F-Measure	Software Metrics	RF	SMOTE	Drupal, Moodle, PHPMyAdmin
Sahin et al. [25]	Symbiotic Genetic Algorithm (I and II)	Accuracy, F_1-score, Precision	Software Metrics	Artificial Neural Networks, Long-Short-Term-Memory (LSTM), Gated Recurrent Unit (GRU)	No Resampling	Drupal, Moodle, PHPMyAdmin

Table 1 continued

Research papers	Feature selection techniques	Performance metrics	Features	Machine learning techniques used	Resampling techniques	Datasets used
Viszkok et al. [26]	NO	Accuracy, Precision, Recall, F-Measure	Software Metrics	Simple Deep Neural Network (DNNs), Complex Deep Neural Network (DNNc), KNN, SVM, RF, LR, Linear Regression, Gaussian NB(GNB), DT	Over-sampling with ratios (25%, 50%, 75%, 100%) and Under-sampling with ratios (25%, 50%, 75%, 100%)	JavaScript Dataset
Kalouptsoglou et al. [27]	Point-BiSerial Correlation (PBSC)	Accuracy, Precision, Recall, F_1-score, F_2-score	Software Metrics, Text Features, Combination of software metrics and text features	DT, RF, NB, SVM, KNN, MLP	Random resampling	JavaScript Dataset
Zhang et al. [32]	NO	Precision, Recall, F_1-score	Combination of software metrics and text features	RF, NB, DT	No Resampling	Drupal, Moodle, PHPMyAdmin
Abunadi et al. [33]	NO	Precision, Recall, F-Measure	Software Metrics	NB, LR, SVM, RF, DT	No Resampling	Drupal, Moodle, PHPMyAdmin
Khalid et al. [34]	NO	Accuracy, Precision, Recall, F_1-score	Combination of software metrics and text features	RF, NB, DT	SMOTE	Drupal, Moodle, PHPMyAdmin
Catal et al. [35]	NO	AUC	Software Metrics	Averaged Perceptron, Bayes point machine, Boosted Decision Tree, Decision Forest, Decision jungle, Deep SVM, SVM, Logistic Regression, Multilayer Perceptron	No Resampling	Drupal, Moodle, PHPMyAdmin
Kalouptsoglou et al. [41]	Tree Based Elimination	Recall, Inspection Rate	Software Metrics	RF, SVM, MLP, XGBoost, Ensemble	Random UnderSampler	Drupal, Moodle, PHPMyAdmin
Current Work	JavaScript Dataset PSO, GA, GWO, HHO, SSA, WOA	AUC, Precision, Recall, F_1-score	Software Metrics, Text Features	DT, RF, NB, SVM, KNN, LR, AB, MLP	SMOTE	Drupal, Moodle, PHPMyAdmin, JavaScript Dataset

3. Background knowledge

This section contains a brief explanation of the machine learning techniques (Section 3.1), resampling techniques used for balancing the datasets (Section 3.2), feature selection algorithms (Section 3.3), and performance evaluation metrics (Section 3.4).

3.1. Machine learning techniques

Machine Learning Techniques majorly are supervised and unsupervised. Supervised machine learning algorithms work on labeled data and unsupervised works on unlabeled data. The current paper uses eight supervised machine learning algorithms namely, random forest, support vector machine, *k*-nearest neighbor, adaboost, naïve bayes, logistic regression, decision tree, and multilayer perceptron. For comparisons with baseline methods, we are using these algorithms.

3.1.1. Decision tree (DT)

Decision trees (DT) are non-parametric supervised machine learning algorithms [42]. This classifier is structured as a tree where internal nodes are the features, branches depict decision rules and each leaf node gives the outcome.

3.1.2. Random forest (RF)

The random forest (RF) algorithm gives the output by taking a majority of the votes from numerous decision trees. RF is simple and can handle large datasets efficiently [43].

3.1.3. Naïve Bayes (NB)

Naive Bayes is the supervised machine learning algorithm based on Bayes' theorem, assuming that there is independence among the features of the class. NB models are of four categories: Gaussian NB (GNB), Multinomial NB (MNB), Bernoulli NB (BNB), and Complement NB (CNB) [44].

3.1.4. Adaboost (AB)

Adaboost, called adaptive boosting, is the boosting algorithm where weak learners are sequentially added and trained by weighted training data to build strong classifiers. The classification output is predicted by calculating the mean weights of the weak classifiers. It can use different base learners to boost its performance but is affected by noisy data and outliers [45].

3.1.5. Support Vector Machine (SVM)

SVM is used to construct the best decision boundary called a hyperplane that separates multidimensional space into classes to place new data points in the correct class. SVM consists of various kernel functions (linear, polynomial, radial basis function, and sigmoid) used for the decision function. It has the overfitting issue, which arises when the number of features is much larger than the number of samples [46].

3.1.6. K-Nearest Neighbor (KNN)

KNN classifies the data points by calculating the distance among them. New data points are classified by comparing them with the stored data. The value of k is crucial to determine as a smaller value may cause underfitting and a larger value may cause overfitting [47].

3.1.7. Logistic regression (LR)

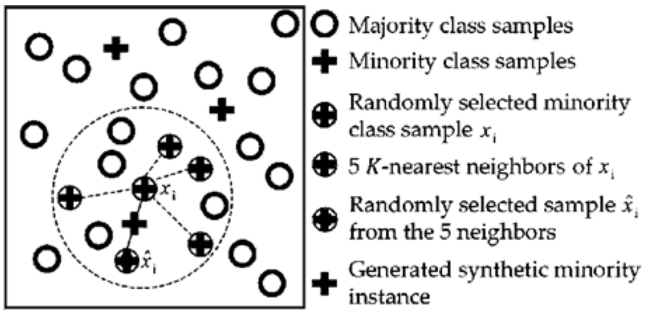
Logistic regression predicts the probability of the target variable. In other words, dependent variables are predicted through the independent variables set [48].

3.1.8. Multilayer perceptron (MLP)

Multilayer perceptron (MLP) is a feed-forward neural network that has three layers namely the input layer, hidden layer, and output layer. The input layer receives the signal for processing, the hidden layer acts as the computational engine and the output layer performs the prediction and classification tasks. MPs are used for non-linearly separable problems [49].

3.2. Resampling techniques

Resampling techniques are used to handle the class imbalance problem by balancing the datasets. The results may favor the majority class if the datasets are imbalanced. Therefore, to avoid biased results data balancing is important. Resampling Techniques are further classified as Under-sampling, Over-sampling, and Hybrid Sampling.



Majority class samples
Minority class samples
Randomly selected minority class sample x_i
5 K-nearest neighbors of x_i
Randomly selected sample $\hat{x}_i$ from the 5 neighbors
Generated synthetic minority instance
x_i
$\hat{x}_i$

Figure 1. SMOTE process

The current study balances the dataset using SMOTE technique because it may not lead to loss of data and is highly used in previous studies. SMOTE [50] executes the k-nearest neighbor algorithm for synthetic sample generation. First, the minority class vector is found; second, the value of k is selected; third, compute the distance between minority data points and any neighbor to plot a synthetic sample; and Lastly, the above step is repeated until the dataset is balanced. SMOTE obviates overfitting problems and increases the performance of the classifiers by generalizing the decision boundaries [51]. Figure 1 explains the SMOTE[1] process.

[1]https://rikunert.com/smote_explained

3.3. Feature selection algorithms

Previous studies have used feature synthesis algorithms for dimensionality reduction. Metaheuristic algorithms are of two types: Evolutionary Algorithms (EA) and Swarm Intelligence (SI). EA includes mechanisms based on biological evolution like mutation, reproduction, recombination, and selection. In EA, the initial population of individual solutions is generated randomly and a fitness function is used which is responsible for the quality of the solutions. After multiple iterations, the initial population evolved and reaches global optimization. Genetic algorithms are one example of evolutionary algorithms. SI is inspired by nature where each factor takes a simple task and exhibits a global intelligent behavior by having a factoring relationship with one another and their random reactions. There exists a plethora of SI-based feature selection algorithms such as particle swarm optimization (PSO), Gravitational Search Algorithm (GSA), Ant Colony Optimization (ACO), Differential Evolution (DE), Artificial Bee Colony Optimization (ABC), Firefly Algorithm (FA), Cuckoo Optimization Algorithm (COA), Bat Algorithm (BA), Grey Wolf Optimization (GWO), Salp Swarm Algorithm (SSA), Whale Optimization Algorithm (WOA), and Harris–Hawk Optimization (HHO).

This paper has incorporated 6 feature subset selection metaheuristic algorithms. It has applied three metaheuristic algorithms GA, PSO, and GWO from the base paper [24] but with seven other machine learning algorithms which are highly popular [2]. In addition to this, whale optimization (fish-based), harris hawk (bird-based), and salp swarm(sea-based) algorithms are mostly used in prediction areas like fault and defect prediction [52–55]]. Furthermore, the study used the feature selection code from GitHub[2] which contained 13 metaheuristic algorithms and we tried to implement the latest and swarm-based algorithms which were HHO, WOA, and SSA.

3.3.1. Genetic algorithms (GA)

A genetic algorithm is a population-based metaheuristic algorithm that imitates natural evolutionary mechanisms. It involves initial population production, selection of good solutions, fitness function definition, crossover, and mutation. Initial population generation includes all the possible solutions to the given problem. The fitness function assigns the fitness score to each individual and the individual with a higher fitness score has a higher chance of being selected. The selection phase creates a region with a high probability of producing the best solution. Reproduction has two operators: crossover and mutation. Crossover interchanges the genetic information of two parents for reproduction. The child population generated is the same in size as the parent population. New genetic information is added to a new child population by changing some bits in the chromosome called Mutation [22].

3.3.2. Particle swarm optimization (PSO)

Particle swarm optimization is inspired by the food search of a flock of birds or a school of fish. A bird flying in search of food randomly, and sharing its discovery can help in getting the best hunt for the entire flock. Each particle dynamically adjusts its velocity depending on its flying experiences and others in the group. Each particle keeps a record of its best result called pbest (personal best) and the best value of any particle called gbest (global

[2]https://github.com/JingweiToo/Wrapper-Feature-Selection-Toolbox-Python/tree/main/FS

best). The position of every particle is modified depending on its current position and velocity, the distance between pbest and its current position, and finally distance between gbest and its current position [22].

3.3.3. Grey Wolf optimization (GWO)

Grey Wolf is inspired by grey wolves and mimics the hunting process of grey wolves in nature. Grey wolves live in groups of 5–12 individuals and follow a hierarchical management system. The social hierarchy of grey wolves consists of Alpha (α) the head of the group and their orders are directions followed by other wolves. Beta (β) are the subordinates that aid α in making decisions. Delta (δ) are scouts which report to α and β. Lastly, Omega (ω) is at the bottom of the management system and is accountable for internal relationships. Grey Wolf Hunting has three phases: chasing and approaching the prey, encircling and harassing the prey, and attacking the prey [22].

3.3.4. Whale optimization algorithm (WOA)

A whale optimization algorithm is a metaheuristic algorithm inspired by the hunting mechanisms of humpback whales. It is easy to implement and robust. Humpback whales search for food in multidimensional space. The algorithm imitates the bubble-net foraging method of searching and attacking the prey by the whales. When the whale locates its prey, it forms a bubble-net spiral path and reaches upwards to the prey. There are three stages to explain predation behavior: surrounding the prey, bubble net attack, and hunting the prey [22].

3.3.5. Salp swarm algorithm (SSA)

A salp swarm algorithm is stimulated by the sea salps' swarming behavior [55]. The salp population is divided into leaders and followers. The salp chain is created when salps shape the swarm in heavy oceans. The leader lies in the front of the chain and the rest are the followers. The SSA algorithm starts with the initialization of the group of solutions randomly. Further, the iterative improvement process tries to reach the global optimum. Follower salps update their position based on the leader's position. To reach the desired solution the process is executed repeatedly [22].

3.3.6. Harris Hawk optimization (HHO)

Harris Hawk optimization algorithm is stimulated by the cooperative hunting behavior of harris hawks. They work in groups and attack the prey from all directions to surprise it. It has three stages which include surprise pounce, trailing the prey, and other attacking mechanisms. The first stage is Exploration which is to find and discover the prey or best candidate solution, second stage is the transformation from exploration to exploitation depending on the external/escaping energy of the prey. The third and final stage is Exploitation in which the prey is attacked hard or soft depending on the energy left with the prey [22].

3.4. Performance measures

The efficacy of the machine learning algorithms is evaluated using performance measures. Accuracy is measured by adding all correct predictions divided by the total predictions made by a machine learning algorithm. Since our datasets are imbalanced therefore we tend to use performance metrics that provide results without any biases [56]. The performance metrics Area under ROC curve (AUC), Precision, Recall, and F_1-score are highly used in previous studies so keeping in mind the comparative analysis, the current study also used the same metrics.

4. Research methodology

This section includes the experimental datasets, setup, performance results, statistical comparisons, and discussions of the results. Experiments are executed to evaluate the various combinations of metaheuristic feature selection and machine learning methods to analyze which combination has shown the highest performance.

4.1. Datasets

This paper uses eight datasets out of which six datasets (metrics and text-features) belong to PHP language and two datasets (metrics and text features) are based on JavaScript language. These are publicly available labeled datasets with labeling “NO” and “YES”. The paper focuses on binary classification hence the datasets are suitable for the purpose. The granularity level for PHP datasets is “file” whereas, for a JavaScript dataset, it is a method/function. This indicates whether the components for vulnerability prediction are files or methods. The experiments are first performed on the PHP dataset then to check the validity of the work, it is implemented on another publicly available JavaScript dataset.

– PHP Dataset[3]: Drupal is the content management system with 202 total files and 62 vulnerable files. Moodle is a learning management system with 2924 total files and 24 vulnerable files. PHPMyAdmin is an open-source administration for MySQL with 322 total files and 27 vulnerable files.
– JavaScript Dataset[4,5]: The JSVulnerability dataset is collected from the Node Security Platform and the Snyk Vulnerability Database. It consists of 12 125 total functions and 1496 vulnerable functions.
– All the metrics-based and text-mining-based datasets are downloaded and saved in CSV files. VPMs are trained on numerical features called software metrics, which are the characteristics of source code. Text-mining-based VPMs are trained on text features called tokens gathered from source code using various text mining techniques such as Bag-of-words (BOW), Term Frequency (TF), Term Frequency-Inverse Document Frequency (TF-IDF), etc.
– Each CSV file that contains metrics datasets shows the dependent and independent variables. The dependent variable determines whether the file/function is vulnerable or not, depending on the values of independent variables. PHP dataset has 13 independent

[3]http://seam.cs.umd.edu/webdata
[4]http://www.inf.u-szeged.hu/~ferenc/papers/JSVulnerabilityDataSet/
[5]https://sites.google.com/view/vulnerability-prediction-data/home

variables and one dependent variable. The JavaScript dataset contains 35 independent variables and one dependent variable.

– Table 2 describes the versions, no. of total files, no. of vulnerable files, no. of vulnerabilities, metrics features, and text features of the datasets used for experimentation
– Table 9 (refer to Appendix) gives the metrics description of the PHP and JavaScript datasets.

Table 2. Dataset descriptions

Dataset	Version	Total files /functions	Vulnerable files/functions	Vulnerabilities	Metrics	Text features
Drupal	6.0	202	62	97	13	3811
PHPMyAdmin	3.3.0	322	27	75	13	5232
Moodle	2.0.0	2924	24	51	13	18306
JSVulnerability	–	12 125	1496	–	35	12 942

4.2. Text mining

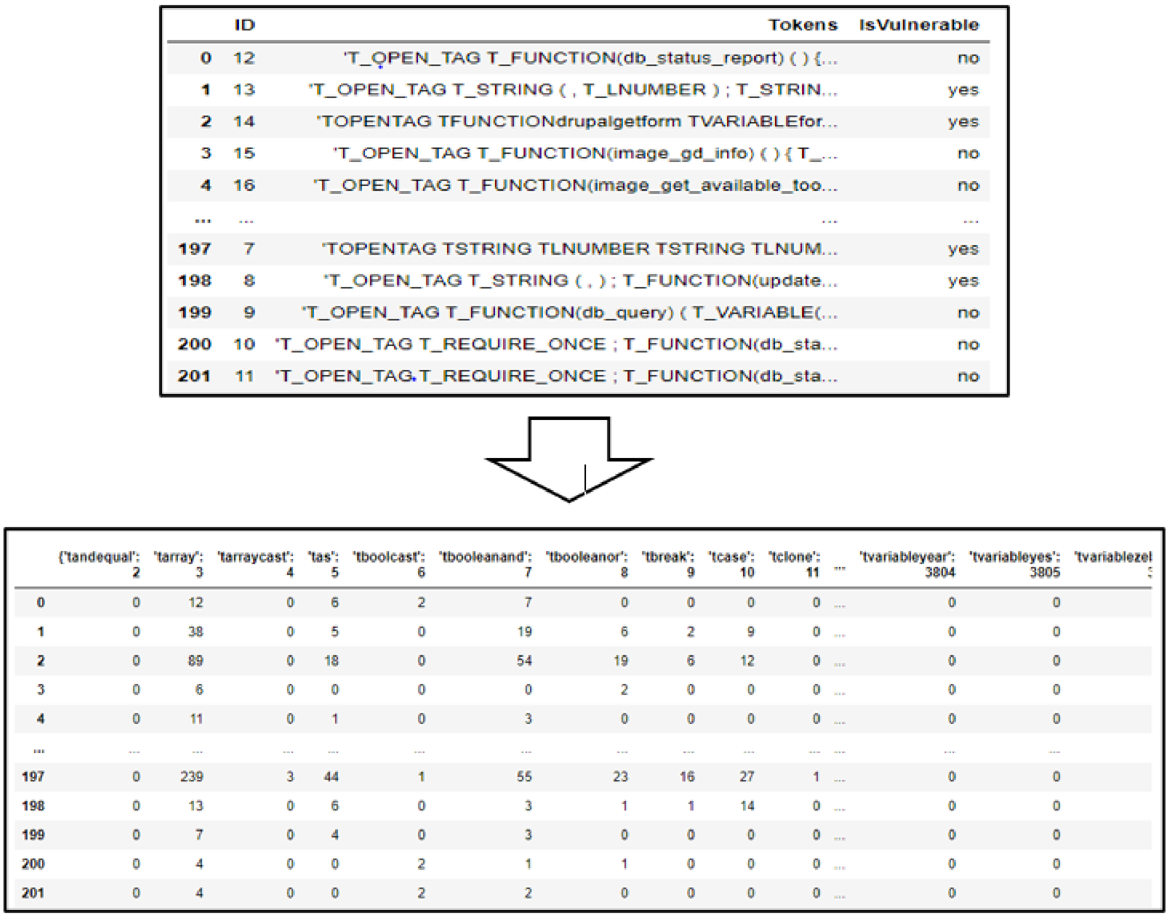
Text mining is the preprocessing of textual features and converting them into vector form which is the input for machine learning algorithms [57, 58]. There exist a lot of methods for performing text mining such as Bag-of-words (BOW), Term-Frequency (TF), Term-Frequency inverse document frequency (TF-IDF), sequence of tokens, etc.

Recent studies [8, 10, 20, 32, 34]] have mainly used BOW for preprocessing PHP dataset text features and for JavaScript datasets BOW and sequence of tokens have been used [12, 26, 27]. BOW calculates the number of occurrences of each word/token in the whole file/method. In text-mining-based VPMs, each token is considered a feature in the source code. To obtain text features from the source code of the Drupal dataset, the study first saved the textual dataset in Microsoft Access. There is a total of 202 files for the Drupal dataset and each file is labeled as vulnerable or not. To convert the source code into vector form, textual analysis is performed on the source code to remove redundant features, white spaces, punctuation marks, arithmetic, and logical operators. Then a dictionary/vocabulary is created which includes all the tokens associated with a key (see: Fig. 2). Each row in the MS-Access table represents a file which is compared against the dictionary to find out the occurrence of each token in the file. Finally, the CSV file is generated that contains the value of each token in the file. CSV file is further processed in MS-Excel to create each column heading indicating text-feature shown in Figure 3.

4.3. Experimental setup

In this paper, experiments are performed on four metrics and four text-features datasets. In addition to this, six metaheuristic feature selection approaches and eight machine learning algorithms are used. SMOTE technique is applied, keeping in view the imbalanced nature of the datasets. The experiments are performed in Jupyter notebook python by replicating the pseudocode mentioned in [24] with some additions mentioned in Algorithm below.

The new pseudocode presented in Table 3, includes eight datasets and four evaluation metrics (precision, recall, AUC, F_1-score). Every metaheuristic and machine learning algorithm uses this pseudocode to get the desired results. The number of generations and Population size is set to 100 and 10, respectively. The hyperparameters of machine learning,



	ID	Tokens	IsVulnerable
0	12	'T_OPEN_TAG T_FUNCTION(db_status_report) ( ) {...	no
1	13	'T_OPEN_TAG T_STRING (, T_LNUMBER) ; T_STRIN...	yes
2	14	'TOPENTAG TFUNCTIONdrupalgetform TVARIABLEfor...	yes
3	15	'T_OPEN_TAG T_FUNCTION(image_gd_info) ( ) { T_...	no
4	16	'T_OPEN_TAG T_FUNCTION(image_get_available_too...	no
...	...	...	...
197	7	'TOPENTAG TSTRING TLNUMBER TSTRING TLNUM...	yes
198	8	'T_OPEN_TAG T_STRING (,) ; T_FUNCTION(update...	yes
199	9	'T_OPEN_TAG T_FUNCTION(db_query) (T_VARIABLE(...	no
200	10	'T_OPEN_TAG T_REQUIRE_ONCE ; T_FUNCTION(db_sta...	no
201	11	'T_OPEN_TAG T_REQUIRE_ONCE ; T_FUNCTION(db_sta...	no

	{'tandequal': 2	'tarray': 3	'tarraycast': 4	'tas': 5	'tboolcast': 6	'tbooleanand': 7	'tbooleanor': 8	'tbreak': 9	'tcase': 10	'tclone': 11	...	'tvariableyear': 3804	'tvariableyes': 3805	'tvariablezel
0	0	12	0	6	2	7	0	0	0	0	...	0	0	
1	0	38	0	5	0	19	6	2	9	0	...	0	0	
2	0	89	0	18	0	54	19	6	12	0	...	0	0	
3	0	6	0	0	0	0	2	0	0	0	...	0	0	
4	0	11	0	1	0	3	0	0	0	0	...	0	0	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	
197	0	239	3	44	1	55	23	16	27	1	...	0	0	
198	0	13	0	6	0	3	1	1	14	0	...	0	0	
199	0	7	0	4	0	3	0	0	0	0	...	0	0	
200	0	4	0	0	2	1	1	0	0	0	...	0	0	
201	0	4	0	0	2	2	0	0	0	0	...	0	0	

Figure 2. Conversion of source code into vector form

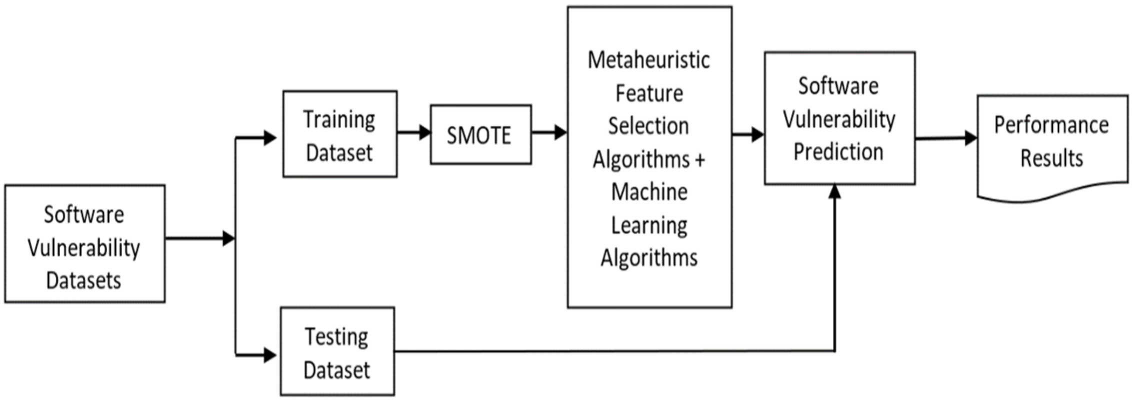

Software Vulnerability Datasets
Training Dataset
Testing Dataset
SMOTE
Metaheuristic Feature Selection Algorithms + Machine Learning Algorithms
Software Vulnerability Prediction
Performance Results

Figure 3. Working methodology

Table 3. Algorithm

```
Input: Dataset (D)
Output: Optimized cross-validated results (precision, recall, AUC, $F_1$-score)
1. Initialize the initial population of metaheuristic as: $\{a_1, a_2, a_3, \ldots, a_n\}$ where $a_i = [0, 1]$, 0 means
   feature is not selected, 1 means feature is selected and $n$ is the number of features in the dataset
2. Take the fitness function as $F_1$-score of ML on the subset of D.
3. Repeat step 1 and 2 until the desired number of iterations or we get maximum $F$-measure = 1.
```

Pseudo-code of Metaheuristics-ML

```
Input: Vulnerability Dataset = $D\{Drupal_m, Drupal_t, Moodle_m, Moodle_t, PHPMyAdmin_m, PHPMyAdmin_t,$
$JavaScript_m, JavaScript_t\}$
Output: Optimal values of precision, recall, AUC, $F_1$-score
1. Initialize the values: Number of dimension = independent features of dataset, Number of genera-
   tion = $n$, population size = $N$, Take initial candidate solution as $\{a_1, a_2, a_3, \ldots, a_n\}$ where $a_i = [0, 1]$
   n is the number of features in the dataset
2. For each iteration:
     – Selection features set $D'$ from $D$
     – Divide $D'$ into 80:20 ratio $\{D_tr, D_te\}$
     – Preprocess and standardize the dataset $D'$
     – Train a ML with $D_tr$
     – Evaluate the ML with 10-fold cross validation.
     – Return fitness = $F_1$-score
   After $n$ generation or $F_1$-score = 1
   Best $F_1$-score, precision, recall, AUC on 10-cross validation.
End
```

data balancing, and metaheuristic feature selection techniques are set to default. The main focus of this study is to compare the different combinations of ML and metaheuristic techniques hence the optimized hyperparameters are to be considered in the future. Figure 3 represents the working methodology of the VPMs. 80% of the dataset is split to train the machine learning classifier, and the classifier's performance is measured using 10-cross validation. The obtained F-measure serves as a fitness function for metaheuristic algorithms. The metaheuristic algorithm selects the best features based on the optimized AUC, precision, recall, and F-measures.

5. Results

The results are gathered after performing experiments and are represented in Tables A2–A9 of the Appendix. Tables A2–A5 show metrics-based results and Tables A6–A9 show text-mining-based results. Each table gives the evaluation metrics (AUC, Precision, Recall, F_1-score) values of each combination of machine learning algorithms and feature selection methods. Furthermore, there lies a column named "N_features" that depicts the count of features selected for every combination. In addition to this, for metrics-based datasets, the index of features selected is also mentioned but for text-features-based datasets, describing the index would be cumbersome. The bold values indicate the highest value of each performance metric among each machine learning algorithm and the yellow shaded + bold values indicates the highest value of each performance metric across all machine learning algorithm per dataset. Tables 4 and 5 represent the best-performing metaheuristic technique for each machine learning algorithm in each dataset.

Table 4. Best AUC values of metaheuristic feature selection for metrics-based VPMs in all machine learning algorithms

Dataset	Machine learning technique	Best performing metaheuristic technique	AUC
Drupal	RF	SSA	0.9643
	SVM	SSA	0.8928
	KNN	HHO	0.9658
	DT	SSA	0.9652
	AB	GWO	0.9656
	NB	GWO	0.8939
	LR	GA	0.9286
	MLP	WOA	0.8927
Moodle	RF	SSA	0.9573
	SVM	GWO	0.8659
	KNN	PSO	0.9616
	DT	SSA	0.9968
	AB	GWO	0.9745
	NB	GA	0.8031
	LR	WOA	0.8756
	MLP	GA	0.9439
PHPMyAdmin	RF	HHO	0.9661
	SVM	GA	0.8306
	KNN	GWO	0.9161
	DT	GWO	0.9833
	AB	SSA	0.9609
	NB	SSA	0.7638
	LR	SSA	0.8649
	MLP	GWO	0.9149
JavaScript	RF	GA	0.9705
	SVM	GA	0.8035
	KNN	SSA	0.9322
	DT	PSO	0.9605
	AB	PSO	0.8923
	NB	PSO	0.6929
	LR	GA	0.7362
	MLP	HHO	0.8745

5.1. Results for metrics-based VPMs

Table 4 describes the best-performing metaheuristic feature selection algorithm for each machine learning technique in the metrics-based VPMs:
- for Drupal, KNN-HHO has performed highest with AUC 0.9658,
- for Moodle, DT-SSA has performed highest with AUC 0.9968,
- for PHPMyAdmin, DT-GWO performed highest with AUC 0.9833,
- for JavaScript, RF-GA performed best with AUC 0.9705.

5.2. Results for text-features-based VPMs

Table 5 describes the best-performing metaheuristic feature selection algorithm for each machine learning technique in the text-features-based VPMs:
- for Drupal, MLP-GWO has performed highest with AUC 0.9986,
- for Moodle, DT-GWO has performed highest with AUC 0.9561,
- for PHPMyAdmin, AB-GWO performed highest with AUC 0.9879,

– for JavaScript, NB-HHO performed best with AUC 0.9998.

Table 5. Best AUC values of metaheuristic feature selection for text-feature-based VPMs in all machine learning algorithms

Dataset	Machine learning technique	Best performing metaheuristic technique	AUC
Drupal	RF	GWO	0.9666
	SVM	GA	0.9289
	KNN	GWO	0.9929
	DT	GA	0.9648
	AB	GA	0.9648
	NB	PSO	0.9289
	LR	GWO	0.9982
	MLP	GWO	0.9986
Moodle	RF	HHO	0.9469
	SVM	SSA	0.8031
	KNN	PSO	0.9421
	DT	GWO	0.9561
	AB	GWO	0.9315
	NB	GA	0.8631
	LR	HHO	0.9144
	MLP	GWO	0.9144
PHPMyAdmin	RF	GWO	0.9833
	SVM	GWO	0.9152
	KNN	GA	0.9833
	DT	GWO	0.9859
	AB	GWO	0.9879
	NB	SSA	0.8474
	LR	PSO	0.9666
	MLP	GWO	0.9878
JavaScript	RF	WOA	0.9995
	SVM	HHO	0.9788
	KNN	WOA	0.9896
	DT	WOA	0.9994
	AB	GWO	0.9995
	NB	HHO	0.9998
	LR	GWO	0.9912
	MLP	WOA	0.9995

The findings show that different machine learning algorithms have different best-performing metaheuristic feature selection techniques. The No-Free-Lunch (NFL) theorem states that no optimization or machine learning algorithm is good enough to solve all issues [59]. As a result, there is no guarantee that a single metaheuristic will uncover the best set of characteristics across all problem domains. Given these considerations, there is always the possibility of generating superior results with novel feature selection metaheuristics.

5.3. Statistical tests and results

Tables 6–9 show whether the feature selection methods have improved the performance of VPMs by comparing the values of each performance metric for both software metrics and text features-based datasets. The highest value of performance metrics is considered among different feature selection algorithms. Furthermore, the Wilcoxon signed rank statistical test [60] is applied to identify whether text-mining-based VPMs perform better than metrics-based VPMs. The authors have performed hundred iterations of machine learning algorithm on different (metrics and text mining) datasets without feature selection and with feature selection. Considering Tables 6–9, for instance, hundred performance values (X-Samples) of RF(metrics) is compared with X-Samples of RF(text) in the without feature selection case. Similarly, the ML+FS combination with highest performance values are selected and their iterative values (X-Samples) are compared using Wilcoxon signed rank test. The significant p-value is considered to be 0.05.

The hypothesis is as follows:
H_0: Text-mining-based VPMs are better than metrics-based VPMs.
If the p-value is less than 0.05 then accept H_0, otherwise reject H_0. Accepting the hypothesis indicates that text-mining-based VPMs are better than metrics-based VPMs and rejecting the hypothesis indicates that metrics-based VPMs are better than text-mining-based VPMs. Tables 6–9, highlight the cases where the p-value is less than 0.05, therefore null hypothesis is accepted in them indicating text-mining-based VPMs are better than metrics-based VPMs.

6. Discussion

Imbalanced datasets, hyperparameter settings, and dimensionality of the dataset have degraded the performance of VPMs. This study is performed to find whether the combination of multiple metaheuristic feature selection and machine learning algorithms increases the efficacy of VPMs. In addition to this, the performance of text-features-based and metrics-based VPMs are compared. Furthermore, the focus is on finding the best metaheuristic technique and if not stating the reason behind it, also which technique has performed satisfactorily for all the datasets. These are illustrated through the answers to the research questions mentioned below.

6.1. Illustration of research questions

RQ 1. Has all the metaheuristics feature selection and machine learning combinations improved the efficacy of VPMs?

The comparison of various machine learning methods based on the usage of feature selection methods for each dataset is shown in Tables 6–9. The findings have shown that the feature selection method has improved the efficacy for both metrics-based and text-features-based VPMs with maximum performance metrics (AUC, Precision, Recall, and F_1-score) values of 0.9833, 0.9962, 0.9974, 0.9962 and 0.9986, 0.9994, 0.9996, 0.9997, respectively.

Table 6 shows the results for the Drupal dataset.

– It has been observed that for metrics-based VPMs AUC, Precision, Recall, and F_1-score have improved by 15.9%, 34.92%, 25.58%, and 34.98%, respectively.
– For text-mining-based VPMs AUC, Precision, Recall, and F_1-score have improved by 13.06%, 34.6%, 25.69%, and 31.94%, respectively.

Table 7 shows the results for Moodle dataset.

– For metrics-based VPMs, the performance metrics AUC, Precision, Recall, and F_1-score have improved by 14.37%, 96.56 %, 42.47%, and 93.69%, respectively.

Table 6. Comparison of various machine learning methods based on the usage of feature selection methods for the Drupal dataset

Machine learning techniques	Without feature selection				With feature selection			
	AUC	Precision	Recall	F_1-score	AUC	Precision	Recall	F_1-score
RF(metrics)	**0.8122**	0.5993	0.6751	**0.6341**	0.9643	0.991	0.9912	0.9587
RF(text)	**0.8681**	0.6225	**0.7422**	**0.6771**	0.9666	0.9587	0.9289	0.9435
p-value	0.0014	0.0015	0.0051	0.0052	0.2541	–	–	–
SVM(metrics)	0.8006	0.5332	0.6316	0.5782	0.8928	0.9166	0.9925	0.9001
SVM(text)	0.8745	0.6397	0.6859	0.6619	0.9289	0.9145	0.9286	0.8848
p-value	0.0014	0.0001	0.0026	0.0001	0.0022	–	–	–
KNN(metrics)	0.7693	0.5327	**0.7099**	0.6086	**0.9658**	0.9941	0.9947	**0.9753**
KNN(text)	0.7732	0.6951	0.7215	0.7080	0.9929	**0.9912**	0.9899	0.9903
p-value	0.3321	0.0001	0.0018	0.0001	0.0009	–	–	0.0
DT(metrics)	0.6726	0.5196	0.5567	0.5091	0.9652	0.9898	0.9925	0.9582
DT(text)	0.6939	0.5301	0.5741	0.5196	0.9648	0.9333	0.9485	0.9608
p-value	0.0045	0.0042	0.0452	0.0412	–	–	–	0.0456
AB(metrics)	0.7747	0.5393	0.6661	0.5961	0.9656	0.9337	0.9957	0.9634
AB(text)	0.7915	**0.6482**	0.7011	0.6736	0.9648	0.9745	0.9486	0.9614
p-value	0.0014	0.0013	0.0036	0.0001	–	0.0035	–	–
NB(metrics)	0.7773	**0.6475**	0.4214	0.4952	0.8939	**0.9948**	0.8571	0.8888
NB(text)	0.8765	0.7088	0.7286	0.7185	0.9289	0.9231	0.8788	0.8888
p-value	0.0001	0.0036	0.0001	0.0001	0.0013	–	0.0012	–
LR(metrics)	0.6386	0.4941	0.5811	0.5341	0.9286	0.9915	**0.9974**	0.9194
LR(text)	0.7154	0.5715	0.6215	0.5954	0.9982	0.9911	**0.9988**	**0.9949**
p-value	0.0004	0.0001	0.0004	0.0051	0.0042	–	0.3156	0.0015
MLP(metrics)	0.7094	0.4274	0.5283	0.4725	0.8927	0.9090	0.9286	0.8965
MLP(text)	0.7615	0.5668	0.5507	0.5586	**0.9986**	0.9901	0.9928	0.9914
p-value	0.0026	0.0001	0.0365	0.0016	0.0001	0.0015	0.0015	0.0001

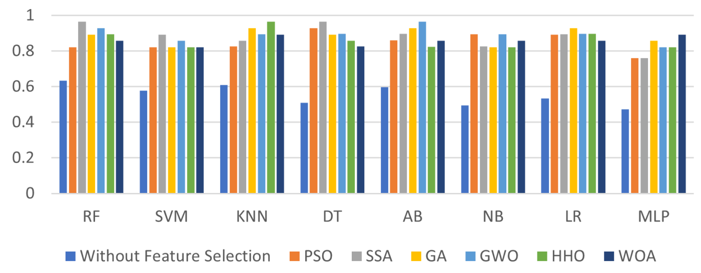


Figure 4. AUC Performance results for Drupal (metrics) dataset

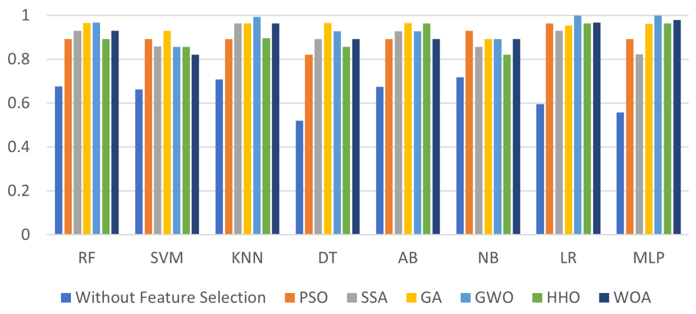


Figure 5. AUC Performance results for Drupal (text) dataset

Table 7. Comparison of various machine learning methods based on the usage of feature selection methods for the Moodle dataset

Machine learning techniques	Without feature selection				With feature selection			
	AUC	Precision	Recall	F_1-score	AUC	Precision	Recall	F_1-score
RF(metrics)	0.7453	0.0207	0.0732	0.0294	0.9573	0.9406	0.9966	0.9581
RF(text)	0.6502	0.0159	0.1903	0.0295	0.9469	0.9364	0.9794	**0.9475**
p-value	–	–	0.0001	0.4851	–	–	–	–
SVM(metrics)	0.7833	0.0218	0.4708	0.0417	0.8659	0.8977	0.8459	0.8611
SVM(text)	0.8245	0.0358	0.5891	0.0676	0.8031	**0.9941**	0.6198	0.7589
p-value	0.0052	0.0152	0.0001	0.0152	–	0.0001	–	–
KNN(metrics)	0.6983	0.0229	0.3801	0.0432	0.9616	0.9269	**0.9968**	0.9605
KNN(text)	0.7035	0.0343	0.4014	0.0632	0.9421	0.9126	0.9829	0.9394
p-value	0.0452	0.0452	0.0452	0.0452	–	–	–	–
DT(metrics)	0.5292	0.0147	0.0784	0.0289	0.9968	**0.9962**	0.9966	**0.9962**
DT(text)	0.5708	0.0178	0.2423	0.0331	**0.9561**	0.9302	0.9623	0.9443
p-value	0.0052	0.0452	0.0001	0.0452	–	–	–	–
AB(metrics)	0.7419	0.0171	0.2553	0.0343	**0.9745**	0.9823	0.9863	0.9742
AB(text)	0.7689	0.0382	0.3641	**0.0692**	0.9315	0.8937	0.9897	0.9346
p-value	0.0098	0.0121	0.0001	0.0121	–	–	0.2465	–
NB(metrics)	**0.8344**	**0.0342**	0.3882	**0.0628**	0.8031	0.8969	0.7329	0.7882
NB(text)	**0.8437**	**0.0487**	0.3961	0.0867	0.8631	0.7849	0.9966	0.8707
p-value	0.0452	0.0016	0.0451	0.0021	0.0052	–	0.0001	0.0001
LR(metrics)	0.6501	0.0209	**0.5734**	0.0402	0.8756	0.8292	0.9589	0.8837
LR(text)	0.7276	0.0313	**0.6166**	0.0596	0.9144	0.8601	**0.9978**	0.9204
p-value	0.0004	0.0452	0.0098	0.0452	0.0041	0.0012	0.0013	0.0056
MLP(metrics)	0.6253	0.0237	0.3269	0.0442	0.9439	0.9218	0.9863	0.9449
MLP(text)	0.7121	0.0323	0.4256	0.0611	0.9144	0.8965	0.9966	0.9186
p-value	0.0001	0.0098	0.0001	0.0098	–	–	0.0425	–

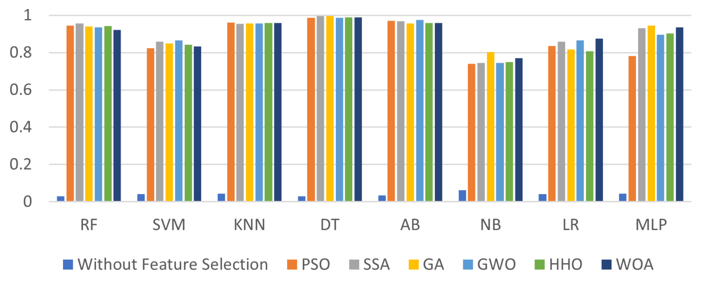


Figure 6. AUC Performance results for Moodle (metrics) dataset

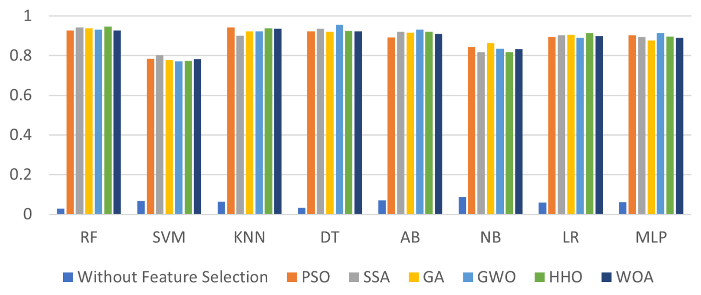


Figure 7. AUC Performance results for Moodle (text) dataset

Table 8. Comparison of various machine learning methods based on the usage of feature selection methods for the PHPMyAdmin dataset

Machine learning techniques	Without feature selection				With feature selection			
	AUC	Precision	Recall	F_1-score	AUC	Precision	Recall	F_1-score
RF(metrics)	**0.7536**	0.2377	0.3562	0.2852	0.9661	0.9655	0.9489	0.9658
RF(text)	**0.7651**	0.3936	0.4253	0.4088	0.9833	0.9677	0.9945	0.9791
p-value	0.0251	0.0001	0.0001	0.0001	0.0026	0.3512	0.0045	0.0026
SVM(metrics)	0.7312	0.1949	0.54	0.2864	0.8306	0.8276	0.8279	0.8277
SVM(text)	0.7917	0.2559	0.5523	0.3497	0.9152	0.9311	0.90	0.9152
p-value	0.0098	0.0001	0.0251	0.0001	0.0001	0.0001	0.0004	0.0001
KNN(metrics)	0.6705	0.1332	0.5208	0.1988	0.9161	0.875	0.9655	0.9181
KNN(text)	0.6735	0.1588	**0.5563**	0.2971	0.9833	0.9666	0.9778	0.9722
p-value	0.3512	0.0251	0.0452	0.0001	0.0014	0.0001	0.0026	0.0056
DT(metrics)	0.6726	**0.5196**	**0.5567**	**0.5375**	**0.9833**	**0.9777**	**0.9897**	**0.9831**
DT(text)	0.7416	0.6441	0.6602	0.6521	0.9859	0.9677	0.9789	0.9736
p-value	0.0001	0.0001	0.0001	0.0001	0.3516	–	–	–
AB(metrics)	0.6092	0.1398	0.3027	0.1913	0.9609	0.9666	0.9782	0.9665
AB(text)	0.6916	0.2854	0.3798	0.3259	0.9879	0.9667	0.9789	0.9727
p-value	0.0098	0.0001	0.0098	0.0001	0.0452	0.5462	0.5462	0.0452
NB(metrics)	0.7009	0.2165	0.3268	0.2605	0.7638	0.8184	0.8666	0.7762
NB(text)	0.7284	0.3514	0.4432	0.3919	0.8474	0.7631	0.8355	0.7976
p-value	0.0125	0.0001	0.0001	0.0001	0.0004	–	–	0.0452
LR(metrics)	0.6379	0.1661	0.2535	0.2007	0.8649	0.8846	0.90	0.8709
LR(text)	0.7782	0.2626	0.3101	0.2844	0.9666	**0.9917**	0.9333	0.9616
p-value	0.0001	0.0002	0.0041	0.0041	0.0001	0.0001	0.0452	0.0004
MLP(metrics)	0.6678	0.1532	0.4383	0.2271	0.9149	0.9285	0.90	0.9122
MLP(text)	0.6878	0.2733	0.5581	0.3669	**0.9878**	0.9756	0.9721	0.9693
p-value	0.0042	0.0452	0.0001	0.0001	0.0056	0.0056	0.0056	0.0065

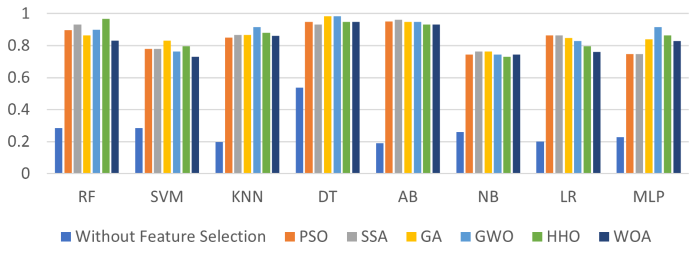


Figure 8. AUC Performance results for PHPMyAdmin (metrics) dataset

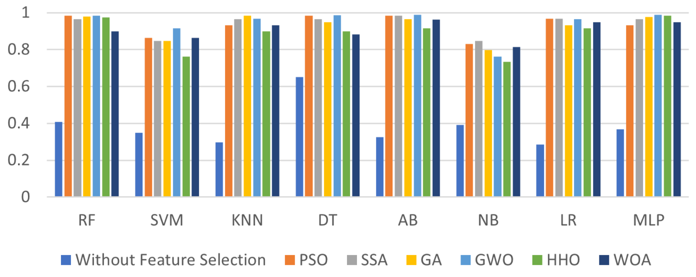


Figure 9. AUC Performance results for PHPMyAdmin (text) dataset

Table 9. Comparison of various machine learning methods based on the usage of feature selection methods for the JavaScript dataset

Machine learning techniques	Without feature selection				With feature selection			
	AUC	Precision	Recall	F_1-score	AUC	Precision	Recall	F_1-score
RF(metrics)	**0.9437**	**0.7261**	0.7655	**0.7458**	**0.9705**	**0.9798**	0.9689	**0.9701**
RF(text)	**0.9591**	0.8792	0.8272	0.8523	**0.9995**	**0.9994**	0.9991	0.9995
p-value	0.0452	0.0001	0.0015	0.0001	0.0452	0.0452	0.0452	0.0452
SVM(metrics)	0.6024	0.5255	0.2404	0.3297	0.8035	0.8721	0.9322	0.7872
SVM(text)	0.7878	0.5335	0.2871	0.3733	0.9788	0.9593	0.9991	0.9793
p-value	0.0001	0.0551	0.0452	0.0245	0.0001	0.0001	0.0245	0.0001
KNN(metrics)	0.8742	0.5054	0.7406	0.5995	0.9322	0.9105	**0.9671**	0.9345
KNN(text)	0.9248	0.4897	0.8386	0.6201	0.9896	0.9869	0.9944	0.9897
p-value	0.0045	–	0.0004	0.0056	0.0023	0.0016	0.0056	0.0045
DT(metrics)	0.8611	0.6146	**0.7713**	0.6868	0.9605	0.9595	0.9633	0.9606
DT(text)	0.9598	0.8894	0.8474	0.8678	0.9994	0.9988	0.9978	0.9972
p-value	0.0001	0.0001	0.0035	0.0001	0.0045	0.0045	0.0045	0.0045
AB(metrics)	0.8777	0.4565	0.7149	0.5572	0.8923	0.9273	0.8561	0.8877
AB(text)	0.9003	0.5664	0.7689	0.6552	0.9995	0.9981	0.9958	0.9964
p-value	0.0452	0.0004	0.0452	0.0001	0.0001	0.0023	0.0001	0.0001
NB(metrics)	0.7772	0.6475	0.4214	0.4952	0.6929	0.6414	0.8749	0.7401
NB(text)	0.9435	**0.9526**	**0.8985**	**0.9194**	0.9998	0.9994	0.9992	0.9996
p-value	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001
LR(metrics)	0.6772	0.2803	0.5371	0.3646	0.7362	0.7307	0.8118	0.7462
LR(text)	0.7231	0.3002	0.5518	0.3888	0.9912	0.9879	0.9884	0.9881
p-value	0.0343	0.0452	0.0452	0.0452	0.0001	0.0001	0.0001	0.0001
MLP(metrics)	0.7913	0.3856	0.7449	0.5051	0.8745	0.8916	0.8758	0.8526
MLP(text)	0.8124	0.4152	0.7664	0.5386	0.9995	0.9995	**0.9996**	**0.9997**
p-value	0.0452	0.0452	0.0452	0.0452	0.0001	0.0001	0.0001	0.0001

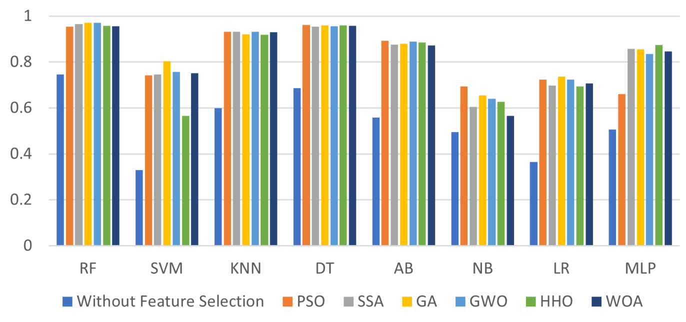


Figure 10. AUC Performance results for JavaScript (metrics) dataset

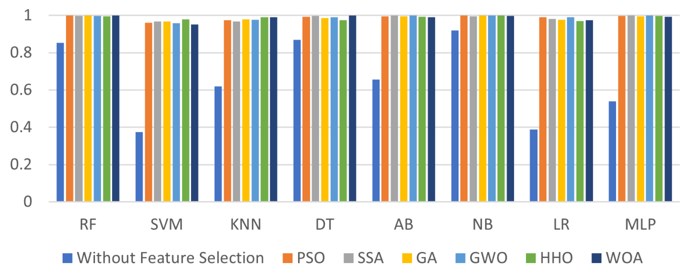


Figure 11. AUC Performance results for JavaScript(text) dataset

– For text-mining-based VPMs AUC, Precision, Recall, and F_1-score have improved by 11.75%, 42.47%, 38.21%, and 92.69%, respectively.
  Table 8 shows the results for the PHPMyAdmin dataset.
– For metrics-based VPMs, the performance metrics AUC, Precision, Recall, and F_1-score have improved by 23.3%, 46.85%, 43.75%, and 66.5%, respectively.
– For text-mining-based VPMs, the performance metrics AUC, Precision, Recall, and F_1-score have improved by 25.58%, 82.52%, 44.06%, and 79.37%.
  Table 9 shows the results for the JavaScript dataset,
– For metrics-based VPMs, the performance metrics AUC, Precision, Recall, and F_1-score have improved by 2.7%, 25.89%, 21.06%, and 23.12%, respectively.
– For text-mining-based VPMs, the performance metrics AUC, Precision, Recall, and F_1-score have improved by 4.04%, 4.68%, 10.11%, and 8.03%, respectively.

Figures 4–11 clearly show that after applying feature selection there is an improvement in the productivity of VPMs.

RQ 2. Which one statistically performed better, metrics-based or text-mining-based VPMs in the context of feature selection?
Tables 6–9 describe the statistical difference between metrics-based and text-mining-based VPMs after applying Wilcoxon signed rank test. For Drupal, without feature selection in all the cases text-mining-based VPMs have performed statistically better than metrics-based VPMs. After applying feature selection, in 13 out of 32 cases text-mining performed statistically better. For Moodle, without feature selection in 29 cases and with feature selection in 9 out of 32 cases text-mining-based VPMs performed better than metrics-based VPMs. For PHPMyAdmin, without feature selection, in 31 cases and with feature selection in 24 out of cases text-mining-based VPMs performed better than metrics-based VPMs. For JavaScript, without feature selection in 30 cases and with feature selection in all the cases text-mining-based VPMs performed better than metrics-based VPMs. Therefore, it is evident that text-mining-based VPMs statistically performed better than metrics-based VPMs in 60.9% of the cases in the context of feature selection.

RQ 3. Which metaheuristic feature selection algorithm has performed the best?
Different machine learning algorithms have different feature-selection methods that are performing best for each dataset. The No-Free-Lunch (NFL) theorem states that no optimization or machine learning algorithm is good enough to solve all issues [59]. As a result, there is no guarantee that a single metaheuristic will uncover the best set of characteristics across all problem domains. Given these considerations, there is always the possibility of generating superior results with novel feature selection metaheuristics. AUC performance metric is considered for describing the best-performing metaheuristic feature selection algorithm depicted in Figures 4–11. Figures 4, 6, 8, and 10 show results about metrics-based and Figures 5, 7, 9, and 11 text-features-based datasets.

For metrics-based VPMs, in the case of Drupal SSA, Moodle SSA, GWO, GA, PHPMyAdmin SSA, GWO, JavaScript GA, PSO has performed maximum for all machine learning algorithms (refer to Table 4). For text-features-based VPMs, in the case of Drupal GWO, Moodle GWO, PHPMyAdmin GWO, and JavaScript WOA have performed maximum times (refer to Table 5). Overall, it can be noticed that GWO has performed satisfactorily for all the datasets.

6.2. Comparison with benchmark studies

Figures 12 and 13 show the comparison of the proposed work with the existing benchmark studies for the PHP and JavaScript datasets, respectively. The paper has considered the F_1-score metric as the comparison criterion since it is preferably used in previous research studies. Figure 12 shows that Drupal and Moodle have the highest F_1-score for the proposed work whereas F_1-score for PHPMyAdmin is slightly less than Sahin et al. [25]. Sahin et al. [25] has not applied any data balancing technique which may produce biased results. Figure 13 shows that the proposed work's F_1-score has outperformed the benchmark studies.

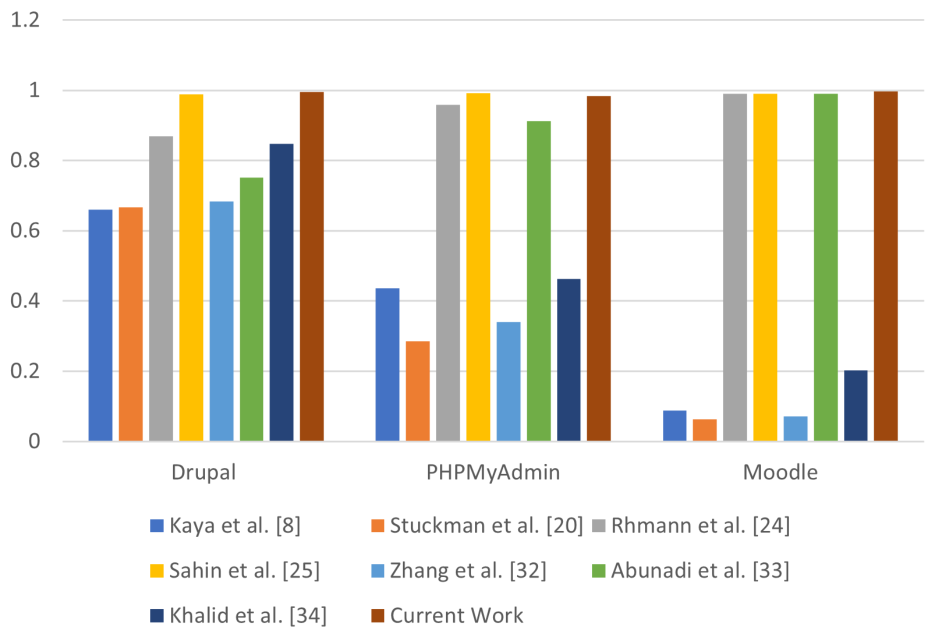


Figure 12. F_1-score performance comparison with existing studies for PHP dataset

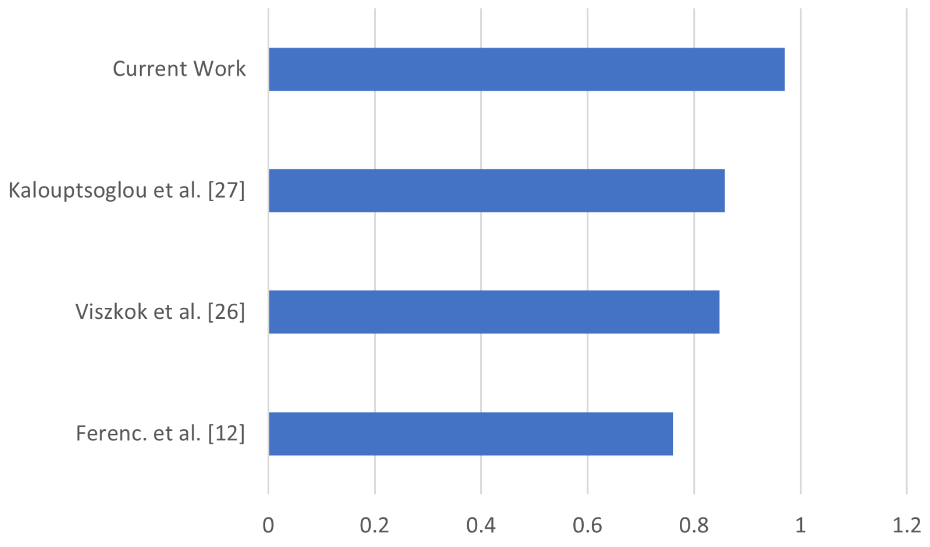


Figure 13. F_1-score performance comparison with existing studies for JavaScript dataset

6.3. Summary

Our study has unveiled the role of metaheuristic feature selection algorithms on the efficacy of VPMs. It has considered a wide variety of datasets, features, and machine-learning techniques and achieved high-performing VPMs with a maximum AUC of 0.9968 for metrics--based and 0.9998 for text-mining-based VPMs. Researchers can further use optimized hyperparameters and consider time and cost complexity issues for VPMs in the future.

7. Threats to validity

The current study covers the following threats to validity:
- Internal Validity: The selection of eight machine learning methods for the current study is based on previous research studies. Under-sampling techniques are not used due to loss of information. The paper has restricted its work to SMOTE data balancing techniques. Hybrid data balancing techniques are left for future scope.
- Construct Validity: This paper uses PHP-based open-source and JavaScript projects. The current study has included metrics-based datasets and text-mining-based datasets. The combination of the metrics and text-mining features is left for future scope. In addition to this, only default hyperparameters were considered and Bag of words was used for text mining. The authors are aware of the fact that optimized hyperparameters increase the performance of machine learning models [17] but are unaware about how would the combination of metaheuristic feature selection and optimized hyperparameters would perform thereby keeping it as a future aspect to be considered.
- Conclusion Validity: This paper uses AUC, precision, recall, and F_1-score for evaluating the performance of the prediction models. We have not used accuracy as they give biased results for imbalanced datasets. Also, considering the readability and clarity of the paper MCC and G-mean metrics are left for future scope.
- External Validity: The paper tries to perform the methodology on the PHP dataset and validate it be executing experiments on JavaScript. In addition, the granularity levels are also different, i.e., file for PHP and function for JavaScript. The work is confined to only two programming languages. In the future, more programming languages can be used and the results may vary.

8. Conclusions and future scope

The need for efficient VPMs has always been crucial and to achieve that, previous studies have done immense work, from balancing classes and appropriate hyperparameters selection to reducing the dimensionality through feature synthesis and feature selection methods. The present paper has worked on feature selection using nature-inspired and swarm intelligence-based algorithms. It has performed the empirical analysis on various combinations of eight machine learning techniques and six metaheuristic feature selection approaches on PHP and JavaScript datasets. The experiments are evaluated using six performance metrics, keeping the imbalanced nature of the datasets in mind. It has been used on metrics-based and text-token-based datasets. Further, the statistical comparison of metrics-based and text-mining-based VPMs is implemented by Wilcoxon signed rank test in the context of feature selection. The comparative analysis concludes that:

Metaheuristics feature selection methods improve the performance of VPMs (metrics and text-mining) in the range of 2.7%–25.58% in terms of AUC, 4.68%–96.56% in terms of precision, 10.11%–44.06% in terms of recall, and 8.03%–93.69% in terms of F_1-score.

The Wilcoxon signed rank test showed that overall 200 p-values are found significant out of 256 among all performance metrics. Therefore, overall it can be said that text-features-based VPMs are significantly better than metrics-based VPMs in 78.12% of the cases. But in the context of feature selection, 78 out of 128 cases performed significantly showing that text-features-based VPMs performed better than metrics-based VPMs for 60.9% of the instances when feature selection is applied.

The highest AUC values were obtained in metrics-based VPMs by KNN-HHO for Drupal, DT-SSA for Moodle, DT-GWO for PHPMyAdmin, and RF-GA for JavaScript. For text-mining based VPMs, the highest AUC values were obtained by MLP-GWO in Drupal, DT-GWO in Moodle, AB-GWO in PHPMyAdmin, and NB-HHO in JavaScript. The paper compares AUC values to find out the maximum-performing feature selection techniques for all machine learning algorithms. For metrics-based datasets, Drupal SSA; Moodle SSA, GWO, GA; PHPMyAdmin GWO; and JavaScript GA have performed the maximum times for all machine learning algorithms. For text-mining datasets, Drupal GWO, Moodle GWO, PHPMyAdmin GWO, and JavaScript WOA have performed maximum times. Overall, GWO has performed the maximum number of times.
Furthermore, the present paper has outperformed the benchmark studies in terms of F_1-score.

In the Future, more deep learning methods like LSTM, GRU, etc., can be applied. Moreover, we have involved only default hyperparameters and in the future, analysis can be done using optimized hyperparameters. More metaheuristic algorithms can be applied and a combination of both metrics and text tokens can be used. Further, other programming languages can also be applied.

CRediT authorship contribution statement

Dr. Deepali Bassi and Dr. Hardeep Singh contributed to the study's conception and design. Material preparation, data collection, and analysis were performed by Deepali Bassi. Hardeep Singh read and approved the final manuscript.

Declaration of competing interest

The authors declare that they have no conflict of interest. We have no relevant financial or non-financial interests to disclose. We have no competing interests to declare that are relevant to the content of this article.

Data availability

The Code Snippet is provided on the link https://drive.google.com/file/d/14qoY98ZPi-WalupRUquMALR_HHChuSaPB/view?usp=sharing

Funding

No funding was received to assist with the preparation of this manuscript.

References

[1] S. Matteson, "Report: Software failure caused $1.7 trillion in financial losses in 2017," *TechRepublic*, 2018. [Online]. https://www.techrepublic.com/article/report-software-failure-caused-1-7-trillion-in-financial-losses-in-2017

[2] H. Hanif, M.H.N.M. Nasir, M.F. Ab Razak, A. Firdaus, and N.B. Anuar, "The rise of software vulnerability: Taxonomy of software vulnerabilities detection and machine learning approaches," *Journal of Network and Computer Applications*, Vol. 179, 2021, p. 103009.

[3] D.R. Kuhn, M.S. Raunak, and R. Kacker, "An analysis of vulnerability trends, 2008–2016," in *International Conference on Software Quality, Reliability and Security Companion (QRS-C)*. IEEE, 2017, pp. 587–588.

[4] S. Frei, "The known unknowns: Empirical analysis of publicly unknown security vulnerabilities," *NSS Labs*, 2013.

[5] T. Zimmermann, N. Nagappan, and L. Williams, "Searching for a needle in a haystack: Predicting security vulnerabilities for Windows Vista," in *Third International Conference on Software Testing, Verification and Validation*. IEEE, 2010, pp. 421–428.

[6] B. Arkin, S. Stender, and G. McGraw, "Software penetration testing," *IEEE Security and Privacy*, Vol. 3, No. 1, 2005, pp. 84–87.

[7] S.M. Ghaffarian and H.R. Shahriari, "Software vulnerability analysis and discovery using machine-learning and data-mining techniques: A survey," *ACM Computing Surveys (CSUR)*, Vol. 50, No. 4, 2017, pp. 1–36.

[8] A. Kaya, A.S. Keceli, C. Catal, and B. Tekinerdogan, "The impact of feature types, classifiers, and data balancing techniques on software vulnerability prediction models," *Journal of Software: Evolution and Process*, Vol. 31, No. 9, 2019, p. e2164.

[9] I. Chowdhury and M. Zulkernine, "Using complexity, coupling, and cohesion metrics as early indicators of vulnerabilities," *Journal of Systems Architecture*, Vol. 57, No. 3, 2011, pp. 294–313.

[10] J. Walden, J. Stuckman, and R. Scandariato, "Predicting vulnerable components: Software metrics vs text mining," in *25th International Symposium on Software Reliability Engineering*. IEEE, 2014, pp. 23–33.

[11] K. Borowska and J. Stepaniuk, "Imbalanced data classification: A novel re-sampling approach combining versatile improved smote and rough sets," in *Computer Information Systems and Industrial Management: 15th IFIP TC8 International Conference*. Springer, 2016, pp. 31–42.

[12] R. Ferenc, P. Hegedűs, P. Gyimesi, G. Antal, D. Bán et al., "Challenging machine learning algorithms in predicting vulnerable JavaScript functions," in *IEEE/ACM 7th International Workshop on Realizing Artificial Intelligence Synergies in Software Engineering (RAISE)*. IEEE, 2019, pp. 8–14.

[13] P.K. Kudjo, S.B. Aformaley, S. Mensah, and J. Chen, "The significant effect of parameter tuning on software vulnerability prediction models," in *19th International Conference on Software Quality, Reliability and Security Companion (QRS-C)*. IEEE, 2019, pp. 526–527.

[14] E. Sara, C. Laila, and I. Ali, "The impact of smote and grid search on maintainability prediction models," in *IEEE/ACS 16th International Conference on Computer Systems and Applications (AICCSA)*. IEEE, 2019, pp. 1–8.

[15] H. Osman, M. Ghafari, and O. Nierstrasz, "Hyperparameter optimization to improve bug prediction accuracy," in *Workshop on Machine Learning Techniques for Software Quality Evaluation (MaLTeSQuE)*. IEEE, 2017, pp. 33–38.
[16] R. Shu, T. Xia, L. Williams, and T. Menzies, "Better security bug report classification via hyperparameter optimization," *arXiv preprint arXiv:1905.06872*, 2019.
[17] D. Bassi and H. Singh, "Optimizing hyperparameters for improvement in software vulnerability prediction models," in *Advances in Distributed Computing and Machine Learning*. Springer, 2022, pp. 533–544.
[18] H. Wang, Y. Zhang, J. Zhang, T. Li, and L. Peng, "A factor graph model for unsupervised feature selection," *Information Sciences*, Vol. 480, 2019, pp. 144–159.
[19] X. Tang, Y. Dai, and Y. Xiang, "Feature selection based on feature interactions with application to text categorization," *Expert Systems with Applications*, Vol. 120, 2019, pp. 207–216.
[20] J. Stuckman, J. Walden, and R. Scandariato, "The effect of dimensionality reduction on software vulnerability prediction models," *IEEE Transactions on Reliability*, Vol. 66, No. 1, 2016, pp. 17–37.
[21] X. Chen, Z. Yuan, Z. Cui, D. Zhang, and X. Ju, "Empirical studies on the impact of filter-based ranking feature selection on security vulnerability prediction," *IET Software*, Vol. 15, No. 1, 2021, pp. 75–89.
[22] M. Rostami, K. Berahmand, E. Nasiri, and S. Forouzandeh, "Review of swarm intelligence-based feature selection methods," *Engineering Applications of Artificial Intelligence*, Vol. 100, 2021, p. 104210.
[23] D. Bassi and H. Singh, "The effect of dual hyperparameter optimization on software vulnerability prediction models," *e-Informatica Software Engineering Journal*, Vol. 17, No. 1, 2023, p. 230102.
[24] W. Rhmann, "Software vulnerability prediction using grey wolf-optimized random forest on the unbalanced data sets," *International Journal of Applied Metaheuristic Computing (IJAMC)*, Vol. 13, No. 1, 2022, pp. 1–15.
[25] C.B. Şahin, Ö.B. Dinler, and L. Abualigah, "Prediction of software vulnerability based deep symbiotic genetic algorithms: Phenotyping of dominant-features," *Applied Intelligence*, Vol. 51, 2021, pp. 8271–8287.
[26] T. Viszkok, P. Hegedűs, and R. Ferenc, "Improving vulnerability prediction of JavaScript functions using process metrics," *arXiv preprint arXiv:2105.07527*, 2021.
[27] I. Kalouptsoglou, M. Siavvas, D. Kehagias, A. Chatzigeorgiou, and A. Ampatzoglou, "Examining the capacity of text mining and software metrics in vulnerability prediction," *Entropy*, Vol. 24, No. 5, 2022, p. 651.
[28] S. Wang and X. Yao, "Using class imbalance learning for software defect prediction," *IEEE Transactions on Reliability*, Vol. 62, No. 2, 2013, pp. 434–443.
[29] Y. Shin, A. Meneely, L. Williams, and J.A. Osborne, "Evaluating complexity, code churn, and developer activity metrics as indicators of software vulnerabilities," *IEEE Transactions on Software Engineering*, Vol. 37, No. 6, 2010, pp. 772–787.
[30] Y. Shin and L. Williams, "Can traditional fault prediction models be used for vulnerability prediction?" *Empirical Software Engineering*, Vol. 18, 2013, pp. 25–59.
[31] R. Lagerström, C. Baldwin, A. MacCormack, D. Sturtevant, and L. Doolan, "Exploring the relationship between architecture coupling and software vulnerabilities," in *Engineering Secure Software and Systems*. Springer, 2017, pp. 53–69.
[32] Y. Zhang, D. Lo, X. Xia, B. Xu, J. Sun et al., "Combining software metrics and text features for vulnerable file prediction," in *20th International Conference on Engineering of Complex Computer Systems (ICECCS)*. IEEE, 2015, pp. 40–49.
[33] I. Abunadi and M. Alenezi, "An empirical investigation of security vulnerabilities within web applications," *J. Univers. Comput. Sci.*, Vol. 22, No. 4, 2016, pp. 537–551.
[34] M.N. Khalid, H. Farooq, M. Iqbal, M.T. Alam, and K. Rasheed, "Predicting web vulnerabilities in web applications based on machine learning," in *Intelligent Technologies and Applications: First International Conference, INTAP*. Springer, 2019, pp. 473–484.

[35] C. Catal, A. Akbulut, E. Ekenoglu, and M. Alemdaroglu, “Development of a software vulnerability prediction web service based on artificial neural networks,” in *Trends and Applications in Knowledge Discovery and Data Mining*. Springer, 2017, pp. 59–67.

[36] Z. Li and Y. Shao, “A survey of feature selection for vulnerability prediction using feature-based machine learning,” in *Proceedings of the 11th International Conference on Machine Learning and Computing*, 2019, pp. 36–42.

[37] T. Sonnekalb, T.S. Heinze, and P. Mäder, “Deep security analysis of program code: A systematic literature review,” *Empirical Software Engineering*, Vol. 27, No. 1, 2022, p. 2.

[38] Z. Li, D. Zou, S. Xu, X. Ou, H. Jin et al., “Vuldeepecker: A deep learning-based system for vulnerability detection,” *arXiv preprint arXiv:1801.01681*, 2018.

[39] Y. Zhou, S. Liu, J. Siow, X. Du, and Y. Liu, “Devign: Effective vulnerability identification by learning comprehensive program semantics via graph neural networks,” *Advances in neural information processing systems*, Vol. 32, 2019.

[40] Z. Li, D. Zou, S. Xu, Z. Chen, Y. Zhu et al., “Vuldeelocator: A deep learning-based fine-grained vulnerability detector,” *IEEE Transactions on Dependable and Secure Computing*, Vol. 19, No. 4, 2021, pp. 2821–2837.

[41] I. Kalouptsoglou, M. Siavvas, D. Tsoukalas, and D. Kehagias, “Cross-project vulnerability prediction based on software metrics and deep learning,” in *Computational Science and Its Applications – ICCSA*. Springer, 2020, pp. 877–893.

[42] H.H. Patel and P. Prajapati, “Study and analysis of decision tree based classification algorithms,” *International Journal of Computer Sciences and Engineering*, Vol. 6, No. 10, 2018, pp. 74–78.

[43] Z. Jin, J. Shang, Q. Zhu, C. Ling, W. Xie et al., “RFRSF: Employee turnover prediction based on random forests and survival analysis,” in *Web Information Systems Engineering – WISE*. Springer, 2020, pp. 503–515.

[44] M. Martinez-Arroyo and L.E. Sucar, “Learning an optimal naive Bayes classifier,” in *18th International Conference on Pattern Recognition (ICPR’06)*, Vol. 3. IEEE, 2006, pp. 1236–1239.

[45] R.E. Schapire, “The boosting approach to machine learning: An overview,” *Nonlinear estimation and classification*, 2003, pp. 149–171.

[46] C.C. Chang and C.J. Lin, “LIBSVM: A library for support vector machines,” Vol. 2, No. 3, 2011. [Online]. https://doi.org/10.1145/1961189.1961199

[47] J.M. Keller, M.R. Gray, and J.A. Givens, “A fuzzy k-nearest neighbor algorithm,” *IEEE Transactions on Systems, Man, and Cybernetics*, No. 4, 1985, pp. 580–585.

[48] H.F. Yu, F.L. Huang, and C.J. Lin, “Dual coordinate descent methods for logistic regression and maximum entropy models,” *Machine Learning*, Vol. 85, 2011, pp. 41–75.

[49] M.C. Popescu, V.E. Balas, L. Perescu-Popescu, and N. Mastorakis, “Multilayer perceptron and neural networks,” *WSEAS Transactions on Circuits and Systems*, Vol. 8, No. 7, 2009, pp. 579–588.

[50] N.V. Chawla, K.W. Bowyer, L.O. Hall, and W.P. Kegelmeyer, “SMOTE: Synthetic minority over-sampling technique,” *Journal of artificial intelligence research*, Vol. 16, 2002, pp. 321–357.

[51] G. Douzas, F. Bacao, and F. Last, “Improving imbalanced learning through a heuristic oversampling method based on k-means and SMOTE,” *Information Sciences*, Vol. 465, 2018, pp. 1–20.

[52] A. Zeb, F. Din, M. Fayaz, G. Mehmood, K.Z. Zamli et al., “A systematic literature review on robust swarm intelligence algorithms in search-based software engineering,” *Complexity*, Vol. 2023, 2023.

[53] S. Kassaymeh, S. Abdullah, M.A. Al-Betar, and M. Alweshah, “Salp swarm optimizer for modeling the software fault prediction problem,” *Journal of King Saud University – Computer and Information Sciences*, Vol. 34, No. 6, 2022, pp. 3365–3378.

[54] F.S. Gharehchopogh and H. Gholizadeh, “A comprehensive survey: Whale optimization algorithm and its applications,” *Swarm and Evolutionary Computation*, Vol. 48, 2019, pp. 1–24.

[55] S. Mirjalili, A.H. Gandomi, S.Z. Mirjalili, S. Saremi, H. Faris et al., “Salp swarm algorithm: A bio-inspired optimizer for engineering design problems,” *Advances in engineering software*, Vol. 114, 2017, pp. 163–191.

[56] J. Huang and C.X. Ling, "Using AUC and accuracy in evaluating learning algorithms," *IEEE Transactions on knowledge and Data Engineering*, Vol. 17, No. 3, 2005, pp. 299–310.

[57] K.Z. Sultana, B.J. Williams, and A. Bosu, "A comparison of nano-patterns vs. software metrics in vulnerability prediction," in *25th Asia-Pacific Software Engineering Conference (APSEC)*. IEEE, 2018, pp. 355–364.

[58] D. Wijayasekara, M. Manic, and M. McQueen, "Vulnerability identification and classification via text mining bug databases," in *IECON 40th Annual Conference of the IEEE Industrial Electronics Society*. IEEE, 2014, pp. 3612–3618.

[59] D.H. Wolpert, "The supervised learning no-free-lunch theorems," *Soft computing and industry: Recent applications*, 2002, pp. 25–42.

[60] D. Tomar and S. Agarwal, "Prediction of defective software modules using class imbalance learning," *Applied Computational Intelligence and Soft Computing*, Vol. 2016, 2016, pp. 6–6.

Appendix A

Table A1. Static source code metrics

Dataset	Metrics	Description
PHP Dataset	nonecholoc	Non-HTML lines of code
	loc	Total lines of code in a PHP file
	nmethods	No. of functions in a file
	ccomdeep, ccom	Cyclomatic complexity
	nest	Maximum depth of nested loops
	hvol	Halstead's volume
	nIncomingCalls	Fan-in
	nIncomingCallsUniq	Internal functions Called
	nOutgoingInternCalls	Fan-out
	nOutgoingExternFlsCalled	Total external calls
	nOutgoingExternFlsCalledUniq	External methods called
	nOutgoingExternCalls	External calls to methods
JavaScript Dataset	CC	Clone Coverage
	CCL	Clone Classes
	CCO	Clone Complexity
	CI	Clone Instances
	CLC	Clone Line Coverage
	LDC	Lines of Duplicated Code
	McCC, CCYL	Cyclomatic Complexity
	NL	Nesting Level
	NLE	Nesting Level without else-if
	CD, TCD	Comment Density
	CLOC, TCLOC	Comment Lines of Code
	DLOC	Documentation Lines of Code
	LLOC, TLLOC	Logical Lines of Code
	LOC, TLOC	Lines of Code
	NOS, TNOS	Number of Statements
	NUMPAR, PARAMS	Number of Parameters
	HOR_D	No. of Distinct Halstead Operators
	HOR_T	No. of Total Halstead Operators
	HON_D	No. of Distinct Halstead Operands
	HON_T	No. of Total Halstead Operands
	HLEN	Halstead Length
	HVOC	Halstead Vocabulary Size
	HDIFF	Halstead Difficulty
	HVOL	Halstead Volume
	HEFF	Halstead Effort
	HBUGS	Halstead Bugs
	HTIME	Halstead Time
	CYCL DENS	Cyclomatic Density

Table A2. Performance results of metrics-based Drupal dataset

Machine Learning Algorithms	Feature Selection Techniques	AUC	Precision	Recall	F_1-score	N_Features
RF	PSO	0.8214	0.9090	0.7143	0.80	5 [0, 2, 3, 5, 12]
	SSA	**0.9643**	**0.991**	0.9285	**0.9587**	5 [1, 4, 6, 10, 11]
	GA	0.8928	0.8666	0.9286	0.8966	3 [3, 5, 6]
	GWO	0.9286	0.875	**0.9912**	0.9295	3 [2, 4, 9]
	HHO	0.8929	0.8235	0.9911	0.8995	6 [2, 3, 7, 9, 11]
	WOA	0.8571	0.7777	0.9910	0.8714	5 [4, 6, 8, 10, 11]
SVM	PSO	0.8215	0.7647	0.9286	0.8387	2 [2, 8]
	SSA	**0.8928**	0.8235	**0.9925**	**0.9001**	3 [4, 6, 11]
	GA	0.8215	0.8462	0.7857	0.8148	5 [4, 6, 7, 9, 11]
	GWO	0.8571	**0.9166**	0.7857	0.8461	1 [4]
	HHO	0.8219	0.7647	0.9285	0.8387	1 [3]
	WOA	0.8215	0.80	0.8571	0.8276	2 [6, 8]
KNN	PSO	0.8254	0.7368	**0.9947**	0.8465	3 [1, 4, 6]
	SSA	0.8576	0.7777	0.9911	0.8715	4 [2, 3, 8, 10]
	GA	0.9286	0.875	0.9915	0.9296	3 [2, 7, 10]
	GWO	0.8936	0.8666	0.9285	0.8965	2 [1, 10]
	HHO	**0.9658**	**0.9941**	0.9572	**0.9753**	4 [1, 3, 7, 9]
	WOA	0.8926	0.8235	0.9912	0.8996	9 [1, 2, 4, 5, 7, 8, 9, 10, 11]
DT	PSO	0.9286	0.875	**0.9925**	0.9301	6 [5, 6, 7, 8, 9, 10]
	SSA	**0.9652**	**0.9898**	0.9286	**0.9582**	7 [1, 3, 4, 5, 8, 10, 11]
	GA	0.8925	0.9231	0.8572	0.8889	4 [1, 3, 5, 7]
	GWO	0.8965	0.8235	0.9924	0.9001	4 [1, 6, 7, 9]
	HHO	0.8573	0.9166	0.7857	0.8461	6 [4, 5, 6, 7, 10, 12]
	WOA	0.8254	0.80	0.8571	0.8276	5 [0, 2, 5, 6, 7]
AB	PSO	0.8589	0.8125	0.9286	0.8666	6 [2, 3, 5, 6, 9, 11]
	SSA	0.8962	0.8666	0.9285	0.8965	5 [3, 5, 6, 8, 9]
	GA	0.9286	0.875	0.9942	0.9307	7 [2, 4, 5, 6, 8, 10, 12]
	GWO	**0.9656**	**0.9333**	**0.9957**	**0.9634**	5 [1, 4, 8, 9, 11]
	HHO	0.8225	0.80	0.8571	0.8275	2 [1, 5]
	WOA	0.8572	0.7777	0.9854	0.8693	6 [3, 5, 8, 9, 10, 11]
NB	PSO	0.8929	**0.9948**	0.7857	0.8779	3 [1, 10, 11]
	SSA	0.8254	0.9090	0.7142	0.7999	2 [1, 5]
	GA	0.8216	0.8461	0.7857	0.8148	1 [11]
	GWO	**0.8939**	0.9231	**0.8571**	**0.8888**	2 [2, 5]
	HHO	0.8214	0.9789	0.6428	0.7761	1 [11]
	WOA	0.8575	0.9788	0.7142	0.8258	1 [11]
LR	PSO	0.8926	0.8235	**0.9974**	0.9021	7 [0, 3, 5, 6, 8, 11, 12]
	SSA	0.8936	0.8666	0.9286	0.8965	5 [2, 3, 6, 8, 9]
	GA	**0.9286**	**0.9915**	0.8571	**0.9194**	4 [0, 1, 5, 11]
	GWO	0.8956	0.9231	0.8571	0.8888	2 [2, 5]
	HHO	0.8965	0.9231	0.7857	0.8752	10 [0, 1, 2, 3, 4, 5, 7, 9, 10, 12]
	WOA	0.8573	0.8125	0.9286	0.8666	3 [1, 5, 11]
MLP	PSO	0.76	0.8181	0.6428	0.7199	7 [1, 3, 4, 6, 8, 11, 12]
	SSA	0.76	0.7059	0.8571	0.7742	5 [3, 5, 7, 8, 9]
	GA	0.8571	0.8125	0.9285	0.8666	6 [2, 3, 4, 5, 8, 9]
	GWO	0.8215	**0.9090**	0.7143	0.7999	4 [2, 5, 8, 9]
	HHO	0.7858	0.75	0.8571	0.7998	7 [1, 2, 3, 4, 5, 11, 12]
	WOA	**0.8927**	0.8666	**0.9286**	**0.8965**	8 [0, 1, 3, 5, 6, 8, 10, 12]

Table A3. Performance results of metrics-based Moodle dataset

Machine Learning Algorithms	Feature Selection Techniques	AUC	Precision	Recall	F_1-score	N_Features
RF	PSO	0.9452	0.9090	0.7143	0.80	5 [0, 2, 3, 5, 12]
	SSA	**0.9573**	**0.9406**	0.9761	**0.9581**	5 [1, 5, 6, 9, 11]
	GA	0.9407	0.9028	0.9863	0.9427	6 [2, 3, 5, 6, 10, 11]
	GWO	0.9366	0.8899	**0.9966**	0.9402	5 [2, 4, 5, 8, 10]
	HHO	0.9435	0.9085	0.9863	0.9458	7 [0, 1, 3, 5, 6, 8, 11]
	WOA	0.9212	0.8639	0.9847	0.9203	8 [0, 1, 2, 4, 5, 8, 10, 12]
SVM	PSO	0.8239	0.8593	0.7739	0.8144	7 [1, 2, 3, 5, 8, 10, 11]
	SSA	0.8596	**0.8977**	0.8116	0.8525	6 [2, 4, 8, 10, 11, 12]
	GA	0.8496	0.8592	0.8356	0.8472	8 [1, 2, 5, 7, 8, 10, 11, 12]
	GWO	**0.8659**	0.8844	0.8391	**0.8611**	7 [1, 3, 5, 8, 10, 11, 12]
	HHO	0.8425	0.8401	**0.8459**	0.8429	3 [2, 3, 8]
	WOA	0.8335	0.8679	0.7876	0.8258	8 [0, 2, 3, 6, 8, 9, 11, 12]
KNN	PSO	**0.9616**	**0.9269**	**0.9968**	**0.9605**	7 [1, 2, 3, 4, 8, 10, 12]
	SSA	0.9539	0.9179	0.9965	0.9556	8 [0, 1, 3, 5, 8, 9, 11, 12]
	GA	0.9558	0.9235	0.9931	0.9571	6 [0, 4, 8, 9, 11, 12]
	GWO	0.9572	0.9265	0.9848	0.9547	7 [0, 2, 3, 7, 9, 11, 12]
	HHO	0.9588	0.9241	0.9878	0.9549	8 [0, 3, 4, 5, 7, 8, 10, 11]
	WOA	0.9578	0.9238	0.9966	0.9588	7 [0, 3, 4, 5, 7, 8, 11]
DT	PSO	0.9865	0.9765	0.9965	0.9864	6 [0, 3, 5, 6, 10, 11]
	SSA	**0.9968**	**0.9962**	0.9963	**0.9962**	5 [0, 4, 5, 6, 10]
	GA	0.9949	0.9932	**0.9966**	0.9948	5 [0, 4, 5, 7, 10]
	GWO	0.9869	0.9797	0.9932	0.9864	5 [2, 3, 6, 8, 11]
	HHO	0.9897	0.9831	0.9965	0.9897	7 [1, 2, 3, 4, 5, 6, 10]
	WOA	0.9885	0.9863	0.9897	0.9881	7 [0, 3, 5, 8, 9, 10, 12]
AB	PSO	0.9708	0.9661	0.9762	0.9711	7 [2, 5, 6, 8, 10, 11, 12]
	SSA	0.9674	**0.9823**	0.9521	0.9669	4 [5, 8, 9, 12]
	GA	0.9556	0.9291	**0.9863**	0.9568	5 [1, 3, 5, 9, 11]
	GWO	**0.9745**	0.9792	0.9692	**0.9742**	7 [1, 2, 4, 5, 6, 10, 11]
	HHO	0.9578	0.9435	0.9726	0.9578	13 [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]
	WOA	0.9591	0.9437	0.9761	0.9595	7 [1, 2, 5, 6, 8, 9, 11]
NB	PSO	0.7398	0.8017	0.6369	0.7099	2 [5, 12]
	SSA	0.7448	0.8235	0.6233	0.7095	4 [1, 4, 5, 12]
	GA	**0.8031**	0.8525	**0.7329**	**0.7882**	3 [5, 10, 12]
	GWO	0.7456	0.7582	0.7089	0.7327	3 [5, 11, 12]
	HHO	0.7486	0.8169	0.6404	0.7179	2 [5, 12]
	WOA	0.7696	**0.8969**	0.5959	0.7161	1 [5]
LR	PSO	0.8356	0.7734	**0.9589**	0.8563	9 [1, 3, 5, 6, 7, 8, 9, 11, 12]
	SSA	0.8589	0.8046	0.9452	0.8691	9 [0, 1, 3, 5, 6, 7, 8, 9, 11]
	GA	0.8169	0.7621	0.9212	0.8341	7 [4, 6, 7, 8, 9, 10, 12]
	GWO	0.8659	**0.8292**	0.9143	0.8697	7 [1, 4, 5, 6, 7, 9, 10]
	HHO	0.8069	0.7687	0.8767	0.8192	9 [1, 2, 3, 4, 6, 7, 9, 11, 12]
	WOA	**0.8756**	0.8073	0.976	**0.8837**	11 [0, 1, 2, 3, 5, 6, 7, 8, 9, 11, 12]
MLP	PSO	0.7828	0.8113	0.7363	0.7719	7 [1, 3, 4, 5, 8, 9, 10]
	SSA	0.9297	0.9114	0.9521	0.9313	10 [1, 2, 4, 3, 5, 8, 9, 10, 11, 12]
	GA	**0.9439**	**0.9218**	0.9692	**0.9449**	9 [0, 2, 3, 4, 5, 8, 10, 11, 12]
	GWO	0.8956	0.8434	0.9589	0.8974	5 [3, 5, 8, 9, 11]
	HHO	0.9023	0.8983	0.9075	0.9029	8 [0, 3, 5, 8, 9, 10, 11, 12]
	WOA	0.9356	0.8944	**0.9863**	0.9381	5 [3, 4, 5, 8, 9]

Table A4. Performance results of metrics-based PHPMyAdmin dataset

Machine Learning Algorithms	Feature Selection Techniques	AUC	Precision	Recall	F_1-score	N_Features
RF	PSO	0.8965	0.8333	0.8856	0.8586	3 [3, 9, 11]
	SSA	0.9327	0.9643	0.90	0.9310	6 [1, 3, 4, 6, 8, 9]
	GA	0.8643	0.8621	0.8625	0.8623	4 [3, 7, 10, 11]
	GWO	0.8994	0.9615	0.8333	0.8929	3 [2, 10, 12]
	HHO	**0.9661**	**0.9655**	**0.9489**	**0.9658**	5 [1, 5, 8, 9, 12]
	WOA	0.8304	0.8333	0.8453	0.8392	6 [2, 4, 5, 8, 10, 12]
SVM	PSO	0.7802	0.8148	0.7333	0.7719	3 [0, 3, 10]
	SSA	0.7799	0.7666	0.7931	0.7797	3 [3, 5, 10]
	GA	**0.8306**	**0.8276**	**0.8279**	**0.8277**	4 [1, 2, 8, 10]
	GWO	0.7629	0.7666	0.7698	0.7688	5 [1, 4, 5, 7, 10]
	HHO	0.7965	0.80	0.8154	0.8076	3 [3, 8, 9]
	WOA	0.7305	0.7916	0.6333	0.7037	1 [9]
KNN	PSO	0.85	0.7631	0.8496	0.8041	6 [2, 3, 4, 5, 8, 12]
	SSA	0.8666	0.7838	0.8989	0.8374	6 [3, 5, 7, 8, 11, 12]
	GA	0.8655	0.8181	0.9310	0.8709	4 [3, 7, 10, 12]
	GWO	**0.9161**	**0.875**	**0.9655**	**0.9181**	3 [2, 11, 12]
	HHO	0.8811	0.8709	0.90	0.8852	4 [2, 4, 5, 10]
	WOA	0.8626	0.8055	0.8655	0.8344	8 [2, 4, 5, 6, 7, 9, 10, 12]
DT	PSO	0.9488	0.9643	0.9311	0.9473	4 [1, 3, 5, 10]
	SSA	0.9327	0.9032	0.9655	0.9333	8 [1, 3, 5, 7, 9, 10, 11, 12]
	GA	0.9827	**0.9777**	0.9655	0.9715	6 [3, 4, 6, 8, 10, 12]
	GWO	**0.9833**	0.9666	**0.9897**	**0.9831**	4 [1, 3, 6, 9]
	HHO	0.9494	0.9655	0.9333	0.9491	9 [2, 3, 4, 5, 6, 8, 9, 10, 12]
	WOA	0.9488	0.9643	0.9310	0.9474	3 [1, 3, 5]
AB	PSO	0.95	0.9063	**0.9782**	0.9408	8 [1, 2, 5, 6, 8, 9, 10, 12]
	SSA	**0.9609**	**0.9666**	**0.9665**	0.9665	6 [1, 4, 5, 7, 9, 11]
	GA	0.9494	0.9333	0.9655	0.9492	6 [0, 2, 4, 5, 9, 11]
	GWO	0.9488	0.9355	0.9666	0.9507	5 [5, 6, 8, 10, 11]
	HHO	0.9321	0.9333	0.9215	0.9273	10 [1, 2, 3, 4, 5, 6, 7, 9, 10, 11, 12]
	WOA	0.9327	0.9032	0.9655	0.9333	7 [0, 5, 6, 7, 9, 10, 11]
NB	PSO	0.7437	0.7027	**0.8666**	**0.7762**	6 [0, 1, 5, 9, 10, 11]
	SSA	**0.7638**	0.7272	0.8276	0.7742	5 [1, 4, 5, 10, 11]
	GA	0.7626	0.7666	0.7661	0.7665	6 [0, 1, 5, 6, 9, 11]
	GWO	0.7438	0.8181	0.6208	0.7059	3 [5, 10, 11]
	HHO	0.7311	**0.8184**	0.60	0.6923	5 [0, 1, 5, 8, 11]
	WOA	0.7454	0.75	0.7241	0.7368	2 [0, 8]
LR	PSO	0.8638	0.8437	**0.90**	**0.8709**	6 [1, 2, 3, 5, 7, 9]
	SSA	**0.8649**	0.8387	0.8966	0.8666	7 [0, 2, 3, 9, 10, 11, 12]
	GA	0.8465	**0.8846**	0.7931	0.8363	7 [2, 4, 6, 8, 9, 10, 12]
	GWO	0.8298	0.8519	0.7933	0.8214	5 [1, 2, 7, 8, 9]
	HHO	0.7971	0.8214	0.7666	0.7931	8 [1, 2, 4, 5, 7, 8, 9, 12]
	WOA	0.7609	0.8261	0.6552	0.7308	6 [1, 2, 4, 6, 9, 10]
MLP	PSO	0.7477	0.8261	0.6333	0.7169	6 [0, 2, 5, 7, 9, 12]
	SSA	0.7465	0.7187	0.7931	0.7541	9 [0, 2, 3, 5, 8, 9, 10, 12]
	GA	0.8393	0.88	0.7586	0.8148	6 [2, 4, 5, 8, 10, 12]
	GWO	**0.9149**	**0.9285**	0.8965	**0.9122**	5 [1, 5, 8, 10, 12]
	HHO	0.8649	0.8387	0.8966	0.8666	10 [1, 2, 4, 5, 7, 8, 9, 10, 11, 12]
	WOA	0.8293	0.7941	**0.90**	0.8437	9 [0, 2, 3, 5, 7, 8, 10, 11, 12]

Table A5: Performance results of metrics-based JavaScript dataset

Machine learning algorithms	Feature selection techniques	AUC	Precision	Recall	F_1-score	N_Features
RF	PSO	0.9544	0.96	0.9583	0.9592	18 [1, 3, 5, 6, 7, 8, 10, 11, 12, 13, 15, 16, 17, 20, 22, 23, 27, 34]
	SSA	0.9653	0.9732	0.9567	0.9649	12 [1, 5, 6, 7, 9, 15, 20, 22, 24, 25, 27, 30]
	GA	**0.9705**	**0.9798**	0.9605	**0.9701**	13 [1, 3, 4, 5, 6, 11, 13, 15, 16, 19, 23, 28, 30]
	GWO	0.9699	0.9708	**0.9689**	0.9698	10 [1, 2, 4, 7, 8, 13, 25, 29, 30, 34]
	HHO	0.9578	0.9673	0.9473	0.9572	23 [0, 1, 2, 3, 5, 6, 7, 8, 9, 10, 11, 14, 15, 18, 20, 21, 22, 24, 26, 29, 30, 31, 33]
	WOA	0.9563	0.9691	0.9426	0.9557	13 [1, 2, 3, 5, 7, 8, 9, 16, 17, 21, 22, 24]
SVM	PSO	0.7423	0.7826	0.6707	0.7223	7 [0, 1, 2, 4, 8, 20, 26]
	SSA	0.7456	0.8707	0.5766	0.6938	18 [1, 4, 5, 6, 9, 12, 14, 15, 16, 17, 19, 20, 22, 25, 27, 30, 32, 34]
	GA	**0.8035**	0.8554	0.7291	**0.7872**	13 [2, 4, 6, 7, 8, 10, 15, 16, 20, 21, 26, 27, 32]
	GWO	0.7569	0.7845	0.7055	0.7429	12 [2, 7, 8, 9, 10, 11, 14, 15, 16, 20, 26, 33]
	HHO	0.5652	0.5374	**0.9322**	0.6818	4 [7, 14, 18, 20]
	WOA	0.7516	**0.8721**	0.5898	0.7037	12 [0, 2, 4, 7, 9, 11, 13, 17, 24, 26, 30, 33]
KNN	PSO	0.9318	**0.9105**	0.9577	0.9335	15 [1, 2, 6, 9, 12, 14, 15, 16, 20, 21, 22, 24, 31, 33, 34]
	SSA	**0.9322**	0.9041	**0.9671**	**0.9345**	16 [4, 5, 7, 11, 12, 13, 14, 15, 19, 20, 21, 23, 24, 25, 26, 31]
	GA	0.9203	0.9011	0.9435	0.9218	15 [2, 4, 5, 6, 7, 10, 11, 13, 15, 18, 19, 25, 27, 31, 34]
	GWO	0.9312	0.9032	0.9661	0.9336	13 [1, 7, 11, 14, 15, 17, 20, 24, 25, 27, 31, 33, 34]
	HHO	0.9186	0.8952	0.9483	0.9209	16 [1, 2, 5, 7, 10, 11, 13, 15, 16, 18, 21, 28, 29, 31, 32, 34]
	WOA	0.9295	0.9043	0.9604	0.9316	23 [2, 4, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 21, 22, 23, 25, 27, 30, 31, 32]
DT	PSO	**0.9605**	0.9579	**0.9633**	**0.9606**	16 [3, 5, 6, 7, 8, 11, 12, 15, 16, 17, 22, 28, 29, 31, 32, 34]
	SSA	0.9532	0.9538	0.9529	0.9534	11 [4, 7, 9, 10, 11, 14, 15, 20, 22, 23, 33]
	GA	0.9592	**0.9595**	0.9586	0.9591	13 [3, 6, 8, 9, 13, 14, 16, 18, 20, 23, 25, 27, 32]
	GWO	0.9551	0.9523	0.9586	0.9554	11 [9, 10, 11, 15, 17, 20, 22, 25, 28, 29, 32]
	HHO	0.9584	0.9479	0.9595	0.9537	13 [0, 8, 9, 11, 13, 16, 17, 18, 19, 22,,26, 31, 34]
	WOA	0.9567	0.9525	0.9614	0.9569	11 [6, 7, 13, 14, 15, 19, 21, 23, 24, 25, 30]
AB	PSO	**0.8923**	**0.9273**	0.8513	**0.8877**	20 [0, 2, 5, 8, 9, 11, 12, 13, 15, 17, 18, 19, 21, 23, 24, 25, 27, 30, 32, 33]
	SSA	0.8749	0.9037	0.8392	0.8702	21 [1, 3, 4, 5, 6, 8, 10, 12, 15, 17, 20, 21, 23, 24, 25, 26, 28, 30, 31, 32, 34]
	GA	0.8797	0.9147	0.8373	0.8743	12 [4, 7, 8, 9, 12, 15, 17, 18, 20, 21, 25, 33]
	GWO	0.8892	0.9183	**0.8561**	0.8861	8 [8, 12, 13, 15, 17, 18, 21, 22]
	HHO	0.8858	0.9108	0.8551	0.8821	19 [1, 3, 4, 5, 6, 7, 9, 10, 12, 13, 17, 20, 22, 25, 28, 30, 31, 33, 34]
	WOA	0.8721	0.9211	0.8137	0.8641	13 [3, 4, 6, 10, 11, 13, 16, 17, 19, 21, 26, 27, 31]
NB	PSO	**0.6929**	**0.6414**	**0.8749**	**0.7401**	13 [1, 2, 3, 4, 8, 14, 15, 18, 21, 24, 25, 30, 34]

Table A5 continued

Machine learning algorithms	Feature selection techniques	AUC	Precision	Recall	F_1-score	N_Features
NB	SSA	0.6036	0.5802	0.7488	0.6538	22 [1, 2, 3, 4, 6, 8, 9, 10, 11, 13, 15, 16, 17, 19, 21, 23, 26, 27, 29, 31, 33, 34]
	GA	0.6553	0.6071	0.7796	0.6826	8 [8, 9, 14, 16, 21, 27, 29, 30]
	GWO	0.6396	0.5975	0.8561	0.7038	8 [5, 10, 12, 14, 15, 17, 26, 33]
	HHO	0.6271	0.5978	0.7761	0.6754	17 [3, 4, 5, 6, 7, 8, 9, 12, 16, 17, 19, 21, 23, 26, 29, 31, 34]
	WOA	0.5654	0.5363	0.5671	0.5513	6 [2, 10, 17, 25, 31, 33]
LR	PSO	0.7226	0.7095	0.7535	0.7308	13 [4, 6, 8, 9, 11, 12, 14, 15, 18, 20, 23, 24, 25, 30]
	SSA	0.6965	0.6618	0.8043	0.7261	15 [5, 7, 8, 10, 11, 13, 19, 20, 22, 23, 24, 25, 28, 32, 33]
	GA	**0.7362**	**0.7307**	0.7478	0.7392	15 [8, 12, 13, 18, 19, 20, 21, 22, 23, 25, 27, 28, 29, 31, 33]
	GWO	0.7238	0.6904	**0.8118**	**0.7462**	16 [0, 1, 3, 6, 11, 12, 14, 16, 19, 24, 25, 28, 29, 30, 31, 33]
	HHO	0.6942	0.6648	0.7855	0.7203	24 [1, 2, 6, 7, 8, 9, 10, 11, 13, 14, 15, 17, 18, 19, 20, 22, 23, 24, 25, 26, 29, 31, 32, 33]
	WOA	0.7063	0.6931	0.7394	0.7155	27 [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 18, 19, 20, 23, 24, 25, 26, 27, 28, 29, 30]
MLP	PSO	0.6603	0.7143	0.5551	0.6247	22 [2, 3, 4, 5, 7, 8, 9, 10, 13, 14, 15, 16, 17, 18, 22, 24, 26, 28, 30, 32, 33, 34]
	SSA	0.8573	**0.8916**	0.8127	0.8504	18 [1, 2, 3, 6, 7, 11, 12, 13, 15, 16, 22, 23, 24, 26, 30, 32, 33, 34]
	GA	0.8546	0.8632	0.8429	**0.8526**	21 [0, 1, 2, 3, 4, 5, 6, 7, 11, 13, 15, 16, 17, 19, 20, 21, 23, 24, 27, 33]
	GWO	0.8345	0.8099	0.8739	0.8407	16 [2, 3, 4, 8, 9, 16, 18, 21, 22, 23, 26, 27, 30, 32, 33, 34]
	HHO	**0.8745**	0.8783	0.8692	0.8737	19 [0, 2, 45, 6, 8, 9, 10, 11, 12, 13, 14, 16, 21, 22, 25, 26, 27, 33]
	WOA	0.8467	0.8275	**0.8758**	0.8509	23 [0, 1, 2, 3, 4, 6, 7, 8, 9, 13, 14, 15, 16, 19, 20, 21, 23, 24, 25, 27, 32, 33, 34]

Table A6. Performance results of text-features-based Drupal dataset

Machine learning algorithms	Feature selection sechniques	AUC	Precision	Recall	F_1-score	N_Features
RF	PSO	0.8929	0.8235	0.8878	0.8544	1740
	SSA	0.9289	0.875	0.9156	0.8948	1814
	GA	0.9658	0.9356	0.9286	0.9321	1614
	GWO	**0.9666**	**0.9587**	**0.9289**	**0.9435**	442
	HHO	0.8928	0.8666	0.9286	0.8965	956
	WOA 0	0.9289	0.9283	0.9254	0.9268	177
SVM	PSO	0.8929	0.9231	0.8571	0.8888	1800
	SSA	0.8578	0.8125	**0.9286**	0.8666	1862
	GA	**0.9289**	**0.9145**	0.8571	**0.8848**	1519
	GWO	0.8573	0.9166	0.7857	0.8461	513
	HHO	0.8571	0.7898	0.7143	0.7501	158
	WOA	0.8215	0.8462	0.7857	0.8148	362
KNN	PSO	0.8929	0.8666	0.9286	0.8965	1739
	SSA	0.9641	0.9333	0.9789	0.9555	1855
	GA	0.9642	0.9845	0.9285	0.9556	1576
	GWO	**0.9929**	**0.9912**	**0.9899**	**0.9903**	597
	HHO	0.8954	0.8235	0.9087	0.8641	804
	WOA	0.9643	0.9356	0.9286	0.9321	122
DT	PSO	0.8216	0.7647	0.9286	0.8387	1808
	SSA	0.8928	0.8235	0.8812	0.8513	1876
	GA	**0.9648**	**0.9333**	**0.9485**	**0.9608**	1736
	GWO	0.9285	0.875	0.9148	0.8945	528
	HHO	0.8573	0.7777	0.8821	0.8266	1836
	WOA	0.8929	0.9974	0.7857	0.8799	1834
AB	PSO	0.8929	0.8666	0.9113	0.8883	1831
	SSA	0.9286	0.875	0.9142	0.8942	1825
	GA	**0.9648**	**0.9745**	**0.9486**	**0.9614**	1614
	GWO	0.9285	0.875	0.9227	0.8982	1181
	HHO	0.9642	0.9333	0.9318	0.9325	1509
	WOA	0.8931	0.8235	0.8988	0.8595	1921
NB	PSO	**0.9289**	0.8847	0.8572	0.8707	1738
	SSA	0.8572	0.8452	0.7143	0.7743	1899
	GA	0.8929	0.8235	**0.8788**	0.8506	1532
	GWO	0.8927	0.8154	0.7857	0.8003	302
	HHO	0.8214	0.9090	0.7143	0.7999	1313
	WOA	0.8927	**0.9231**	0.8571	**0.8888**	1499
LR	PSO	0.9642	0.9333	0.9415	0.9374	1805
	SSA	0.9288	0.875	0.9012	0.8879	1870
	GA	0.9542	0.9233	0.9892	0.9552	1584
	GWO	**0.9982**	**0.9911**	**0.9988**	**0.9949**	365
	HHO	0.9641	0.9312	0.9325	0.9318	2007
	WOA	0.9682	0.9433	0.9512	0.9472	360
MLP	PSO	0.8929	0.8666	0.9286	0.8966	1798
	SSA	0.8234	0.7647	0.9287	0.8387	1915
	GA	0.9613	0.9224	0.9825	0.9515	1514
	GWO	**0.9986**	**0.9901**	**0.9928**	**0.9914**	642
	HHO	0.9642	0.9333	0.9242	0.9246	815
	WOA	0.9788	0.9333	0.9415	0.9378	1813

Table A7. Performance results of text-features-based Moodle dataset

Machine learning algorithms	Feature selection techniques	AUC	Precision	Recall	F_1-score	N_Features
RF	PSO	0.9263	0.9029	0.9554	0.9284	7938
	SSA	0.9434	0.9164	0.9761	0.9453	8097
	GA	0.9383	0.9324	0.9452	0.9387	7836
	GWO	0.9315	0.8937	**0.9794**	0.9346	3508
	HHO	**0.9469**	**0.9364**	0.9589	**0.9475**	4898
	WOA	0.9263	0.9055	0.9521	0.9282	4468
SVM	PSO	0.7842	0.9415	0.6062	0.7375	7843
	SSA	**0.8031**	0.9784	**0.6198**	**0.7589**	8048
	GA	0.7774	0.9133	0.6131	0.7336	8148
	GWO	0.7723	0.9035	0.6096	0.7281	7081
	HHO	0.7739	0.9348	0.5891	0.7227	8110
	WOA	0.7825	**0.9941**	0.5684	0.7233	5425
KNN	PSO	**0.9421**	0.8997	**0.9829**	**0.9394**	8049
	SSA	0.9001	0.8498	0.9692	0.9056	8146
	GA	0.9221	0.8742	0.9761	0.9223	8165
	GWO	0.9224	**0.9126**	0.9657	0.9384	5653
	HHO	0.9386	0.8816	0.9692	0.9233	8273
	WOA	0.9365	0.8974	0.9589	0.9272	9683
DT	PSO	0.9232	0.8782	**0.9623**	0.9183	8032
	SSA	0.9359	0.8984	0.9384	0.9179	8182
	GA	0.9212	0.8907	0.9486	0.9187	7777
	GWO	**0.9561**	**0.9302**	0.9589	**0.9443**	4126
	HHO	0.9242	0.8846	0.9452	0.9139	7445
	WOA	0.9221	0.9085	0.9178	0.9131	13546
AB	PSO	0.8921	0.8225	0.9758	0.9026	8105
	SSA	0.9195	0.8769	0.9761	0.9238	8139
	GA	0.9161	0.8627	**0.9897**	0.9218	7558
	GWO	**0.9315**	**0.8937**	0.9795	**0.9346**	5260
	HHO	0.9195	0.8816	0.9692	0.9233	8304
	WOA	0.9092	0.8699	0.9623	0.9138	8374
NB	PSO	0.8442	0.7638	**0.9966**	0.8648	8060
	SSA	0.8168	0.7318	0.9818	0.8385	8229
	GA	**0.8631**	0**0.7849**	0.9778	**0.8707**	7596
	GWO	0.8356	0.7526	0.9818	0.8521	3992
	HHO	0.8185	0.7337	0.9878	0.8419	11754
	WOA	0.8322	0.7487	0.9888	0.8521	14599
LR	PSO	0.8938	0.8267	0.9966	0.9037	8056
	SSA	0.9024	0.8366	**0.9978**	0.9111	8259
	GA	0.9041	0.8391	0.9918	0.9091	7583
	GWO	0.8904	0.8202	0.9789	0.8925	3107
	HHO	**0.9144**	**0.8601**	0.9897	**0.9204**	7383
	WOA	0.8989	0.8319	0.9978	0.9082	7005
MLP	PSO	0.9023	0.8366	0.9789	0.9021	7847
	SSA	0.8938	**0.8965**	0.8904	0.8934	8247
	GA	0.8767	0.8437	0.9246	0.8822	8111
	GWO	**0.9144**	0.8757	0.9657	**0.9186**	7920
	HHO	0.8972	0.8295	0.9856	0.9008	6583
	WOA	0.8904	0.8221	**0.9966**	0.9009	9036

Table A8. Performance results of text-features-based PHPMyAdmin dataset

Machine learning algorithms	Feature selection techniques	AUC	Precision	Recall	F_1-score	N_Features
RF	PSO	0.9831	**0.9677**	0.9712	0.9694	2268
	SSA	0.9661	0.9375	0.9145	0.9258	2421
	GA	0.9789	0.9442	0.9645	0.9542	2118
	GWO	**0.9833**	0.9666	0.9918	**0.9791**	822
	HHO	0.9742	0.9555	**0.9945**	0.9656	1237
	WOA	0.8983	0.8965	0.8978	0.8972	1721
SVM	PSO	0.8644	0.92	0.7931	0.8518	2388
	SSA	0.8475	0.8166	0.7586	0.7865	2519
	GA	0.8478	0.8565	0.7333	0.7903	2041
	GWO	**0.9152**	**0.9311**	**0.90**	**0.9152**	560
	HHO	0.7626	0.90	0.60	0.72	2021
	WOA	0.8644	0.8541	0.7333	0.7891	462
KNN	PSO	0.9322	0.8824	0.8978	0.8903	2381
	SSA	0.9661	0.9375	0.9415	0.9395	2500
	GA	**0.9833**	**0.9666**	**0.9778**	**0.9722**	2390
	GWO	0.9662	0.9465	0.9558	0.9511	894
	HHO	0.8983	0.8333	0.8812	0.8566	1353
	WOA	0.9322	0.8787	0.9015	0.8899	2736
DT	PSO	0.9831	0.9345	0.9289	0.9317	2330
	SSA	0.9661	0.9476	0.9554	0.9516	2465
	GA	0.9492	0.9063	0.9331	0.9196	2349
	GWO	**0.9859**	**0.9677**	**0.9789**	**0.9736**	738
	HHO	0.8983	0.8286	0.8844	0.8555	1859
	WOA	0.8827	0.8235	0.9655	0.8888	2856
AB	PSO	0.9827	0.9544	0.9614	0.9578	2410
	SSA	0.9877	0.9456	0.9541	0.9498	2378
	GA	0.9661	0.9375	0.9542	0.9457	2197
	GWO	**0.9879**	**0.9667**	**0.9789**	**0.9727**	1540
	HHO	0.9152	0.9286	0.8965	0.9123	2195
	WOA	0.9616	0.9412	0.9433	0.9422	3200
NB	PSO	0.8305	0.75	0.8245	0.7855	2375
	SSA	**0.8474**	**0.7631**	**0.8355**	**0.7976**	2463
	GA	0.7966	0.7073	0.7889	0.7458	2080
	GWO	0.7627	0.6905	0.7088	0.6995	647
	HHO	0.7333	0.6444	0.7225	0.6813	2271
	WOA	0.8135	0.7317	0.8145	0.7708	3206
LR	PSO	**0.9666**	**0.9917**	**0.9333**	**0.9616**	2335
	SSA	0.9491	0.9121	0.8965	0.9042	2539
	GA	0.9322	0.9643	0.90	0.9311	2081
	GWO	0.9661	0.9356	0.9123	0.9238	624
	HHO	0.9155	0.9311	0.90	0.9153	2678
	WOA	0.9492	0.9655	0.9331	0.9492	1143
MLP	PSO	0.9316	0.9629	0.8965	0.9286	2360
	SSA	0.9655	0.9375	0.9412	0.9393	2478
	GA	0.9778	0.9485	0.9389	0.9437	2203
	GWO	**0.9878**	**0.9756**	0.9614	0.9684	685
	HHO	0.9831	0.9666	**0.9721**	**0.9693**	2624
	WOA	0.9491	0.9333	0.9655	0.9492	2073

Table A9. Performance results of text-features-based JavaScript dataset

Machine learning algorithms	Feature selection techniques	AUC	Precision	Recall	F_1-score	N_Features
RF	PSO	0.9985	0.9981	0.9929	0.9985	3172
	SSA	0.9972	0.9984	0.9945	0.9972	3282
	GA	0.9995	0.9878	0.9981	0.9978	3159
	GWO	0.9981	0.9991	0.9972	0.9981	2727
	HHO	0.9946	0.9978	0.9758	0.9882	5729
	WOA	**0.9995**	**0.9994**	**0.9991**	**0.9995**	6552
SVM	PSO	0.9609	0.9321	0.9943	0.9622	3272
	SSA	0.9665	0.9404	0.9962	0.9675	3300
	GA	0.9665	0.9435	0.9925	0.9674	3267
	GWO	0.9586	0.9333	0.9878	0.9597	1462
	HHO	**0.9788**	**0.9593**	**0.9991**	**0.9793**	1625
	WOA	0.9519	0.9255	0.9831	0.9534	5687
KNN	PSO	0.9741	0.9623	0.9868	0.9744	3221
	SSA	0.9675	0.9437	**0.9944**	0.9684	3216
	GA	0.9788	0.9652	0.9934	0.9792	3304
	GWO	0.9755	0.9676	0.9839	0.9757	2829
	HHO	0.9892	0.9859	0.9925	0.9892	7920
	WOA	**0.9896**	**0.9869**	0.9928	**0.9897**	9766
DT	PSO	0.9915	0.9887	0.9789	0.9837	3099
	SSA	0.9978	0.9784	0.9578	0.9679	3176
	GA	0.9847	0.9812	0.9846	0.9828	2901
	GWO	0.9914	0.9963	**0.9978**	**0.9972**	1204
	HHO	0.9745	0.9625	0.9562	0.9593	3168
	WOA	**0.9994**	**0.9988**	0.9921	0.9954	3433
AB	PSO	0.9947	0.9978	0.9952	**0.9964**	3043
	SSA	0.9985	0.9956	**0.9958**	0.9916	3250
	GA	0.9942	0.9924	0.9845	0.9884	3133
	GWO	**0.9995**	**0.9981**	0.9854	0.9917	7719
	HHO	0.9932	0.9954	0.9876	0.9914	8188
	WOA	0.9914	0.9911	0.9863	0.9886	8197
NB	PSO	0.9985	0.9991	0.9981	0.9986	3100
	SSA	0.9957	0.9953	0.9963	0.9957	3208
	GA	0.9995	0.9991	0.9990	**0.9996**	2817
	GWO	0.9995	**0.9994**	**0.9992**	0.9993	6659
	HHO	**0.9998**	0.9945	0.9946	0.9942	4856
	WOA	0.9978	0.9947	0.9952	0.9949	4522
LR	PSO	0.9912	0.9445	0.9685	0.9563	3051
	SSA	0.9818	0.9525	0.9669	0.9596	3210
	GA	0.9771	0.9859	0.9554	0.9704	2770
	GWO	**0.9912**	**0.9879**	0.9715	0.9795	4643
	HHO	0.9698	0.9781	0.9772	0.9776	5757
	WOA	0.9745	0.9878	**0.9884**	**0.9881**	6319
MLP	PSO	0.9976	0.9972	0.9981	0.9974	3019
	SSA	0.9991	0.9992	0.9989	0.9989	4195
	GA	0.9945	0.9965	0.9978	0.9984	3212
	GWO	**0.9995**	0.9981	0.9925	0.9991	3094
	HHO	0.9964	**0.9995**	0.9987	0.9994	5880
	WOA	0.9915	0.9991	**0.9996**	**0.9997**	2289

Authors and affiliations

Deepali Bassi
e-mail: deepalics.rsh@gndu.ac.in
ORCID: https://orcid.org/0000-0002-6744-1957
Department of Computer Science, Guru Nanak Dev University, India

Hardeep Singh
e-mail: hardeep.dcse@gndu.ac.in
Department of Computer Science, Guru Nanak Dev University, India

e-Informatica Software Engineering Journal
Volume 19, Issue 1, pages: 1–37
DOI: 10.37190/e-Inf250104

BibTeX

Emotion Classification on Software Engineering Q&A Websites

Didi Awovi Ahavi-Tete[*], Sangeeta Sangeeta[*]
[*]Corresponding authors: `didi.ahavitete@gmail.com`, `s.sangeeta@keele.ac.uk`

Article info

Dataset link: https://drive.google.com/drive/folders/1qXyLx9OhpHVcXLMTsTYdjhxhV-t6G54j

Keywords:
empirical and experimental studies in software engineering
data mining in software engineering
prediction models in software engineering
AI and knowledge based software engineering

Submitted: 20 Nov. 2023
Revised: 2 Oct. 2024
Accepted: 6 Oct. 2024
Available online: 19 Nov. 2024

Abstract

Background. With the rapid proliferation of question-and-answer websites for software developers like Stack Overflow, there is an increasing need to discern developers' emotions from their posts to assess the influence of these emotions on their productivity such as efficiency in bug fixing.
Aim. We aimed to develop a reliable emotion classification tool capable of accurately categorizing emotions in Software Engineering (SE) websites using data augmentation techniques to address the data scarcity problem because previous research has shown that tools trained on other domains can perform poorly when applied to SE domain directly.
Method. We utilized four machine learning techniques, namely BERT, CodeBERT, RFC (Random Forest Classifier), and LSTM. Taking an innovative approach to dataset augmentation, we employed word substitution, back translation, and easy data augmentation methods. Using these we developed sixteen unique emotion classification models: *EmoClassBERT--Original*, *EmoClassRFC-Original*, *EmoClassLSTMOriginal*, *EmoClassCodeBERT-Original*, *EmoClassLSTM-Substitution*, *EmoClassBERT-Substitution*, *EmoClassRFC-Substitution*, *EmoClassCodeBERT-Substitution*, *EmoClassBERT-Translation*, *EmoClassLSTM-Translation*, *EmoClassRFC Translation*, *EmoClassCodeBERT-Translation*, *EmoClassBERT-EDA*, *Emo ClassLSTM-EDA*, *EmoClassCodeBERT-EDA*, and *EmoClassRFC-EDA*. We compared the performance of this model on a gold standard state-of-the-art database and techniques (Multi-label SO BERT and EmoTxt).
Results. An initial investigation of models trained on the augmented datasets demonstrated superior performance to those trained on the original dataset. EmoClassLSTM-Substitution, EmoClassBERT-Substitution, EmoClassCodeBERT-Substitution, and EmoClassRFC-Substitution models show improvements of 13%, 5%, 5%, and 10% as compared to EmoClassLSTM-Original, EmoClassBERT-Original, EmoClassCodeBERT-Original, and EmoClassRFC-Original, respectively, in average F_1-score. The *EmoClassCodeBERT-Substitution* performed the best and outperformed the Multi-label SO BERT and Emotxt by 2.37% and 21.17%, respectively, in average F_1-score. A detailed investigation of the models on 100 runs of the dataset shows that BERT-based and CodeBERT-based models gave the best performance. This detailed investigation reveals no significant differences in the performance of models trained on augmented datasets and the original dataset on multiple runs of the dataset.
Conclusion. This research not only underlines the strengths and weaknesses of each architecture but also highlights the pivotal role of data augmentation in refining model performance, especially in the software engineering domain.

1. Introduction

Software engineering (SE) is a domain that, while inherently technical, is deeply influenced by human factors such as emotions, cognitive biases, and decision-making processes. These human-centric aspects play a crucial role in shaping the dynamics of software development, from team collaborations to the end product's quality [1]. The emotional undertones evident in various communication channels, whether in code comments, pull requests, or interactive developer forums, can provide insights into the effect of developers [2]. They can highlight potential misunderstandings, pinpoint areas that spark contention, or even forecast the emergence of software bugs and vulnerabilities [3, 4]. Such insights, if harnessed correctly, can be instrumental in anticipating issues and enhancing overall software development efficiency.

Previous research shows that emotion can greatly impact various software development activities. For example, positive emotion can improve job satisfaction and productivity [5]. Experiments by Girardi et al. [6] show that positive emotions occur when developers work on implementing new features. However, their results also show that negative emotions are triggered in developers when they encounter unexpected code behavior and missing documentation. It can also be caused by time pressure or being stuck with the task. A study by Graziotin et al. [7] shows possible consequences of positive and negative emotions. For example, their study shows that positive emotion leads to several positive consequences like high code quality, high motivation, higher creativity, etc., whereas negative emotion causes various negative consequences like low productivity, low participation, and work withdrawal. A study by Novielli et al. [8], shows that negative emotion can also lead to difficulty in learning new programming languages. All of the above examples indicate the importance of correctly recognizing the emotions of software developers.

The above research shows that emotion recognition is important for various software development tasks. However, it is found to be very challenging because of data scarcity issues. In the software engineering domain, there is limited availability of the ground truth or manually annotated data because manual annotation is resource resource-intensive task [9] [10]. Also, there are researches that show that emotion classification models trained on a dataset of other domains do not perform well when used in the software engineering domain [11]. Advancements in natural language processing (NLP) have unveiled powerful models like BERT, CodeBERT, LSTM networks, and ensemble methods like RFC. These models have demonstrated state-of-the-art results in various NLP tasks [12, 13], prompting exploration into their potential for emotion classification within the SE realm [14, 15]. Yet, one perennial challenge in machine learning (ML) and NLP tasks is the need for extensive and diverse training datasets [2]. Hence, there is a need to address this data scarcity issue. In this paper, we focus on improving the performance of emotion classification in the SE domain using the data augmentation technique.

Data augmentation, a technique of artificially enhancing the dataset size and variability, has shown promising results in improving model robustness and generalization [16]. Among various data augmentation techniques, word substitution and back translation have garnered attention for their ability to retain semantic integrity while introducing syntactic variability [2, 16, 17]. Additionally, Kufakou et al. [18] show that the easy data augmentation approach gave the best results in their experiment. This study aims to investigate the efficacy of data augmentation techniques with machine learning algorithms in the context of emotion classification in the SE domain. In this research, we utilized four machine learning techniques, namely Bidirectional Encoder Representations from Transformers (BERT), CodeBERT, Long

Short-Term Memory (LSTM) neural network, and the Random Forest Classifier (RFC) model. We used three data argumentation techniques: *word substitution*, *back translation*, and *Easy Data Augmentation (EDA)*. Using these we developed sixteen unique emotion classification models: *EmoClassBERTOriginal*, *EmoClassCodeBERT-Original*, *EmoClassRFC-Original*, *EmoClassLSTM-Original*, *EmoClassLSTM-Substitution*, *EmoClassBERT-Substitution*, *Emo ClassCodeBERT-Substitution*, *EmoClassRFC-Substitution*, *EmoClassBERT-Translation*, *EmoClassCodeBERT-Translation*, *EmoClassLSTM-Translation*, *EmoClassRFC-Translation*, *EmoClassBERT-EDA*, *EmoClassCodeBERT-EDA*, *EmoClassLSTM-EDA*, and *EmoClass RFC-EDA*. We evaluated the performance of the proposed model(s) on a gold-standard state-of-the-art database [19]. We compared its performance with state-of-art techniques Multi-label SO BERT and EmoTxt [14]. Specifically, we answer the following research questions in this study:

- **RQ1: Which classification model performs better between LSTM, BERT, CodeBERT, and RFC?** Experimental results show that the BERT and codeBERT model outperformed LSTM and RFC in emotion classification.
- **RQ2: An initial investigation: Can data augmentation improve the model's performance?** Experimental results show that models trained on the augmented datasets demonstrated superior performance to those trained on the original dataset. EmoClassLSTM-Substitution, EmoClassBERT-Substitution, EmoClassCodeBERT-Substitution, and EmoClassRFC-Substitution models show improvements of 13%, 5%, and 10% as compared to EmoClassLSTM-Original, EmoClassBERT-Original, and EmoClassRFC-Original, respectively, in average F_1-score.
- **RQ3: How do EmoClassLSTM, EmoClassBERT, EmoClassCodeBERT, and EmoClassRFC compare to existing tools?** The *EmoClassCodeBERT-Substitution* performed best and outperformed the Multi-label SO BERT and Emotxt by 2.37% and 21.17%, respectively, in average F_1-score.
- **RQ4: How does algorithm randomness affect the performance of the proposed models?** The BERT-based and CodeBERT models perform best for emotion classification. There is no significant difference in the performance of models trained on augmented and non-augmented data.

By bridging the advanced NLP techniques with the unique challenges and intricacies of SE texts, this research hopes to contribute a robust methodology for emotion recognition in this vital domain.

2. Background

In today's interconnected world, a vast number of individuals across the globe are utilizing various online platforms like blogs, forums, and social media sites to express their thoughts and share opinions. In the SE domain, online communities and channels have become prominent platforms for individuals to express their views and share experiences. SE communities, which include forums, chat groups, and dedicated platforms like GitHub[1] and Stack Overflow[2], serve as virtual gathering spaces for developers, programmers, and information technology project managers. These channels have emerged as valuable hubs of knowledge, where professionals discuss coding practices and issues [20]. Consequently,

[1]https://github.com/
[2]https://stackoverflow.com/

a substantial amount of valuable data is generated within these communities, forming a rich source of insights into the thoughts, opinions, and challenges software engineers worldwide face.

Liu [21] describes sentiment analysis, also known as opinion mining, as a discipline intersecting natural language processing, text mining, and computational linguistics. It evaluates the emotional tone of texts from diverse sources like social media, e-commerce sites, and blogs. This analysis aids organizations in discerning public sentiment, understanding product perceptions, and detecting emerging trends [22]. Emotion classification, also known as affective computing, is a subfield of sentiment analysis that focuses on identifying, understanding, and interpreting human emotions [23]. While sentiment analysis classifies the feelings expressed in a text into three categories: positive, negative, and neutral, emotion classification goes further to recognize a wide range of human emotions, including *Joy*, *Anger*, *Sadness*, *Surprise*, *Disgust*, and *Fear* [24].

The SE field is not only technical but also deeply human, involving collaboration, creativity, and problem-solving [25]. Emotions, like *Sadness*, *Anger*, and *Joy*, play a pivotal role in influencing productivity, team dynamics, and decision-making in SE [1, 26]. SE researchers have been employing sentiment analysis techniques as discussed in several applications [9, 15, 27]. For example, Murgia et al. [28] observed that issue reports carry emotions. Ortu et al. [3] reported that emotions could influence team communication, decision-making, and problem-solving strategies, thereby significantly affecting the software development process. They showed there is a correlation between emotion expressed in issue comments and bug-fixing productivity. They mined emotions from 560 000 Jira comments, revealing that expressions of *Joy* and *Love* correlated with faster issue resolutions, while *Sadness* was linked to longer delays. Understanding and addressing these emotions is essential for fostering a positive and productive work environment. Uddin et al. [29] mined the Application Programming Interface (API) discussion from StackOverflow and reported that sentiments can be used to predict pros and cons related to the adoption of APIs. Several studies use sentiment to detect issues in applications' reviews [30]. Gu et al. [31] analyzed sentiment in user reviews. They proposed the SUR-Miner model which helps classify user reviews into one of the predefined classes like aspect evaluation, bug reports, feature requests, praise, and others. SUR-Miner's ability to discern and categorize user feedback into predefined classes significantly enhances the analysis and interpretation of user sentiments when evaluating applications. Panichella et al. [32], used sentiment analysis techniques combined with natural language processing and text analysis to classify user reviews into the following classes: Information Giving, Information Seeking, Feature Request, and Problem Discovery. Their approach proves valuable for pinpointing problem areas in the software and directing efforts towards resolving those identified bugs. Furthermore, this method helps to quickly spot areas requiring improvement, empowering developers to swiftly address these issues and deploy new functionalities that align with end users' preferences and needs. Rahman et al. [33] used opinion mining to recommend insightful comments from source code on StackOverflow.

Despite the progress made in the field, Imran et al. [2], reported the unsuitability of state-of-the-art emotion categorization tools on SE data. The research also highlighted how the tool's accuracy decreases when trained on one communication channel and assessed on another. Hence, from this, we can say that emotion classification on SE Q&A websites is still a developing field of study. This research aims to develop an emotion classification algorithm capable of effectively identifying the emotions of software developers on SE communication channels to investigate and implement techniques for improving the accuracy and generalization of the prediction model. We use the gold standard (manually) annotated dataset[3] extracted from Stack Overflow extracted by Novielli et al. [19] for this research. By leveraging NLP, the preprocessing tasks were performed followed by the implementation of data augmentation techniques. We developed three emotion classification algorithms using the LSTM, the BERT, and the RFC model. The models were evaluated against the Multi-label Stack Overflow BERT model and EmoTxt presented in [14].

3. Related work

The recognition of the role of emotion in SE has gained substantial momentum within the academic and industrial communities in recent years. This section aims to provide an in-depth review of the literature relating to this topic, surveying the progression and future trajectories of this field.

3.1. Sentiment analysis in software engineering

In their research, Jongeling et al. [34] conducted an in-depth evaluation of the performance of two widely used sentiment analysis tools, namely SentiStrength [35] and NLTK [36]. Their analysis initially included four tools but ultimately focused on SentiStrength and NLTK. They used seven datasets for their investigation, including issue trackers and questions from Stack Overflow, a popular online platform for the programming community. Their findings highlighted a significant challenge when applying these sentiment analysis tools to SE contexts. Both SentiStrength and NLTK were initially developed for non-SE domains, which have substantial differences in language and sentiment expression compared to texts in the SE field. Their observations underscore the need for sentiment analysis tools specifically designed and trained for the unique characteristics of SE texts.

Guzman et al. [27] adopted a lexical-based technique to analyze the sentiments expressed in 60425 commit comments of 29 OSS projects. SentiStrength was used to convert emotions expressed in commit comments into quantitative values. SentiStrength allocates specific scores to tokens listed in a dictionary, which also encompasses common emoticons. Words expressing negative sentiments are assigned a value ranging between $[-5, -1]$, while those expressing positive sentiments receive a value between $[1, 5]$. Words with neutral sentiment are assigned values of 1 and -1. On the other hand, extreme sentiment expressions, words with very positive and negative feelings, are given scores of 5 and -5, respectively. A commit comment is considered to be positive if its overall emotion score falls within the range of $[1, 5]$, negative if the score is in the $[-1, -5]$ range, and neutral if the score lies within the $[-1, 1]$ range. Furthermore, an analysis was conducted on the correlation between these quantified emotions and various factors such as the programming language used, the team distribution, and others. The researchers emphasized looking beyond the average emotion score of the committed messages. They recommended considering both average positive and negative scores, and the spread of positive, negative, and neutral documents for a deeper understanding of the emotional content.

To overcome the limitations associated with SentiStrength, Islam and Zibran [37] implemented SentiStrength-SE, a sentiment analysis tool built upon SentiStrength (lexical-based

[3]https://github.com/collab-uniba/EmotionDatasetMSR18/blob/master/Emotions_GoldSandard_and Annotation.xlsx

approach) and specifically tailored to the SE domain. This tool integrates an understanding of the nuances and jargon used in the field, enabling it to interpret sentiment more accurately than general sentiment analysis tools. SentiStrength-SE proved superior to the original SentiStrength tool when evaluated using a substantial dataset. This dataset has 5600 issue comments from various SE projects.

The research led by Ahmed et al. [15] resulted in the creation of SentiCR, a specialized sentiment analysis tool for SE. This development came about due to the inadequacy of the existing tools they evaluated using their dataset of 2000 code review comments from 20 Open Source Software projects. SentiCR was developed using the Python programming language, incorporating the Natural Language Toolkit (NLTK) for language preprocessing tasks. Then, the scikit-learn library was employed for the supervised learning algorithms. As part of the data preprocessing tasks, the Term Frequency – Inverse Document Frequency (TF-IDF) method was used for feature extraction, and then eight supervised learning algorithms were evaluated. These include Adaptive Boosting, Decision Tree, Gradient Boosting Tree, Naive Bayes, Random Forest, Multilayer Perceptron, Support Vector Machine with Stochastic Gradient Descent and Linear Support Vector Machine. The researchers observed that the Gradient Boosting Tree performed better than other models, with an accuracy of 82%.

Calefato et al. [38] introduced Senti4SD, a sentiment polarity classifier. Over 4000 manually annotated posts from Stack Overflow served as the training and testing basis for the classifier. Senti4SD's semantic features are derived from a distributional semantic model (DSM) that utilizes word embedding. The DSM was established by executing Word2vec on a corpus of more than 20 million documents sourced from Stack Overflow, thereby generating word vectors that encapsulate the communication style of developers. Senti4SD, trained using Support Vector Machines (SVM), overcame the problem of negative bias prevalent in existing sentiment analysis tools by combining lexicon-based, keyword-based, and semantic features. Negative bias refers to the phenomenon where texts that are actually neutral in tone are incorrectly identified as expressing negative emotions. Notably, a 19% improvement in precision for the negative class and a 25% improvement in recall for the neutral class were observed when compared with SentiStrength.

In contrast to the aforementioned studies, our research focuses on developing an emotion recognition model tailored to the software engineering domain, with the unique ability to classify and differentiate between specific emotions. By doing so, we aim to provide a nuanced and domain-specific understanding of emotional states within the context of software development, which can have significant implications for improving the overall SE processes and work environment.

3.2. Emotion classification in software engineering

Identifying specific emotions, rather than just general sentiment, offers a richer understanding of software engineers' emotional states. This detailed perspective aids in grasping team dynamics, decision-making, and productivity [39]. For example, spotting frustration may indicate task challenges, while joy or satisfaction could signify successful teamwork or development. Responding to this need, Calefato et al. [40] proposed EmoTxt, an open-source toolkit tailored for emotion detection in text. It was trained on two key datasets: 4800 Stack Overflow posts created for the study, and 4000 Jira comments from Ortu et al. [3]. EmoTxt uses six binary classifiers to detect specific emotions: *Joy*, *Love*, *Sadness*, *Anger*, *Surprise*, and *Fear*. Utilizing a supervised learning approach with Support Vector Machines (SVM), it effectively identifies emotional patterns in written content.

Murgia et al. [20] developed classifiers for each emotion category (*Love*, *Joy*, *Sadness*, and *Neutral*), with each classifier calculating the probability of a particular emotion being present in a comment. They created five versions of each classifier using SVM, Naive Bayes (NB), Single Layer Perceptron (SLP), K-Nearest Neighbor (KNN), and Random Forest (RF). Using bootstrap validation on Apache Software Foundation project comments, the SVM classifiers proved most effective for detecting love, joy, and sadness, warranting further examination. The SVM models' performance was later assessed on a separate test set of comments.

Recognizing the limitations of traditional ML techniques, researchers began to explore more advanced methods, particularly Deep Learning (DL) algorithms. DL techniques are especially suitable for complex tasks such as emotion classification because they can handle high-dimensional data and capture intricate patterns within the data. Bleyl and Buxton [14] implemented BERT models for emotion recognition in Stack Overflow comments drawing on Novielli, et al's. [19] dataset. Due to the dataset's imbalance, they augmented underrepresented emotion classes. They also fine-tuned BERT for the SE context by adding 993 prevalent technical words and emoticons from Stack Overflow to BERT's tokenizer vocabulary. Then, leveraging Masked Language Modelling, they trained BERT on a large dataset of unlabeled Stack Overflow comments and fine-tuned it on the Stack Overflow annotated dataset. Their multi-label BERT model outperformed other models.

Our research builds upon previous studies by investigating various machine learning architectures and techniques to improve the overall performance of emotion recognition models in the context of software engineering. In this paper, we explored data augmentation methods for enhancing the robustness and generalization capabilities of our emotion recognition models. In this paper, we explored data augmentation methods for enhancing the robustness and generalization capabilities of our emotion recognition models. Imran et al. [2], proposed data augmentation-based techniques for emotion classification on the SE dataset to address the data scarcity issue. They report an improvement of 9.3% in micro F_1-score as compared to popular SE tools. However, they explore only 3 types of data augmentation techniques: Unconstrained, lexicon, and polarity-based. They used a stacked approach for data augmentation. They used a stacked approach for data augmentation. In this, we focus on using simple data augmentation techniques like "back translation" to find out their effectiveness for emotion classification in the software engineering domain. In addition, Imran et al. [2], use existing emotion classification models like ESEM-E, EMTk, and SEntiEmoji whereas, in the paper, we checked the efficiency of 3 classifiers LSTM, Radom Forest, and BERT for emotion classification on the augmented dataset. The approach used in this paper extends the work done by Imran et al. [2] by adding one more dimension of using data augmentation for emotion. classification

4. Methodology

Figure 1 provides an overview of the methodology for this study. We used the Stack Overflow dataset provided by Novielli et al. [19]. This is the gold standard dataset. We use Python programming language for implementing various machine learning libraries. We notice that this dataset is highly imbalanced in nature. Hence, we explored the uses of data augmentation methods for improving the accuracy of emotion detection. We developed three main emotion classification algorithms using RF, LSTM and BERT models. We then compare the performance of these different approaches. Finally, we evaluate how the implemented emotion classification algorithms compare to existing tools in the SE domain.

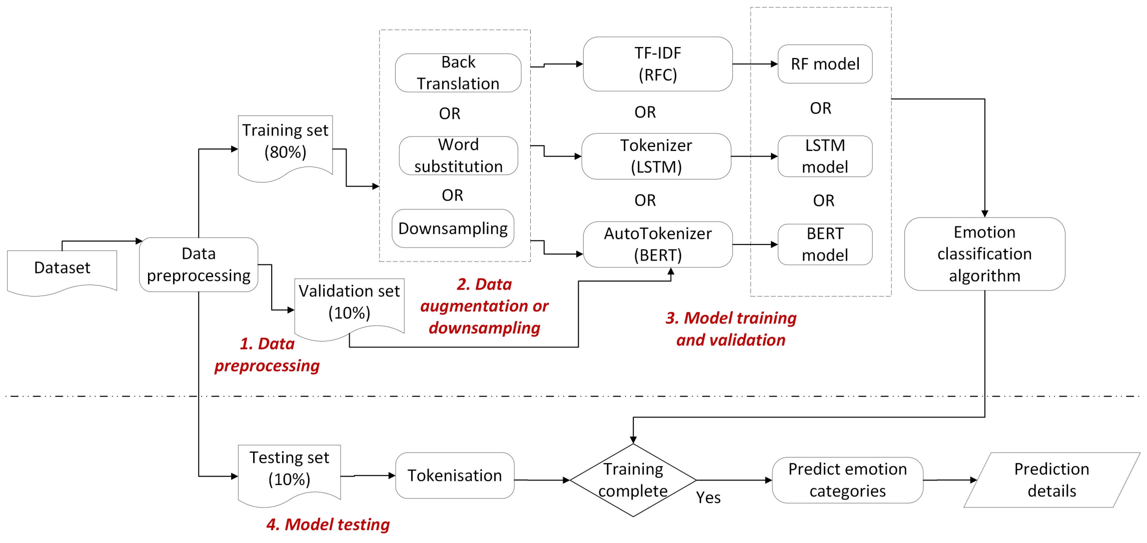


Figure 1. Overview of the methodology

4.1. Dataset description

The dataset contains 4800 Stack Overflow entries, encompassing questions, answers, and comments, and is a sample from the unlabeled Stack Overflow dataset of June 2008 to September 2015 [19]. Part of the Stack Exchange network of Q&A websites, Stack Overflow is a popular Q&A site for software developers. On Stack Overflow, users can ask questions, answer questions, vote on questions and answers, and earn reputation and badges. As discussed by Novielli et al. [19], the dataset was annotated by a group of twelve volunteers. Each entry received annotations from three different individuals, focusing on the six fundamental emotions (*Love*, *Joy*, *Surprise*, *Anger*, *Sadness*, and *Fear*). Determining the emotion for an observation relied on a majority consensus approach. If at least two of the three evaluators identified a specific emotion for an observation, then that emotion was assigned to the sample. Table 1 shows, an example of the dataset. However, not every observation-emotion combination was labeled, and some observations were labeled with more than one emotion. Approximately 56% of the comments are labeled with just one emotion, 6% are marked with two or more emotions, and the remaining comments are without emotion labels [14]. For this study, any post not annotated with emotion was regarded as devoid of emotion and, therefore, classified as neutral posts. This dataset is organized into individual worksheets for each emotion label: *Love*, *Joy*, *Surprise*, *Anger*, *Sadness*, and *Fear* [19]. The worksheets were, therefore, merged into a single sheet and saved as a CSV file.

Table 1. Examples from Novielli et al. dataset [19]

Text	Rater 1	Rater 2	Rater 3	Gold label
SVG transform on text attribute works excellent! This snippet, for example, will increase your text by 2× at Y-axis.	X		X	LOVE
Excellent! This is exactly what I needed. Thanks!	X	X	X	LOVE
Have added a modern solution as of May 2014 in answers below.				
Have you tried removing “preload” attribute? (Afraid I can’t be much help otherwise!)				

Table 2. Emotion label distribution

Number of observations conveying the emotion							Total
Love	Joy	Surprise	Anger	Sadness	Fear	Neutral	
1181	488	43	867	227	103	1918	4735

4.2. Data preprocessing

4.2.1. Text cleaning techniques

After merging the worksheets, some duplicates were identified in the dataset. Duplicate entries can lead to biased or skewed results because they do not represent unique instances of the data. The duplicated observations were removed from the experimental dataset. We removed duplicates from the dataset using a two-step process: 1) automated text matching and 2) manual verification. We removed a total of 65 duplicate entries (Love: 39, Joy: 3, Surprise: 2, Anger: 15, Sadness: 3, Fear: 3). We also notice the presence of some irrelevant attributes in the dataset such as information about the group, set, id, and raters. We removed all these attributes from the experimental dataset. The label Neutral was assigned to the entries not annotated in the original dataset. Table 2 shows the number of instances for each emotion category.

Removal of non-alphabetic characters. Non-alphabetic characters were removed as part of an essential approach designed to streamline raw textual data. This curtails the presence of excessive symbols, punctuations, and numbers seen as noise, which can add meaningless variability. By filtering out such characters, the resultant text not only becomes more readable but also more concise. This makes it more compatible with the strict requirements of computational processing and linguistic analysis, leading to a more efficient and accurate prediction algorithm [41].

Case folding. Furthermore, the entire text corpus was converted to lowercase to ensure the homogeneity of the dataset. This is because the words “Analysis” and “analysis”, though semantically identical, would be processed as separate tokens due to their case difference. Such distinctions introduce redundancies, thereby increasing the dimensionality of the data without adding meaningful variance [42]. Using consistent casing in the dataset ensures that the text is standardized. This standardization is vital for constructing consistent and reproducible models that can generalize effectively to unseen data, thereby enhancing the reliability of the prediction model.

Stop words, stemming and lemmatization. While it is usually essential to remove stop words when handling NLP tasks, in line with previous studies [38], stop words were not removed, as comments such as "I am happy with this output" and "I am not happy with this output" express different emotions. Neither Stemming nor lemmatization was performed since, according to Calefato et al., [38], a varied form might convey useful information.

4.2.2. Tokenization

Tokenization, in NLP, is breaking down the text into smaller pieces, known as "tokens". While these tokens are commonly individual words, they can also be sentences, parts of words, or even single characters [43]. The type of token selected is usually based on the particular NLP task. As an example, tokenizing the phrase "I love coding" results in ["I", "love", "coding"]. Tokenization is crucial because, before text data can be analyzed or fed into machine learning algorithms, it often must be transformed from its raw form into a structured format [44]. The Keras, BERT, and scikit-learn libraries were utilized to tokenize the Stack Overflow posts. The tokenization process was handled differently for the three models in the NLP task. For the LSTM model, the tokenizer module from the Keras library was used to process the posts, while the BERT tokenizer was used for the BERT model. The Term Frequency-Inverse Document Frequency (TF-IDF) was used to extract features for the RFC model. This essential step was carried out to convert the text into numerical form and to aid in building the vocabulary for the dataset.

4.3. Text exploratory analysis

Text Exploratory Analysis (TEA) aims to meticulously decipher the embedded structure, recurrent patterns, and potential aberrations within the dataset [45].

4.3.1. Sentiment polarity

Sentiment polarity in text analysis evaluates the overall sentiment or tone of a piece of writing. It categorizes the sentiment as positive, negative, or neutral, allowing for a quick assessment of the general mood of the expressed thoughts [46]. For instance, a statement such as *"Excellent, I'm glad that worked for you!"* would likely be categorized as possessing a positive polarity, whereas *"This is one of the shortcomings of DGV that I absolutely hate and why I almost always bind to an IEnumerable of an anonymous type."* would be attributed a negative polarity. A statement such as, *"I understand that server-side validation is an absolute must to prevent malicious users (or simply users who choose to disable javascript) from bypassing client-side validation"* might be considered neutral. We used TextBlob to obtain a brief overview of the sentiment polarity within the dataset. TextBlob, a Python library rooted in NLTK and Pattern, offers lexicon-based sentiment analysis by producing polarity and subjectivity scores. Polarity scores range from -1 (negative) to 1 (positive), with 0 being neutral. Subjectivity scores span from 0 (factual) to 1 (opinion-based).

Figure 2 showcases the sentiment polarity distribution in the dataset, detailing percentages of positive, negative, and neutral sentiments for an overall mood assessment. However, sentiment polarity provides a generalized perspective, missing the detailed layers of emotion

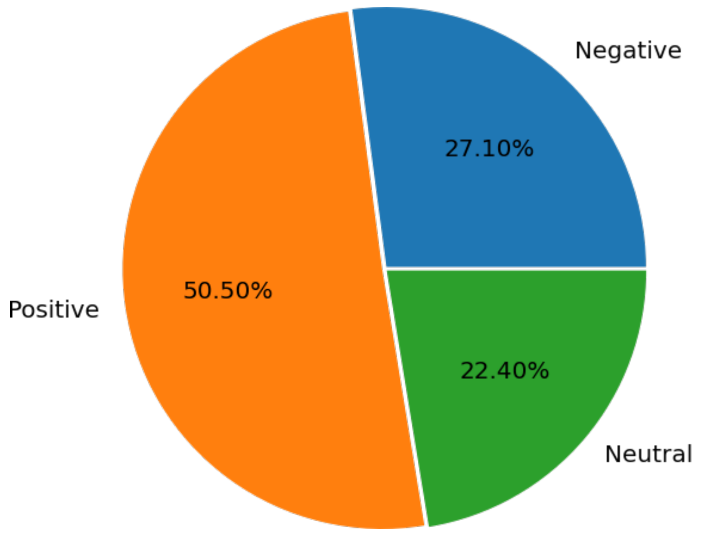


Figure 2. Sentiment polarity in the dataset

detection. Unlike broad labels of positive, negative, or neutral, emotion detection identifies specific feelings like *Joy*, *Anger*, *Love*, *Fear*, *Sadness* and *Surprise*.

4.3.2. Distribution of emotion categories

This was performed to understand the distribution, quality, and structure of the emotion-labeled dataset. Understanding the distribution of emotions in the dataset is crucial since class imbalance can introduce biases into the ML models [47]. By visualizing the distribution, one can take necessary measures to augment the data for under-represented categories or use techniques to address imbalances during model training. The graph presented in Figure 3 provides a visualization of the distribution of various emotion categories in the dataset. The bar chart shows the count of instances for each emotion category, while the pie

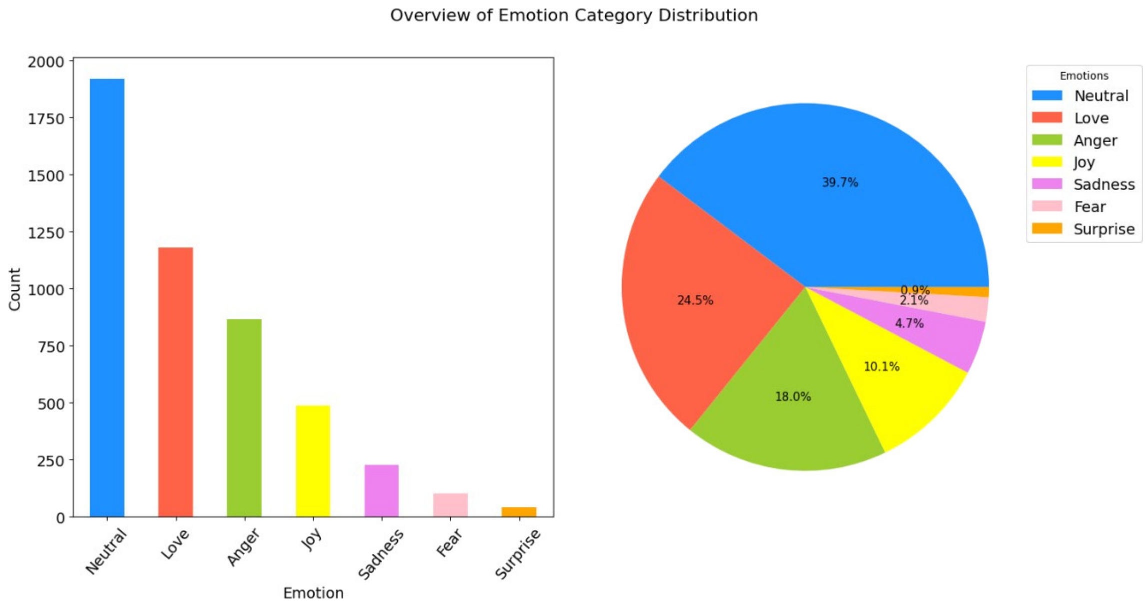


Figure 3. Overview of emotion category distribution

chart illustrates the proportion of each class. A quick analysis of this visualization reveals a pronounced imbalance among the different types of emotions. The dataset does not adequately represent certain emotion categories, namely *Joy*, *Sadness*, *Fear*, and *Surprise*.

4.4. Addressing the data imbalance

In ML, data imbalance pertains to the uneven distribution of classes within a dataset. It is a prevalent challenge, especially in classification tasks, where certain classes are significantly underrepresented compared to others. This skewed representation often leads to suboptimal model performance, as the algorithms tend to exhibit a bias towards the majority class, consequently neglecting the minority class [47]. To address this imbalance in our dataset, we considered various strategies and opted for under-sampling the majority class and apply text augmentation for the minority classes.

4.4.1. Under-sampling the majority class

Under-sampling involves decreasing the number of observations from the predominant class to achieve a more balanced class representation [48]. Specifically, we randomly selected 950 samples from the Neutral emotion category. We selected this number through experimentation. We notice that the algorithm is giving better results when the data points in the neutral category have a similar number of points as in the other categories. After the neutral category, the second highest number of data points were present in the “love” category, i.e., 945. Hence, we selected 950 samples for “neutral” category.

4.4.2. Text augmentation using a contextual word embedding with BERT

Another approach adopted in this study to balance the dataset is data augmentation. This involves creating new data by slightly altering existing samples, thereby artificially enlarging the dataset. Especially, text augmentation with word substitution was performed on the minority classes. Word substitution, an effective technique to augment textual data, refers to the process of replacing words in a text with other words while aiming to retain the overall meaning or intent of the original text [20]. Before the text augmentation, the dataset was split into training, validation, and testing sets in a stratified ratio of 80–10–10 using the scikit-learn library. Only the training dataset was enhanced through augmentation, ensuring that the validation and testing sets reflect real-world situations.

Leveraging the ContextualWordEmbsAug class from the nlpaug library, 94% of the Surprise, 44% of the Sadness and 91% of the Fear emotion category samples were randomly chosen and augmented using the bert-base-uncased model. To have a relatively balanced dataset and to avoid introducing noise in the dataset, each sample from the Surprise, Sadness, and Fear categories were augmented four, one and two times, respectively. This was performed to introduce variety in the increased dataset while still preserving the original meaning.

For each sample, randomly selected words were replaced by the nearest words derived from the embedding space provided by the BERT model. Unlike traditional word embeddings, which give every word a fixed vector representation regardless of its context in a sentence, contextual embeddings adjust word representations based on the surrounding words in a given sentence, making it suitable for text augmentation [49]. Tables 3, 4, and 5 show the distribution of the categories in the augmented training dataset, test dataset, and

Table 3. Distribution of the emotion categories in the training set, test set, and validation set for word substitution

	Emotion category						
	Love	Anger	Joy	Sadness	Fear	Surprise	Neutral
Number of samples (Training set)	945	694	390	262	232	162	760
Number of samples (Test set)	118	87	49	23	10	4	95
Number of samples (Validation set)	118	86	49	22	11	5	95

Table 4. Distribution of the emotion categories in the training set, test set, and validation set using back translation

	Emotion category						
	Love	Anger	Joy	Sadness	Fear	Surprise	Neutral
Number of samples (Training set)	945	694	390	262	157	66	760
Number of samples (Test set)	118	87	49	23	10	4	95
Number of samples (Validation set)	118	86	49	22	11	5	95

Table 5. Distribution of the emotion categories in the training set, test set, and validation set using easy data augmentation approach

	Emotion category						
	Love	Anger	Joy	Sadness	Fear	Surprise	Neutral
Number of samples (Training set)	945	694	390	262	232	162	760
Number of samples (Test set)	118	87	49	23	10	4	95
Number of samples (Validation set)	118	86	49	22	11	5	95

validation dataset using word substitution, back translation, and easy data augmentation technique, respectively. For the Word substitution and EDA methods, the selected samples in the Surprise, Fear, and Sadness categories were augmented 4, 2, and 1 times, respectively. However, the actual number of samples in the EDA-augmented training set may be lower for certain folders. This discrepancy occurred because the code used was unable to process some rows. For the Back translation approach, the selected samples were augmented only once to prevent duplicates in the augmented training set.

4.5. Emotion classification algorithms

We employed RF, LSTM, and BERT to develop the emotion classification algorithms. The section below presents the description and architecture of the three models.

4.5.1. LSTM

We chose LSTM for this emotion classification task because it is particularly adept at processing sequence data and learning long-term dependencies, which is often inherent in language-based tasks. Moreover, LSTM excels at detecting complex patterns within natural language (NL), patterns that other models might not [50], and has proved reliable for NL understanding tasks like text classification [51, 52] and sentiment analysis [53]. LSTM, a type of Recurrent Neural Network (RNN), was introduced [54] to overcome the vanishing and exploding gradients problem that RNNs suffer from. LSTM networks have four main components: the input gate, forget gate, output gate, and memory cell. The memory cell holds relevant information, with the gates managing data intake, retention, and output [54]. This architecture enables LSTMs to manage long sequences efficiently, making them particularly effective for text classification and capturing intricate human emotions in text [55].

4.5.2. BERT transformer model

Researchers have started exploring the use of transformer models for sentiment analysis and emotion classification tasks [14, 56]. Thanks to their architecture and pre-training on extensive corpora, they are adept at detecting subtle nuances in textual data. Developed by researchers at Google in 2018 [57], BERT represents a significant advancement in the NLP domain known for its bidirectional understanding of language and has paved the way for models RoBERTa, FlauBERT. The transformer architecture, presented by Vaswani et al. [58], utilizes self-attention mechanisms for contextual understanding, processing words concurrently for efficiency. BERT, building on this, discerns context by masking and predicting certain input tokens, thereby enhancing linguistic representations. BERT employs token, segment, and positional embeddings for input representation. It uses WordPiece tokenization to manage out-of-vocabulary words and maintains input data sequence by integrating these embeddings [57]. Initially, BERT was offered in two versions:
– BERT-BASE with 12 layers, 768 hidden sizes, 12 attention heads, and 110 million parameters.
– BERT-LARGE with 24 layers, 1024 hidden sizes, 16 attention heads, and 340 million parameters.

BERT's pre-training involved the Masked Language Model (MLM) and Next Sentence Prediction (NSP) tasks. In MLM, BERT predicts concealed input tokens, while in NSP, it identifies sentence sequences, aiding question-answering tasks [57].

4.5.3. Random Forest Classifier

Many researchers have used RFC for text classification tasks [59]. Specifically, studies like the one by [15] have shown that Random Forest is one of the reliable models for detecting sentiment in posts on Q&A websites for software developers. Furthermore, we selected RFC because of its capability to train on small datasets, as is the case in this study. Introduced adequately by Breiman in [60], RFC is an ensemble learning method that creates numerous decision trees during its training phase and merges their results for more accurate and reliable predictions [60]. Each tree makes its own classification decision based on the input data. The final class determination for a given input is achieved by taking a majority vote from the classifications of all individual trees.

5. Evaluation metrics

After developing a machine learning model, it is essential to use evaluation metrics to determine its performance on previously unseen test data. We selected popular metrics used for the classification task.

5.1. Precision, recall, and *F*-score

While the balanced accuracy score can provide an overview of the model performance, it does not show the performance of each class in the dataset. Precision provides insight into a model's correct predictions for each class. For class i, precision is:

$$Precision_i = \frac{TP_i}{TP_i + FP_i}$$

where TP_i is the number of correctly predicted instances of class i, and FP_i is instances wrongly predicted as class i. Recall evaluates the model's ability to identify all possible positive instances within the dataset [61]. It is determined by the following formula, where FN_i denotes instances of class i wrongly predicted as another class.

$$Recall_i = \frac{TP_i}{TP_i + FN_i}$$

F_1-score provides a harmonic mean of the two metrics, and is computed as follows:

$$F_1\text{-}score_i = \frac{2 * Precision_i * Recall_i}{Precision_i + Recall_i}$$

6. Results

This section provides an overview of the results from various experiments conducted on the Stack Overflow dataset. The investigation comprises three primary research queries addressed in the paragraphs below. We made all the dataset and source code publicly available for replication: https://drive.google.com/drive/folders/1qXyLx9OhpHVcXLMT sTYdjhxhV-t6G54j.

6.1. RQ1: Which classification model performs best among LSTM, BERT, CodeBERT, and RFC?

Motivation. With the rapid advancements in ML and NLP, many models have been proposed to solve classification problems in text data. Among these, the RFC, Support Vector Machines (SVM) have been commonly employed. These algorithms have demonstrated their capacity to yield reliable classification results across diverse contexts. In recent years, researchers have used more sophisticated tools, such as LSTM [52], BERT [56], CodeBERT [62], and RFC [59]. In this RQ, we compare the performance of LSTM, BERT, CodeBERT, and RFC in classifying emotions within the Stack Overflow dataset.

Approach. In this part, we give a detailed description of the parameters used for all the algorithms.

LSTM. After the preprocessing techniques, the LSTM model was implemented using the `keras` and `tensorflow` libraries. Textual data was tokenized and normalized to sequences of integers with a uniform length of 195 and the labels were one-hot encoded. The LSTM model includes an embedding layer converting input to a 128-dimensional vector and a two-layered LSTM: the first layer with 128 neurons (and 0.2 dropout rate) and the second with 64 neurons. The dropout parameter helps to prevent overfitting, whereas the 2-layered

LSTM structure allows the model to capture more complex patterns. The model's output layer has 7 units with softmax activation for multi-class classification. It is compiled using the `categorical_crossentropy` loss function with the `adam` optimizer. The model is set to train for a maximum of 20 epochs using a batch size of 64 and the `class_weight` parameter which handles the class imbalance. However, the early stopping criteria could terminate it prematurely. Regularization techniques like `EarlyStopping` and `ModelCheckpoint` were used to monitor the validation loss and ensure the model stops training if there is no improvement after 7 epochs. Several optimization strategies were employed to enhance the model's performance and prevent overfitting. We explored a variety of hyperparameters (dropout rate, number of neurons in the layers, number of epochs) to fine-tune the model. Different hyperparameter combinations were tested to determine which gave the highest results.

BERT. The emotion classification model, built with BERT, utilized the same augmented training, validation, and testing datasets as the LSTM neural network. It was developed using the `bert-base-uncased` pre-trained model and the `datasets`, `scikit-learn`, `torch`, and `transformers` libraries; and trained on a Graphics Processing Unit (GPU). The model was trained for 4 epochs – to avoid overfitting while still allowing it to detect the emotion contained in the comment – and enhanced with the Adam with Weight Decay (`adamw_torch`) optimizer, which is a renowned gradient descent optimization algorithm for transformers models. Before the tokenization, the data frames are converted to HuggingFace's Dataset format, and subsequently mapped to a DatasetDict (Dataset dictionary) object. Each text entry is tokenized, padded, and truncated using the AutoTokenizer function of the bert-base-uncased model. Tokenization is crucial as it converts the input data into a format the model can understand. Meanwhile, padding and truncating ensure that all input sequences have the same length, a requirement for batch processing in neural networks. Training parameters such as the number of epochs (4), learning rate (2×10^{-5}), batch size (16), optimizer `(adamw_torch)`, and others are set using the `TrainingArguments` class. These parameters play a pivotal role in guiding the model's learning behavior. The number of epochs dictates how many times the model reviews the entire dataset, the learning rate determines the step size when updating weights, and the batch size indicates the number of data points processed simultaneously. The `Trainer` class from the Transformers library is used to train the model on the training dataset while validating it on the validation dataset. The HuggingFace Trainer class simplifies the process of training machine learning models by encapsulating the necessary training tasks, making it both efficient and user-friendly.

CodeBERT. It is a bimodal pre-trained model developed using transformer-based neural architecture. It is designed for NL-PL applications such as natural language code search and documentation generation. The model is trained using the hybrid objective function. This uses both bimodal and uni-modal data for model training. The bimodal data provides the input token and the uni-modal data is used for learning better generators. As recommended by the authors in [62][4], we used the `RobertaTokenizer` for tokenizing our input data. Then, we fine-tuned the `microsoft/codebert-base` model on our dataset utilizing the same training parameters as those used for the BERT model we previously developed. This approach was taken to enable a direct performance comparison between the two transformer-based models. By keeping the parameters consistent, we ensured that any differences in performance could be attributed to the models themselves, rather than variations in the training process.

[4]https://github.com/microsoft/CodeBERT

Table 6. Performance of the models

	EmoClassLSTM-Original			EmoClassBERT-Original			EmoClassCodeBERT-Original			EmoClassRFC-Original		
Emotion	Precision	Recall	F_1-score	Precision	Recall	F_1-score	Precision	Recall	F_1-score	Precision	Recall	F_1-score
Love	0.72	0.70	0.71	0.76	0.84	**0.80**	0.81	0.79	**0.80**	0.72	0.78	0.75
Joy	0.35	0.39	0.37	0.50	0.45	0.47	0.54	0.65	**0.59**	0.53	0.35	0.42
Surprise	0.07	0.50	**0.12**	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Anger	0.63	0.22	0.32	0.75	0.75	**0.75**	0.75	0.68	0.71	0.72	0.72	0.72
Sadness	0.38	0.13	0.19	0.68	0.65	**0.67**	0.59	0.70	0.64	0.68	0.57	0.62
Fear	0.18	0.20	0.19	0.58	0.70	0.64	0.70	0.70	**0.70**	0.00	0.00	0.00
Neutral	0.40	0.58	0.47	0.79	0.75	**0.77**	0.76	0.77	0.76	0.59	0.73	0.65

RFC. The `scikit-learn` library was used to implement the RFC model using the same training, validation, and testing sets as the previous models. The training set was utilized for training the model, while the validation set was employed for fine-tuning the hyperparameters and determining the best combination for the model. Finally, the testing set was used to evaluate the model's performance. For feature extraction, the code uses the TF-IDF method to convert the text data into numerical features. While the Term Frequency computes the frequency of words in a document, the Inverse Document Frequency calculates the importance of a word. Together, the TF-IDF method captures words' significance in the dataset while diminishing the weight of frequently occurring but potentially uninformative words [63]. Subsequently, the RFC is instantiated and trained on the TF-IDF processed training data using 300 trees `(n_estimators=300)`. The decision to utilize 300 trees was made to strike a harmonious balance. With too few trees, the model might not capture all the nuances in the data. Conversely, an excessive number could lead to computational inefficiencies without significantly improving performance. Furthermore, we used the `GridSearchCV` technique to obtain the optimal set of parameters for the vectorizer and RFC model. Finally, the performance of the model is evaluated on the test set.

Using the above parameters we created the following models:

– **EmoClassLSTM-Original.** an LSTM model trained with the parameters described above.
– **EmoClassBERT-Original.** a BERT model trained with the parameters described above.
– **EmoClassCodeBERT-Original.** a CodeBERT model trained with the parameters described above.
– **EmoClassRFC-Original.** an RFC model trained with the parameters described above.

Results. To gain a more comprehensive insight into the model's performance, precision, recall, and the F_1-score were computed for each emotion category. These metrics offer a nuanced understanding of how well the model identifies and classifies each emotion. Table 6 details the results of each model in identifying a specific emotion category. For example, for emotion category *Joy*, EmoClassBERT-Original achieved an F_1-score of 59%, while EmoClassBERT-Original, EmoClassLSTM-Original, and EmoClassRFC-Original gave an F_1-score of 47%, 37%, and 42%, respectively. Based on the obtained results, BERT outperformed the other models in detecting most emotion categories, while EmoClassLSTM-Original and EmoClassCodeERT-Original achieved better performance for *Surprise* and *Fear*, respectively. BERT's superior performance can be attributed to its bidirectional architecture, which enables it to grasp both past and future context, and its extensive pre-training on vast text corpora allows it to understand language nuances deeply. On the other hand, while the LSTM produced significant results compared to other emotion detection tools in the SE domain, as detailed in [2], its unidirectional processing of sequences and the limited dataset, could be factors contributing to its average inferior performance compared to BERT and RFC.

Given these results, BERT emerges as a more optimal choice for tasks that require a profound understanding of context, especially in complex datasets like Stack Overflow comments. Nevertheless, the low score for the Surprise from all the models could be due to insufficient samples and the complexity in detecting the Surprise emotion, as noted by [14].

6.2. RQ2: An initial investigation: Can data augmentation improve the model's performance?

Motivation. Training machine learning models to decipher the complex world of human emotions requires vast amounts of data, particularly labelled data that indicates which emotion is present in a given comment. In the context of Stack Overflow, this becomes even more intricate given the specific lexicon used. Given these challenges, procuring an adequately representative dataset for emotion classification on Stack Overflow is not just resource intensive but also requires extensive domain-specific knowledge to annotate the data accurately. This complexity, combined with the need for large-scale data to train robust models, leads to an intriguing proposition: Could data augmentation be utilized to synthetically expand and diversify the dataset, instead of solely relying on manual data collection and annotation?

The choice to explore Random Forest Classifier (RFC) and Long Short-Term Memory (LSTM) models alongside BERT and CodeBERT was driven by a desire to evaluate various approaches and assess their performance comprehensively. While BERT indeed demonstrated superior performance, considering alternative models allowed us to provide a more nuanced understanding of the dataset and its characteristics. The motivation for incorporating RFC and LSTM models aimed to explore how these models would handle the enriched dataset. This approach provides a broader perspective on the robustness and adaptability of different models to variations in data volume and complexity. Hence, we tested 16 combinations of various classifiers for exhaustive testing.

Table 7. Examples of data augmented using back translation from the dataset

S No.	Approach	Comment
1.	Original	million unique visitors per hour? Wow! Is this Experts Exchange or some pr n site
	Translation	millions **of** unique visitors per hour wow is this exchange **of experts** or **another site** pr n
2	Original	wow would have expected a quick answer on this well found my own answer ....
	Translation	wow would have **waited for** a quick answer on this **property** found my own answer...

Approach. Data augmentation techniques were employed to increase the diversity of the dataset. Data augmentation is an established strategy in machine learning, which

Table 8. Examples of data augmented using word substituted from the dataset

S No.	Approach	Comment
1.	Original	a unit test should do the same thing every time that it runs otherwise you may run into a situation where the unit test only fails occasionally…
	Substitution	a unit test should do the same **then** every time that it runs otherwise you may run into another situation where the unit test **still** fails occasionally…
2.	Original	I m very sorry about my horrible English only for this example I use radio button…
	Substitution	**we** m very sorry concerning my **short** english only **at** this example I use **one** button…

can significantly enhance the quality and versatility of datasets without the need for additional data collection [2]. The dataset was divided into training, validation, and testing sets using a ratio of 80–10–10. To ensure that the validation and testing sets mirrored real-world applications, only the training set was augmented. The text augmentation methods evaluated and implemented for the underrepresented categories include:

– **Back Translation.** The BackTranslationAug class was employed for the translation. This technique involves translating a text into a secondary language (in our case, French was chosen due to its rich linguistic structure) and then reverting it to its original language, English. This often results in texts that maintain their core sentiment but are structurally or lexically varied. Table 7 shows some examples of the original and augmented datasets.
– **Word substitution.** To introduce lexical diversity, words were replaced with their closest synonyms in the contextual embedding space. While the overall sentiment remains intact, this technique ensures that the model is not biased towards specific wordings. The ContextualWordEmbsAug class was employed with the substitute action parameter for this augmentation process. Table 8 shows some examples of the original and augmented datasets.
– **Easy data augmentation.** Easy data augmentation for a given sentence performs one of the four operations randomly, i.e., synonym replacement, random insertion, swap, and random deletion.

Leveraging the nlpaug library, 94% of the Surprise, 44% of the Sadness and 91% of the Fear emotion category samples were randomly chosen and augmented. For the word substitution augmentation technique, each sample from the Surprise, Sadness and Fear categories was augmented four, one, and two times, respectively. This augmentation was done to prevent the introduction of duplicate entries in the training data and to maintain a balanced distribution across the various emotion categories. Through trial runs, we observed that excessively augmenting the dataset did not contribute to increased diversity. For the back translation approach, samples were translated only once to prevent introducing duplicates in the augmented data. These strategies were adopted to introduce variety in the augmented data while still preserving the original meaning.

For every augmentation method employed, the enhanced dataset was merged with the original training data. Once combined, this consolidated data was then provided to the machine learning model for training. Three different versions of LSTM, BERT, CodeBERT, and RFC models were developed utilizing the original and augmented training sets. To ensure consistency and optimal learning, each LSTM model was trained for a duration of 20 epochs using a batch size of 64 with regularization techniques. On the other hand, each BERT model was trained for 4 epochs with a batch size of 16 and a learning rate set

Table 9. Details of the emotion classification models implemented

Model	Augmentation technique	Reference
LSTM	Word Substitution	EmoClassLSTM-Substitution (EmoClassLSTM-S)
	Back Translation	EmoClassLSTM-Translation (EmoClassLSTM-T)
	Easy data augmentation	EmoClassLSTM-EDA (EmoClassLSTM-E)
	None	EmoClassLSTM-Original (EmoClassLSTM-O)
BERT	Word Substitution	EmoClassBERT-Substitution (EmoClassBERT-S)
	Back Translation	EmoClassBERT-Translation (EmoClassBERT-T)
	Easy data augmentation	EmoClassBERT-EDA (EmoClassBERT-E)
	None	EmoClassBERT-Original (EmoClassBERT-O)
CodeBERT	Word Substitution	EmoClassCodeBERT-Substitution (EmoClassCodeBERT-S)
	Back Translation	EmoClassCodeBERT-Translation (EmoClassCodeBERT-T)
	Easy data augmentation	EmoClassCodeBERT-EDA (EmoClassCodeBERT-E)
	None	EmoClassCodeBERT-Original (EmoClassCodeBERT-O)
RFC	Word Substitution	EmoClassRFC-Substitution (EmoClassRFC-S)
	Back Translation	EmoClassRFC-Translation(EmoClassRFC-T)
	Easy data augmentation	EmoClassRFC-EDA (EmoClassRFC-E)
	None	EmoClassRFC-Original (EmoClassRFC-O)

at 2×10^{-5}, striking a balance between speed and prediction performance, and each RFC model utilized the same hyperparameters as described earlier. Table 9 provides a summary of the models implemented.

Results. The performance of emotion detection tools for each specific emotion is outlined in Table 10 and Table 11. Table 10 shows the performance of all the models with and without augmentation whereas Table 11 shows the average F_1-score for all the models. Specifically, for the BERT models, there is not much difference between precision and recall, suggesting that these models are equally adept at identifying true positive cases (precision) as they are at capturing the total positive instances (recall). This balance is crucial in emotion detection, as it means that the model is accurate in its predictions and minimizes the risk of missing out on instances where a specific emotion is present. For the CodeBERT model, the models trained on the augmented dataset outperformed the models trained on the original dataset in most cases. However, the results for the LSTM model emphasize the importance of having sufficient training data or training it on a more balanced dataset. In most cases, the RFC models performed better than the LSTM models, indicating that RFC is more suitable for smaller datasets. The highest F_1-scores, highlighted in bold, show that:

– EmoClassBERT-Substitution performed best for *Love*;
– EmoClassCodeBERT-Original performed best for *Joy*;
– EmoClassBERT-Translation and EmoClassCodeBERT-Translation performed best for *Anger*;
– EmoClassBERT-Original performed best for *Sadness*;
– EmoClassBERT-Translation, EmoClassRFC-Substitution, and EmoClassRFC-EDA performed best for *Surprise*;
– EmoClassCodeBERT-Translation give the best F_1-score for (*Neutral*).

From these findings, several implications can be derived. The efficacy of word substitution in introducing variance through word substitution makes it suitable for a wide range of emotions. Table 11 shows that substitution-based models outperformed the other models. EmoClassBERT-substitution gives the highest F_1-score of 63%. The substitution method shows a considerable improvement as compared to the original models, for example, EmoClassLSTM-Substitution, EmoClassBERT-Substitution, EmoClassCodeBERT-Substitution, and EmoClassRFC-Substitution models show improvements of 13%, 5%, 5%, and 10% as compared to EmoClassLSTM-Original, EmoClassBERT-Original, and EmoClass-RFC-Original, respectively, in average F_1-score. On the other hand, the nuanced linguistic changes introduced by back translation make it effective for emotions like Surprise and Anger, as was demonstrated. The reduced F_1-score for the *Surprise* category might be due to its smaller sample size. Additionally, the nature of Surprise as an emotion is inherently ambiguous [9], often intertwining with both positive and negative emotions, further complicating its classification [14]. Nevertheless, EmoClassRFC-Substitution, EmoClassRFC-EDA, and EmoClassBERT-Translation outperformed other models in detecting the *Surprise* category with an F_1-score of 0.33. While data augmentation techniques showed effectiveness in various emotional categories, they did not consistently outperform the original model. Interestingly, the model performed best at detecting the emotion *Joy* when trained on non-augmented data. However, models trained on the original dataset on average underperformed compared to models trained on the augmented dataset, except the CodeBERT models (refer to Table 11). This underscores that augmentation's effectiveness can vary depending on the particular emotion under study.

Table 10. Performance of all variants of LSTM, BERT, CodeBERT and RFC

Emotion	Base model	Model	Precision	Recall	F_1-score
Love	LSTM	EmoClassLSTM-S	0.79	0.55	0.65
		EmoClassLSTM-T	0.84	0.39	0.53
		EmoClassLSTM-E	0.75	0.70	0.73
		EmoClassLSTM-O	0.72	0.70	0.71
	BERT	EmoClassBERT-S	0.81	0.81	0.81
		EmoClassBERT-T	0.82	0.77	0.79
		EmoClassBERT-E	0.81	0.85	**0.83**
		EmoClassBERT-O	0.76	0.84	0.80
	CodeBERT	EmoClassCodeBERT-S	0.79	0.85	0.82
		EmoClassCodeBERT-T	0.79	0.78	0.79
		EmoClassCodeBERT-E	0.82	0.79	0.80
		EmoClassCodeBERT-O	0.81	0.79	0.80
	RFC	EmoClassRFC-S	0.75	0.80	0.77
		EmoClassRFC-T	0.72	0.79	0.75
		EmoClassRFC-E	0.74	0.80	0.77
		EmoClassRFC-O	0.72	0.78	0.75
Joy	LSTM	EmoClassLSTM-S	0.30	0.55	0.39
		EmoClassLSTM-T	0.24	0.39	0.31
		EmoClassLSTM-E	0.42	0.41	0.41
		EmoClassLSTM-O	0.35	0.39	0.37
	BERT	EmoClassBERT-S	0.50	0.49	0.49
		EmoClassBERT-T	0.43	0.53	0.47
		EmoClassBERT-E	0.52	0.49	0.51
		EmoClassBERT-O	0.50	0.45	0.47
	CodeBERT	EmoClassCodeBERT-S	0.57	0.49	0.53
		EmoClassCodeBERT-T	0.49	0.55	0.52
		EmoClassCodeBERT-E	0.49	0.59	0.54
		EmoClassCodeBERT-O	0.54	0.65	**0.59**

Table 10 continued

Emotion	Base model	Model	Precision	Recall	F_1-score
Joy	RFC	EmoClassRFC-S	0.53	0.33	0.41
		EmoClassRFC-T	0.55	0.37	0.44
		EmoClassRFC-E	0.51	0.39	0.44
		EmoClassRFC-O	0.53	0.35	0.42
Surprise	LSTM	EmoClassLSTM-S	0.20	0.25	0.22
		EmoClassLSTM-T	0.20	0.43	0.22
		EmoClassLSTM-E	0.00	0.00	0.00
		EmoClassLSTM-O	0.07	0.50	0.12
	BERT	EmoClassBERT-S	0.33	0.25	0.29
		EmoClassBERT-T	0.50	0.25	**0.33**
		EmoClasBERT-E	0.00	0.00	0.00
		EmoClassBERT-O	0.00	0.00	0.00
	CodeBERT	EmoClassCodeBERT-S	0.33	0.25	0.29
		EmoClassCodeBERT-T	0.00	0.00	0.00
		EmoClassCodeBERT-E	0.00	0.00	0.00
		EmoClassCodeBERT-O	0.00	0.00	0.00
	RFC	EmoClassRFC-S	0.50	0.25	**0.33**
		EmoClassRFC-T	0.00	0.00	0.00
		EmoClassRFC-E	0.50	0.25	**0.33**
		EmoClassRFC-O	0.00	0.00	0.00
Anger	LSTM	EmoClassLSTM-S	0.57	0.63	0.60
		EmoClassLSTM-T	0.69	0.51	0.58
		EmoClassLSTM-E	0.66	0.54	0.59
		EmoClassLSTM-O	0.63	0.22	0.32
	BERT	EmoClassBERT-S	0.73	0.76	0.75
		EmoClassBERT-T	0.74	0.78	**0.76**
		EmoClassBERT-E	0.73	0.74	0.73
		EmoClassBERT-O	0.75	0.75	0.75
	CodeBERT	EmoClassCodeBERT-S	0.71	0.74	0.72
		EmoClassCodeBERT-T	0.76	0.76	**0.76**
		EmoClassCodeBERT-E	0.74	0.75	0.74
		EmoClassCodeBERT-O	0.75	0.68	0.71
	RFC	EmoClassRFC-S	0.72	0.71	0.72
		EmoClassRFC-T	0.76	0.74	0.75
		EmoClassRFC-E	0.73	0.72	0.73
		EmoClassRFC-O	0.72	0.72	0.72
Sadness	LSTM	EmoClassLSTM-S	0.52	0.52	0.52
		EmoClassLSTM-T	0.27	0.43	0.33
		EmoClassLSTM-E	0.57	0.0.57	0.57
		EmoClassLSTM-O	0.38	0.13	0.19
	BERT	EmoClassBERT-S	0.72	0.57	0.63
		EmoClassBERT-T	0.67	0.52	0.59
		EmoClassBERT-E	0.64	0.61	0.62
		EmoClassBERT-O	0.68	0.65	0.67
	CodeBERT	EmoClassCodeBERT-S	0.68	0.65	0.67
		EmoClassCodeBERT-T	0.67	0.61	0.64
		EmoClassCodeBERT-E	0.67	0.61	0.64
		EmoClassCodeBERT-O	0.59	0.70	0.64

Table 10 continued

Emotion	Base model	Model	Precision	Recall	F_1-score
Sadness	RFC	EmoClassRFC-S	0.67	0.52	0.59
		EmoClassRFC-T	0.65	0.57	0.60
		EmoClassRFC-E	0.71	0.65	**0.68**
		EmoClassRFC-O	0.68	0.57	0.62
Fear	LSTM	EmoClassLSTM-S	0.50	0.40	0.44
		EmoClassLSTM-T	0.40	0.20	0.27
		EmoClassLSTM-E	0.17	0.30	0.21
		EmoClassLSTM-O	0.18	0.20	0.19
	BERT	EmoClassBERT-S	0.70	0.70	0.70
		EmoClassBERT-T	0.54	0.70	0.61
		EmoClassBERT-E	0.67	0.80	0.73
		EmoClassBERT-O	0.58	0.70	0.64
	CodeBERT	EmoClassCodeBERT-S	0.73	0.80	**0.76**
		EmoClassCodeBERT-T	0.53	0.80	0.64
		EmoClassCodeBERT-E	0.64	0.70	0.67
		EmoClassCodeBERT-O	0.70	0.70	0.70
	RFC	EmoClassRFC-S	0.33	0.40	0.36
		EmoClassRFC-T	0.10	0.10	0.10
		EmoClassRFC-E	0.00	0.00	0.00
		EmoClassRFC-O	0.00	0.00	0.00
Neutral	LSTM	EmoClassLSTM-S	0.67	0.57	0.61
		EmoClassLSTM-T	0.50	0.69	0.58
		EmoClassLSTM-E	0.58	0.67	0.62
		EmoClassLSTM-O	0.40	0.58	0.47
	BERT	EmoClassBERT-S	0.72	0.75	0.74
		EmoClassBERT-T	0.80	0.75	0.77
		EmoClassBERT-E	0.74	0.72	0.73
		EmoClassBERT-O	0.79	0.75	0.77
	CodeBERT	EmoClassCodeBERT-S	0.78	0.75	0.76
		EmoClassCodeBERT-T	0.80	0.76	**0.78**
		EmoClassCodeBERT-E	0.78	0.73	0.75
		EmoClassCodeBERT-O	0.76	0.77	0.76
	RFC	EmoClassRFC-S	0.62	0.74	0.68
		EmoClassRFC-T	0.60	0.68	0.64
		EmoClassRFC-E	0.62	0.73	0.67
		EmoClassRFC-O	0.59	0.73	0.65

In summary, a nuanced understanding of the role of data augmentation in emotion classification was provided by this study. Significant enhancements in model performance can be achieved through tailored data augmentation. However, the importance of a judicious evaluation based on the emotion in focus and the augmentation technique being employed was emphasized. On average, the substitution method gives the highest F_1-score. While word substitution performed best on average, none of the augmentation methods was identified as a one-size-fits-all solution.

Table 11. Average performance of EmoClass classifiers

Model name	Avg. F_1-score	Avg improvement (in %) as compared to the original models
EmoClassLSTM-Original	0.36	–
EmoClassLSTM-Translation	0.39	3%
EmoClassLSTM-Substitution	**0.49**	13 %
EmoClassLSTM-EDA	0.44	8%
EmoClassBERT-Original	0.58	–
EmoClassBERT-Translation	0.62	4%
EmoClassBERT-Substitution	**0.63**	5%
EmoClassBERT-EDA	0.59	1%
EmoClassCodeBERT-Original	0.6	–
EmoClassCodeBERT-Translation	0.59	−1%
EmoClassCodeBERT-Substitution	**0.65**	**5%**
EmoClassCodeBERT-EDA	0.59	−1%
EmoClassRFC-Original	0.45	–
EmoClassRFC-Translation	0.47	2%
EmoClassRFC-Substitution	**0.55**	10%
EmoClassRFC-EDA	0.51	6%

6.3. RQ3: How do EmoClassLSTM, EmoClassBERT, EmoClassCodeBERT, and EmoClassRFC compare to existing tools?

Motivation. Researchers have harnessed ML algorithms for sentiment analysis or emotion classification tasks. Techniques such as SVM (as discussed by Calefato et al. [40]) and RFC (highlighted by Murgia et al. [20] have been widely employed. The prevalent strategy involves developing One-vs-All emotion classifiers. This approach involves developing distinct binary classifiers, each dedicated to one of the six basic emotions. However, the efficacy of this approach has been challenged. Bleyl and Buxton, in their [14] study, underscored the superior effectiveness of multi-label classification tools when juxtaposed against the One-vs-All methodology. Their findings suggest that the multi-label tool provides a better performance. The current research endeavors to delve deeper into this domain by examining the proficiency of the three tools: EmoClassLSTM, EmoClassBERT, and EmoClassRFC. The objective is to critically assess their performance against existing emotion classification tools developed for the SE domain.

Approach. In our endeavor to benchmark the performance of our models against existing tools, a pivotal step was the selection of a consistent dataset for a fair evaluation. Consequently, the dataset employed in the studies of Bleyl and Buxton [14] and Calefato et al. [40] was selected. We compare our results with the Multi-label BERT model and EmoTxt detailed in [14]. **EmoTxt** is an emotion classification tool implemented in a supervised learning method using the Support Vector Machines One-vs-All approach, where a binary classifier was developed for each emotion category. **Multi-label SO BERT** is a fine-tuned version of the BERT model, created by incorporating technical texts into its tokenizer and utilizing augmented data during the model training process. We compare the state-of-the-art methods with our EmoClassLSTM-Substitution, EmoClassBERT-Substitution, EmoClassCodeBERT-Substitution, and EmoClassRFC-Substitution models (as they gave the best results, refer to RQ2 in Section 6.2 for more details).

Table 12. EmoClassLSTM, EmoClassBERT, EmoClassCodeBERT and EmoClassRFC vs. existing tools built on the same dataset

Emotion	EmoClassLSTM -Substitution F_1	EmoClassBERT -Substitution F_1	EmoClassCodeBERT -Substitution F_1	EmoClassRFC -Substitution F_1	Multi-label SO BERT F_1	EmoTxt F_1
Love	68%	81%	82%	77%	**84%**	69%
Joy	31%	49%	53%	41%	**56%**	38%
Surprise	15%	29%	29%	**33%**	26%	23%
Anger	59%	75%	72%	72%	**80%**	68%
Sadness	43%	63%	**67%**	59%	60%	52%
Fear	12%	70%	**76%**	36%	59%	6%
Average	38.00%	61.17%	**63.17%**	53.00%	60.80%	42.00%

Results. Table 12 presents the performance of EmoClassLSTM-Substitution, EmoClass-BERT-Substitution, EmoClassCodeBERT-Substitution, and EmoClassRFC-Substitution against that of the multi-label BERT model and EmoTxt reported in [14]. In Table 12, the best-performing model for each specific emotion is shown in bold. Among the models compared, Multi-label SO BERT demonstrates notable proficiency, especially in identifying the emotions of *Love*, *Joy*, and *Anger*. On the other hand, EmoClassCodeBERT-Substitution stands out when it comes to categorizing the emotions of *Sadness* and *Fear*, while EmoClassRFC-Substitution outperformed other models in detecting *Surprise* emotions in the text. On average EmoClassCdeoBERT-Substitution performed the best and gave the highest F_1-score of 63.17%. This model outperformed the Multi-label SO BERT and Emotxt by 2.37% and 21.17%, respectively.

This distinction in performance across different emotions emphasizes the importance of selecting the right model based on the specific needs of an emotion analysis task. The lower F_1-score of the *Surprise* emotion could be attributed to its low sample size and the difficulty in detecting it as reported in [14]. Overall, EmoClassCodeBERT-Substitution emerged as the top-performing emotion classification tool built with the dataset with an average of 63.17%.

6.4. RQ 4: How does algorithm randomness affect the performance of the proposed models?

Motivation. Performance evaluation of a model on only one dataset does not provide a clear indication of whether the results obtained are statistically significant or not. Hence, to address this issue, in this RQ, we performed an in-depth evaluation of the various models proposed in this paper.

Approach. We run the various data augmentation techniques for 100 iterations and generate 100 training, testing, and validation datasets [64]. We evaluated each model on these 100 datasets and reported the median values of the performance metric (i.e., F_1-score). We computed the Wilcoxon signed-rank test to compare the performances of different models. Additionally, we computed Cliff's delta [65] to quantify the difference between the two distributions.

Table 13. Median F_1-score

Emotion	Base model	Model	Median F_1-score
Love	LSTM	EmoClassLSTM-S	0.70
		EmoClassLSTM-T	0.70
		EmoClassLSTM-E	0.69
		EmoClassLSTM-O	0.69
	BERT	EmoClassBERT-S	**0.95**
		EmoClassBERT-T	**0.95**
		EmoClassBERT-E	**0.95**
		EmoClassBERT-O	**0.95**
	CodeBERT	EmoClassCodeBERT-S	0.94
		EmoClassCodeBERT-T	0.94
		EmoClassCodeBERT-E	**0.95**
		EmoClassCodeBERT-O	**0.95**
	RFC	EmoClassRFC-S	0.76
		EmoClassRFC-T	0.76
		EmoClassRFC-E	0.76
		EmoClassRFC-O	0.76
Joy	LSTM	EmoClassLSTM-S	0.35
		EmoClassLSTM-T	0.35
		EmoClassLSTM-E	0.36
		EmoClassLSTM-O	0.35
	BERT	EmoClassBERT-S	0.87
		EmoClassBERT-T	0.88
		EmoClassBERT-E	**0.89**
		EmoClassBERT-O	0.88
	CodeBERT	EmoClassCodeBERT-S	0.86
		EmoClassCodeBERT-T	0.87
		EmoClassCodeBERT-E	**0.89**
		EmoClassCodeBERT-O	0.88
	RFC	EmoClassRFC-S	0.34
		EmoClassRFC-T	0.35
		EmoClassRFC-E	0.35
		EmoClassRFC-O	0.34
Surprise	LSTM	EmoClassLSTM-S	0.13
		EmoClassLSTM-T	0.16
		EmoClassLSTM-E	0.10
		EmoClassLSTM-O	0.12
	BERT	EmoClassBERT-S	0.73
		EmoClassBERT-T	**0.75**
		EmoClassBERT-E	**0.75**
		EmoClassBERT-O	**0.75**
	CodeBERT	EmoClassCodeBERT-S	0.73
		EmoClassCodeBERT-T	0.73
		EmoClassCodeBERT-E	**0.75**
		EmoClassCodeBERT-O	**0.75**
	RFC	EmoClassRFC-S	0.00
		EmoClassRFC-T	0.00
		EmoClassRFC-E	0.00
		EmoClassRFC-O	0.00

Table 13 continued

Emotion	Base model	Model	Median F_1-score
Anger	LSTM	EmoClassLSTM-S	0.56
		EmoClassLSTM-T	0.56
		EmoClassLSTM-E	0.56
		EmoClassLSTM-O	0.57
	BERT	EmoClassBERT-S	**0.97**
		EmoClassBERT-T	**0.97**
		EmoClassBERT-E	**0.97**
		EmoClassBERT-O	**0.97**
	CodeBERT	EmoClassCodeBERT-S	**0.97**
		EmoClassCodeBERT-T	**0.97**
		EmoClassCodeBERT-E	**0.97**
		EmoClassCodeBERT-O	**0.97**
	RFC	EmoClassRFC-S	0.72
		EmoClassRFC-T	0.71
		EmoClassRFC-E	0.71
		EmoClassRFC-O	0.72
Sadness	LSTM	EmoClassLSTM-S	0.29
		EmoClassLSTM-T	0.29
		EmoClassLSTM-E	0.28
		EmoClassLSTM-O	0.30
	BERT	EmoClassBERT-S	0.88
		EmoClassBERT-T	0.89
		EmoClassBERT-E	**0.90**
		EmoClassBERT-O	0.89
	CodeBERT	EmoClassCodeBERT-S	0.87
		EmoClassCodeBERT-T	0.88
		EmoClassCodeBERT-E	0.89
		EmoClassCodeBERT-O	**0.90**
	RFC	EmoClassRFC-S	0.42
		EmoClassRFC-T	0.43
		EmoClassRFC-E	0.45
		EmoClassRFC-O	0.46
Fear	LSTM	EmoClassLSTM-S	0.14
		EmoClassLSTM-T	0.17
		EmoClassLSTM-E	0.15
		EmoClassLSTM-O	0.16
	BERT	EmoClassBERT-S	0.89
		EmoClassBERT-T	**0.90**
		EmoClassBERT-E	**0.90**
		EmoClassBERT-O	**0.90**
	CodeBERT	EmoClassCodeBERT-S	0.88
		EmoClassCodeBERT-T	**0.90**
		EmoClassCodeBERT-E	**0.90**
		EmoClassCodeBERT-O	**0.90**
	RFC	EmoClassRFC-S	0.25
		EmoClassRFC-E	0.14
		EmoClassRFC-T	0.23
		EmoClassRFC-O	0.14

Table 13 continued

Emotion	Base model	Model	Median F_1-score
Neutral	LSTM	EmoClassLSTM-S	0.53
		EmoClassLSTM-T	0.53
		EmoClassLSTM-E	0.53
		EmoClassLSTM-O	0.55
	BERT	EmoClassBERT-S	**1.00**
		EmoClassBERT-T	**1.00**
		EmoClassBERT-E	**1.00**
		EmoClassBERT-O	**1.00**
	CodeBERT	EmoClassCodeBERT-S	**1.00**
		EmoClassCodeBERT-T	**1.00**
		EmoClassCodeBERT-E	**1.00**
		EmoClassCodeBERT-O	**1.00**
	RFC	EmoClassRFC-S	0.64
		EmoClassRFC-T	0.65
		EmoClassRFC-E	0.66
		EmoClassRFC-O	0.67

Table 14. Results of Wilcoxon Rank Test and Cliff's Delta: Value: V, Practical Difference: PD, value marked as **bold*** for Wilcoxon Rank test p-value indicate that p-value > 0.5.

Comparison between		Wilcoxon result		Cliff's Delta	
		Statistic	p-value	Value	PD
EmoClassBERT-S	EmoClassBERT-T	16 051.5	1.18212×10^{-24}	-0.06769	negligible
EmoClassBERT-S	EmoClassBERT-E	12 605.0	4.27656×10^{-34}	-0.09323	negligible
EmoClassBERT-S	EmoClassBERT-O	32 752.0	1.21951×10^{-12}	-0.06890	negligible
EmoClassBERT-S	EmoClassLSTM-S	32 752.0	1.21951×10^{-12}	-0.06890	negligible
EmoClassBERT-S	EmoClassLSTM-T	0.0	4.17153×10^{-116}	0.96236	large
EmoClassBERT-S	EmoClassLSTM-E	0.0	4.17161×10^{-116}	0.96305	large
EmoClassBERT-S	EmoClassLSTM-O	3.0	2.90243×10^{-116}	0.96131	large
EmoClassBERT-S	EmoClassRFC-S	339.0	1.15648×10^{-115}	0.92741	large
EmoClassBERT-S	EmoClassRFC-T	300.0	9.29481×10^{-116}	0.92580	large
EmoClassBERT-S	EmoClassRFC-E	376.0	1.37104×10^{-115}	0.92513	large
EmoClassBERT-S	EmoClassRFC-O	349.0	1.09128×10^{-115}	0.92444	large
EmoClassBERT-S	EmoClassCodeBERT-S	21 084.5	1.87545×10^{-18}	0.07175	negligible
EmoClassBERT-S	EmoClassCodeBERT-T	28 534.5	1.38822×10^{-3}	0.01398	negligible
EmoClassBERT-S	EmoClassCodeBERT-E	14 393.0	6.22144×10^{-27}	-0.06830	negligible
EmoClassBERT-S	EmoClassCodeBERT-O	11 412.0	2.17045×10^{-36}	-0.08747	negligible
EmoClassBERT-T	EmoClassBERT-E	21 473.0	9.12036×10^{-5}	-0.02774	negligible
EmoClassBERT-T	EmoClassBERT-O	38 388.0	5.11859×10^{-1} *	-0.00416	negligible
EmoClassBERT-T	EmoClassLSTM-S	0.0	4.17150×10^{-116}	0.97394	large
EmoClassBERT-T	EmoClassLSTM-T	3.0	2.90203×10^{-116}	0.97416	large
EmoClassBERT-T	EmoClassLSTM-E	1.0	2.87717×10^{-116}	0.97453	large
EmoClassBERT-T	EmoClassLSTM-O	0.0	2.86518×10^{-116}	0.97328	large
EmoClassBERT-T	EmoClassRFC-S	6.0	2.76076×10^{-116}	0.94493	large
EmoClassBERT-T	EmoClassRFC-T	10.0	2.64816×10^{-116}	0.94356	large
EmoClassBERT-T	EmoClassRFC-E	7.0	2.79445×10^{-116}	0.94290	large
EmoClassBERT-T	EmoClassRFC-O	6.0	2.46871×10^{-116}	0.94235	large
EmoClassBERT-T	EmoClassCodeBERT-S	12 402.5	5.43028×10^{-46}	0.13961	negligible
EmoClassBERT-T	EmoClassCodeBERT-T	10 832.5	1.23414×10^{-33}	0.08178	negligible
EmoClassBERT-T	EmoClassCodeBERT-E	27 205.5	6.64069×10^{-1} *	-0.00096	negligible
EmoClassBERT-T	EmoClassCodeBERT-O	22 043.0	3.59604×10^{-5}	-0.02094	negligible
EmoClassBERT-E	EmoClassBERT-O	24 588.0	2.74271×10^{-3}	0.02283	negligible

Table 14 continued

Comparison between		Wilcoxon result		Cliff's Delta	
		Statistic	p-value	Value	PD
EmoClassBERT-E	EmoClassLSTM-S	0.0	4.17134×10^{-116}	0.97392	large
EmoClassBERT-E	EmoClassLSTM-T	0.0	4.17135×10^{-116}	0.97416	large
EmoClassBERT-E	EmoClassLSTM-E	0.0	2.86496×10^{-116}	0.97437	large
EmoClassBERT-E	EmoClassLSTM-O	0.0	2.86523×10^{-116}	0.97334	large
EmoClassBERT-E	EmoClassRFC-S	3.0	2.71017×10^{-116}	0.94491	large
EmoClassBERT-E	EmoClassRFC-T	5.0	2.56950×10^{-116}	0.94353	large
EmoClassBERT-E	EmoClassRFC-E	4.0	2.75170×10^{-116}	0.94254	large
EmoClassBERT-E	EmoClassRFC-O	4.0	2.43425×10^{-116}	0.94192	large
EmoClassBERT-E	EmoClassCodeBERT-S	11 183.5	1.40559×10^{-50}	0.16362	small
EmoClassBERT-E	EmoClassCodeBERT-T	13 700.0	8.50572×10^{-38}	0.10781	negligible
EmoClassBERT-E	EmoClassCodeBERT-E	19 343.0	1.79090×10^{-4}	0.02713	negligible
EmoClassBERT-E	EmoClassCodeBERT-O	26 083.0	7.38967×10^{-1} *	0.00699	negligible
EmoClassBERT-O	EmoClassLSTM-S	204.0	1.00153×10^{-115}	0.96252	large
EmoClassBERT-O	EmoClassLSTM-T	19.0	3.10836×10^{-116}	0.96268	large
EmoClassBERT-O	EmoClassLSTM-E	6.0	2.93978×10^{-116}	0.96326	large
EmoClassBERT-O	EmoClassLSTM-O	363.5	1.35996×10^{-115}	0.96135	large
EmoClassBERT-O	EmoClassRFC-S	775.0	7.46804×10^{-115}	0.93024	large
EmoClassBERT-O	EmoClassRFC-T	776.0	7.16232×10^{-115}	0.92869	large
EmoClassBERT-O	EmoClassRFC-E	743.0	6.56139×10^{-115}	0.92785	large
EmoClassBERT-O	EmoClassRFC-O	774.0	6.77701×10^{-115}	0.92725	large
EmoClassBERT-O	EmoClassCodeBERT-S	28 564.5	1.34261×10^{-26}	0.13898	negligible
EmoClassBERT-O	EmoClassCodeBERT-T	30 526.5	1.31764×10^{-15}	0.08349	negligible
EmoClassBERT-O	EmoClassCodeBERT-E	36 832.5	6.83737×10^{-1} *	0.00298	negligible
EmoClassBERT-O	EmoClassCodeBERT-O	21 531.0	1.15393×10^{-3}	−0.01634	negligible
EmoClassLSTM-S	EmoClassLSTM-T	108 055.5	1.88391×10^{-1}	−0.02240	negligible
EmoClassLSTM-S	EmoClassLSTM-E	109 302.0	2.06887×10^{-1}	0.02461	negligible
EmoClassLSTM-S	EmoClassLSTM-O	106 642.0	1.11156×10^{-1}	−0.01544	negligible
EmoClassLSTM-S	EmoClassRFC-S	76 989.5	2.08736×10^{-17}	−0.21437	small
EmoClassLSTM-S	EmoClassRFC-T	83 650.5	4.43458×10^{-13}	−0.19516	small
EmoClassLSTM-S	EmoClassRFC-E	83 893.5	6.19643×10^{-13}	−0.20649	small
EmoClassLSTM-S	EmoClassRFC-O	91 179.5	7.52484×10^{-9}	−0.17538	small
EmoClassLSTM-S	EmoClassCodeBERT-S	215.0	7.20206×10^{-116}	−0.94840	large
EmoClassLSTM-S	EmoClassCodeBERT-T	119.0	4.77297×10^{-116}	−0.95625	large
EmoClassLSTM-S	EmoClassCodeBERT-E	0.0	4.17157×10^{-116}	−0.97449	large
EmoClassLSTM-S	EmoClassCodeBERT-O	0.0	4.17135×10^{-116}	−0.97679	large
EmoClassLSTM-T	EmoClassLSTM-E	99 377.0	2.07059×10^{-3}	0.04728	negligible
EmoClassLSTM-T	EmoClassLSTM-O	116 046.5	9.90057×10^{-1} *	0.00678	negligible
EmoClassLSTM-T	EmoClassRFC-S	80 052.0	3.63798×10^{-1}	−0.19956	small
EmoClassLSTM-T	EmoClassRFC-T	86 731.0	2.65058×10^{-11}	−0.18003	small
EmoClassLSTM-T	EmoClassRFC-E	86 810.5	2.93300×10^{-11}	−0.19122	small
EmoClassLSTM-T	EmoClassRFC-O	94 476.0	1.37314×10^{-7}	−0.16183	small
EmoClassLSTM-T	EmoClassCodeBERT-S	40.0	4.95359×10^{-11}	−0.94836	large
EmoClassLSTM-T	EmoClassCodeBERT-T	0.0	4.17170×10^{-116}	−0.95633	large
EmoClassLSTM-T	EmoClassCodeBERT-E	0.0	2.86477×10^{-116}	−0.97482	large
EmoClassLSTM-T	EmoClassCodeBERT-O	0.0	2.86491×10^{-116}	−0.97707	large
EmoClassLSTM-E	EmoClassLSTM-O	104 530.0	2.05498×10^{-2}	−0.04039	negligible
EmoClassLSTM-E	EmoClassRFC-S	75 175.0	2.43168×10^{-18}	−0.23016	small
EmoClassLSTM-E	EmoClassRFC-T	81 695.5	4.08018×10^{-14}	−0.20997	small
EmoClassLSTM-E	EmoClassRFC-E	81 649.5	2.61120×10^{-14}	−0.22027	small
EmoClassLSTM-E	EmoClassRFC-O	89 755.5	1.48505×10^{-9}	−0.18798	small
EmoClassLSTM-E	EmoClassCodeBERT-S	157.0	8.18665×10^{-116}	−0.94948	large
EmoClassLSTM-E	EmoClassCodeBERT-T	0.0	4.17173×10^{-116}	−0.95734	large
EmoClassLSTM-E	EmoClassCodeBERT-E	0.0	2.86496×10^{-116}	−0.97492	large
EmoClassLSTM-E	EmoClassCodeBERT-O	0.0	2.86489×10^{-116}	−0.97727	large
EmoClassLSTM-O	EmoClassRFC-S	78 856.0	2.67139×10^{-16}	−0.20616	small

Table 14 continued

Comparison between		Wilcoxon result		Cliff's Delta	
		Statistic	p-value	Value	PD
EmoClassLSTM-O	EmoClassRFC-T	85 162.0	2.39909×10^{-12}	-0.18758	small
EmoClassLSTM-O	EmoClassRFC-E	85 894.0	6.31752×10^{-12}	-0.19880	small
EmoClassLSTM-O	EmoClassRFC-O	93 549.5	5.27101×10^{-8}	-0.16845	small
EmoClassLSTM-O	EmoClassCodeBERT-S	14.0	3.04265×10^{-116}	-0.94738	large
EmoClassLSTM-O	EmoClassCodeBERT-T	5.0	2.92716×10^{-116}	-0.95528	large
EmoClassLSTM-O	EmoClassCodeBERT-E	0.0	2.86508×10^{-116}	-0.97385	large
EmoClassLSTM-O	EmoClassCodeBERT-O	0.0	2.86511×10^{-116}	-0.97637	large
EmoClassRFC-S	EmoClassRFC-T	86 325.0	9.50083×10^{-1} *	0.01028	negligible
EmoClassRFC-S	EmoClassRFC-E	88 114.5	3.04052×10^{-1}	0.00134	negligible
EmoClassRFC-S	EmoClassRFC-O	84 969.5	5.94841×10^{-1} *	0.01800	negligible
EmoClassRFC-S	EmoClassCodeBERT-S	1204.0	4.72830×10^{-114}	-0.90868	large
EmoClassRFC-S	EmoClassCodeBERT-T	379.0	1.38250×10^{-115}	-0.92185	large
EmoClassRFC-S	EmoClassCodeBERT-E	0.0	2.67542×10^{-116}	-0.94549	large
EmoClassRFC-S	EmoClassCodeBERT-O	0.0	2.67547×10^{-116}	-0.94745	large
EmoClassRFC-T	EmoClassRFC-E	80 746.0	3.49193×10^{-1}	-0.00890	negligible
EmoClassRFC-T	EmoClassRFC-O	77 657.0	8.74585×10^{-1} *	0.00859	negligible
EmoClassRFC-T	EmoClassCodeBERT-S	1099.0	2.92091×10^{-114}	-0.90710	large
EmoClassRFC-T	EmoClassCodeBERT-T	346.0	1.14477×10^{-115}	-0.92034	large
EmoClassRFC-T	EmoClassCodeBERT-E	0.0	2.51494×10^{-116}	-0.94391	large
EmoClassRFC-T	EmoClassCodeBERT-O	0.0	2.51493×10^{-116}	-0.94602	large
EmoClassRFC-E	EmoClassRFC-O	59 421.0	3.21570×10^{-1}	0.01738	negligible
EmoClassRFC-E	EmoClassCodeBERT-S	1226.0	5.21858×10^{-114}	-0.90604	large
EmoClassRFC-E	EmoClassCodeBERT-T	425.0	1.70137×10^{-115}	-0.91955	large
EmoClassRFC-E	EmoClassCodeBERT-E	0.0	2.70487×10^{-116}	-0.94291	large
EmoClassRFC-E	EmoClassCodeBERT-O	0.0	2.70488×10^{-116}	-0.94533	large
EmoClassRFC-O	EmoClassCodeBERT-S	1172.0	3.83828×10^{-114}	-0.90525	large
EmoClassRFC-O	EmoClassCodeBERT-T	417.0	1.48150×10^{-115}	-0.91886	large
EmoClassRFC-O	EmoClassCodeBERT-E	0.0	2.37949×10^{-116}	-0.94224	large
EmoClassRFC-O	EmoClassCodeBERT-O	0.0	2.37952×10^{-116}	-0.94465	large
EmoClassCodeBERT-S	EmoClassCodeBERT-T	29 327.0	4.90961×10^{-9}	-0.05782	negligible
EmoClassCodeBERT-S	EmoClassCodeBERT-E	10 957.0	3.83547×10^{-48}	-0.14031	negligible
EmoClassCodeBERT-S	EmoClassCodeBERT-O	8776.5	4.37258×10^{-57}	-0.15875	small
EmoClassCodeBERT-T	EmoClassCodeBERT-E	13 447.0	3.84252×10^{-31}	-0.08313	negligible
EmoClassCodeBERT-T	EmoClassCodeBERT-O	10 694.0	9.68241×10^{-42}	-0.10171	negligible
EmoClassCodeBERT-E	EmoClassCodeBERT-O	17 422.5	1.90539×10^{-6}	-0.02017	negligible

Results. Table 13 shows that the BERT-based and CodeBERT-based models performed for all emotion categories. We also notice that there is not much difference in the performance of models on augmented datasets and non-augmented datasets. These results show an interesting insight that data augmentation techniques need to be adapted using SE-specific vocabulary. In the future, we work on improving these data augmentation techniques using SE-specific data. Table 14 shows the results of Wilcoxon signed-rank and Cliff's delta. These results indicate that in most of the cases, the p-value obtained is lower than 0.05 and hence, the null hypothesis is rejected, which shows that the model performance is significantly different.

The BERT-based and CodeBERT-baed models perform best for emotion classification. There is no significant difference in the performance of models trained on augmented data and non-augmented data.

7. Threats to validity

This section delves into possible factors that could impact the credibility of this research. These factors are categorized into internal, external, and construct validity [66].

7.1. Internal validity

Internal validity relates to the potential design elements of a study that could affect its outcomes. One of those is overfitting, which emerges when a model, in its attempt to minimize loss, starts to memorize the training data rather than understand the underlying patterns [67] leading it to excel on the training dataset but underperform on unseen data. To prevent overfitting, the training process of all three models was monitored closely. Early stopping techniques and dropout were adopted for the LSTM model. Experiments were conducted with the BERT model to identify the most appropriate BERT pre-trained model and set of hyperparameters for our task. For the RFC model, the GridSearchCV function was used to obtain the best combination of parameters.

Data augmentation techniques, like word substitution and back translation, were employed to enrich our limited dataset. However, they pose challenges. Word substitution can change emotional nuances, and back translation might alter original sentiments, potentially misleading the model during training. To address these, we employed contextual word substitution and reviewed a subset of the augmented data for accuracy. For the back translation approach, the samples were increased once to avoid introducing duplicates. Furthermore, only a portion of the minority classes was increased using the augmentation methods. Additionally, labeling Stack Overflow posts can introduce interpretation variances. We countered this by using the gold label, determined by a majority vote system as noted by [19].

7.2. External validity

This section outlines potential limitations that might affect the broader applicability of the results from this study. This research utilized an annotated dataset extracted from Stack Overflow, a prominent Q&A platform for software developers. Although Stack Overflow is a major hub for developer discussions, it is worth noting that there are other platforms, such as GitHub, where developers also engage in conversations. Since the models implemented in this study were trained on a dataset extracted from Stack Overflow, the results might differ if evaluated on the dataset from another platform. Furthermore, factors such as differences in the domain, annotator biases, and varying annotation guidelines can impact performance as observed by [68]. Nonetheless, for a more comprehensive generalization, future research could incorporate posts and comments from other Q&A platforms.

7.3. Construct validity

Construct validity examines how well theoretical concepts are translated into actual observations. It evaluates whether the methods used in research truly capture the abstract ideas they intend to measure. Concerns about construct validity arise from the appropriateness of our evaluation metrics, and the reliability of the manually annotated dataset used. Evaluation metrics, including precision, recall, and the F_1-score, are the benchmarks against which the efficacy of sentiment analysis or emotion classification solutions are gauged, as supported by [2, 14, 40, 56]. These metrics serve as the foundation for understanding our solutions' true performance and reliability. However, a significant aspect to consider is the source of our data. By using publicly available datasets utilized in prior works, we are not just leveraging the data but also inheriting any potential inaccuracies or biases inherent in the dataset. As a result, it is essential to recognize this inherited vulnerability when interpreting our findings and drawing conclusions.

8. Conclusion and future work

Detecting software engineers' emotions has become increasingly important in understanding team dynamics, improving collaboration, and enhancing overall productivity in software development projects. In this study, we implemented four different ML architectures – BERT, CodeBERT, RFC, and LSTM – for the emotion classification task. To improve the performance and robustness of the models, three techniques, word substitution, back translation, and Easy Data Augmentation, were used to augment the training data. Four variations of each architecture were implemented: one using data augmented through word substitution, the second using back-translated data, the third using easy data augmentation, and the fourth using the original data.

EmoClassBERT-substitution gives the highest F_1-score of 63%. The substitution method shows a considerable improvement as compared to the original models, for example, EmoClassLSTM-Substitution, EmoClassBERT-Substitution, EmoClassCodeBERT-Substitution, and EmoClassRFC-Substitution models show improvements of 13%, 5%, 5%, and 10% as compared to EmoClassLSTM-Original. Overall, the BERT-based and CodeBERT-based models perform best for emotion classification. The results reveal no significant difference in the performance of models trained on augmented data and non-augmented data.

This underscores the ability of transformer-based models to capture semantic and contextual relationships, making them particularly suited for tasks involving complex textual data such as emotion classification. In comparison, the LSTM models demonstrated inferior performance, likely due to limited data as they need abundant data for training. Despite being the simplest of the three models, the RFC provided better results than the LSTM model, highlighting the potential for traditional ML techniques in emotion classification tasks when combined with robust feature extraction and data augmentation. In summary, BERT mostly outperformed both RFC and LSTM in terms of the F_1-score. Furthermore, the augmentation techniques played a pivotal role in refining our models. Specifically, word substitution showcased a more pronounced improvement in the model's performance than back translation. Moreover, EmoClassBERT-Substitution demonstrated a reliable performance when compared to existing tools.

From the findings obtained, several interesting future directions are possible. For example, In the future, we plan to explore additional data augmentation techniques, such as Generative Adversarial Networks (GANs), and combine the strengths of various models to form an ensemble model. We plan to do an exhaustive comparison of various data augmentation techniques for emotion classification on the SE dataset as well as using a stack of data augmentation as proposed by [2]. We also plan to use LLM like RoBERT, and ALBERT to analyze their performance for emotion classification in the software engineering domain. Furthermore, deeper hyperparameter tuning could be performed. The dataset could also be expanded to encompass various sources, such as combining data from Stack Overflow and GitHub. Additionally, we will work on improving these data augmentation techniques using SE-specific data.

CRediT authorship contribution statement

Author 1: Methodology, software, investigation, writing – original draft, writing – review and editing, visualization.
Author 2: Conceptualization, software, writing – original draft, writing – review and editing, supervision, project administration.

Declaration of competing interest

No competing interest.

Funding

This research was conducted without any external funding.

References

[1] D. Girardi, F. Lanubile, N. Novielli, and A. Serebrenik, "Emotions and perceived productivity of software developers at the workplace," *IEEE Transactions on Software Engineering*, Vol. 48, No. 9, 2021, pp. 3326–3341.
[2] M.M. Imran, Y. Jain, P. Chatterjee, and K. Damevski, "Data augmentation for improving emotion recognition in software engineering communication," in *Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering*, 2022, pp. 1–13.
[3] M. Ortu, B. Adams, G. Destefanis, P. Tourani, M. Marchesi et al., "Are bullies more productive? Empirical study of affectiveness vs. issue fixing time," in *12th Working Conference on Mining Software Repositories.* IEEE, 2015, pp. 303–313.
[4] S. Cagnoni, L. Cozzini, G. Lombardo, M. Mordonini, A. Poggi et al., "Emotion-based analysis of programming languages on Stack Overflow," *ICT Express*, Vol. 6, No. 3, 2020, pp. 238–242.
[5] N. Forsgren, M.A. Storey, C. Maddila, T. Zimmermann, B. Houck et al., "The space of developer productivity: There's more to it than you think." *Queue*, Vol. 19, No. 1, 2021, pp. 20–48.
[6] D. Girardi, N. Novielli, D. Fucci, and F. Lanubile, "Recognizing developers' emotions while programming," in *Proceedings of the ACM/IEEE 42nd international conference on software engineering*, 2020, pp. 666–677.
[7] D. Graziotin, F. Fagerholm, X. Wang, and P. Abrahamsson, "What happens when software developers are (un) happy," *Journal of Systems and Software*, Vol. 140, 2018, pp. 32–47.
[8] N. Novielli and A. Serebrenik, "Sentiment and emotion in software engineering," *IEEE Software*, Vol. 36, No. 5, 2019, pp. 6–23.
[9] B. Lin, N. Cassee, A. Serebrenik, G. Bavota, N. Novielli et al., "Opinion mining for software development: A systematic literature review," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, Vol. 31, No. 3, 2022, pp. 1–41.
[10] N. Imtiaz, J. Middleton, P. Girouard, and E. Murphy-Hill, "Sentiment and politeness analysis tools on developer discussions are unreliable, but so are people," in *Proceedings of the 3rd International Workshop on Emotion Awareness in Software Engineering*, 2018, pp. 55–61.
[11] F. Calefato, F. Lanubile, F. Maiorano, and N. Novielli, "Sentiment polarity detection for software development," in *Proceedings of the 40th International Conference on Software Engineering*, 2018, p. 128.
[12] L. Yao and Y. Guan, "An improved LSTM structure for natural language processing," in *International Conference of Safety Produce Informatization (IICSPI).* IEEE, 2018, pp. 565–569.

[13] J. Antony Vijay, H. Anwar Basha, and J. Arun Nehru, "A dynamic approach for detecting the fake news using random forest classifier and NLP," in *Computational Methods and Data Engineering*. Springer, 2020, pp. 331–341.

[14] D. Bleyl and E.K. Buxton, "Emotion recognition on Stack Overflow posts using BERT," in *International Conference on Big Data (Big Data)*. IEEE, 2022, pp. 5881–5885.

[15] T. Ahmed, A. Bosu, A. Iqbal, and S. Rahimi, "SentiCR: A customized sentiment analysis tool for code review interactions," in *32nd IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 2017, pp. 106–111.

[16] V. Kumar, A. Choudhary, and E. Cho, "Data augmentation using pre-trained transformer models," *arXiv preprint arXiv:2003.02245*, 2020.

[17] S. Shleifer, "Low resource text classification with ulmfit and backtranslation," *CoRR*, Vol. abs/1903.09244, 2019. [Online]. http://arxiv.org/abs/1903.09244

[18] A. Koufakou, D. Grisales, O. Fox et al., "Data augmentation for emotion detection in small imbalanced text data," *arXiv preprint arXiv:2310.17015*, 2023.

[19] N. Novielli, F. Calefato, and F. Lanubile, "A gold standard for emotion annotation in stack overflow," in *Proceedings of the 15th International Conference on Mining Software Repositories*, MSR '18. New York, NY, USA: Association for Computing Machinery, 2018, pp. 14–17. [Online]. https://doi.org/10.1145/3196398.3196453

[20] A. Murgia, M. Ortu, P. Tourani, B. Adams, and S. Demeyer, "An exploratory qualitative and quantitative analysis of emotions in issue report comments of open source systems," *Empirical Software Engineering*, Vol. 23, 2017, pp. 521–564.

[21] B. Liu, *Sentiment analysis and opinion mining*. Springer Nature, 2022.

[22] P. Sudhir and V.D. Suresh, "Comparative study of various approaches, applications and classifiers for sentiment analysis," *Global Transitions Proceedings*, Vol. 2, No. 2, 2021, pp. 205–211.

[23] O. Bruna, H. Avetisyan, and J. Holub, "Emotion models for textual emotion classification," *Journal of Physics: Conference Series*, Vol. 772, No. 1, 2016, p. 012063. [Online]. https://dx.doi.org/10.1088/1742-6596/772/1/012063

[24] Z. Teng, F. Ren, and S. Kuroiwa, "Retracted: recognition of emotion with svms," in *Computational Intelligence: International Conference on Intelligent Computing*. Springer, 2006, pp. 701–710.

[25] E. Guzman and B. Bruegge, "Towards emotional awareness in software development teams," in *Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering*, ESEC/FSE 2013. New York, NY, USA: Association for Computing Machinery, 2013, pp. 671–674. [Online]. https://doi.org/10.1145/2491411.2494578

[26] A. Fontão, O.M. Ekwoge, R. Santos, and A.C. Dias-Neto, "Facing up the primary emotions in mobile software ecosystems from developer experience," in *Proceedings of the 2nd Workshop on Social, Human, and Economic Aspects of Software*, WASHES '17. New York, NY, USA: Association for Computing Machinery, 2017, pp. 5–11. [Online]. https://doi.org/10.1145/3098322.3098325

[27] E. Guzman, D. Azócar, and Y. Li, "Sentiment analysis of commit comments in github: An empirical study," in *Proceedings of the 11th Working Conference on Mining Software Repositories*, MSR 2014. New York, NY, USA: Association for Computing Machinery, 2014, pp. 352–355. [Online]. https://doi.org/10.1145/2597073.2597118

[28] A. Murgia, P. Tourani, B. Adams, and M. Ortu, "Do developers feel emotions? an exploratory analysis of emotions in software artifacts," in *Proceedings of the 11th Working Conference on Mining Software Repositories*, MSR 2014. New York, NY, USA: Association for Computing Machinery, 2014, pp. 262–271. [Online]. https://doi.org/10.1145/2597073.2597086

[29] G. Uddin and F. Khomh, "Automatic mining of opinions expressed about apis in stack overflow," *IEEE Transactions on Software Engineering*, Vol. 47, No. 3, 2019, pp. 522–559.

[30] A. Ciurumelea, A. Schaufelbühl, S. Panichella, and H.C. Gall, "Analyzing reviews and code of mobile apps for better release planning," in *2017 IEEE 24th International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 2017, pp. 91–102.

[31] X. Gu and S. Kim, "" what parts of your apps are loved by users?"(t)," in *2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 2015, pp. 760–770.

[32] S. Panichella, A. Di Sorbo, E. Guzman, C.A. Visaggio, G. Canfora et al., "How can i improve my app? classifying user reviews for software maintenance and evolution," in *2015 IEEE international conference on software maintenance and evolution (ICSME)*. IEEE, 2015, pp. 281–290.

[33] M.M. Rahman, C.K. Roy, and I. Keivanloo, "Recommending insightful comments for source code using crowdsourced knowledge," in *2015 IEEE 15th International Working Conference on Source Code Analysis and Manipulation (SCAM)*. IEEE, 2015, pp. 81–90.

[34] R. Jongeling, S. Datta, and A. Serebrenik, "Choosing your weapons: On sentiment analysis tools for software engineering research," in *2015 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, 2015, pp. 531–535.

[35] M. Thelwall, K. Buckley, G. Paltoglou, A. Kappas, and D. Cai, "Sentiment strength detection in short informal text," *Journal of the American Society for Information Science and Technology*, 2010.

[36] E. Loper and S. Bird, "Nltk: The natural language toolkit," in *Proceedings of the ACL-02 Workshop on Effective Tools and Methodologies for Teaching Natural Language Processing and Computational Linguistics*, ETMTNLP '02, Vol. 1. Association for Computational Linguistics, 2002, pp. 63–70. [Online]. https://doi.org/10.3115/1118108.1118117

[37] M.R. Islam and M.F. Zibran, "Leveraging automated sentiment analysis in software engineering," in *2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR)*, 2017, pp. 203–214.

[38] F. Calefato, F. Lanubile, F. Maiorano, and N. Novielli, "Sentiment polarity detection for software development," *Empirical Software Engineering*, Vol. 23, No. 3, 2017, pp. 1352–1382.

[39] D. Graziotin, X. Wang, and P. Abrahamsson, "Do feelings matter? on the correlation of affects and the self-assessed productivity in software engineering," *Journal of Software: Evolution and Process*, Vol. 27, No. 7, 2015, pp. 467–487. [Online]. https://onlinelibrary.wiley.com/doi/abs/10.1002/smr.1673

[40] F. Calefato, F. Lanubile, and N. Novielli, "Emotxt: A toolkit for emotion recognition from text," *CoRR*, Vol. abs/1708.03892, 2017. [Online]. http://arxiv.org/abs/1708.03892

[41] N. Boucher, I. Shumailov, R.J. Anderson, and N. Papernot, "Bad characters: Imperceptible NLP attacks," *CoRR*, Vol. abs/2106.09898, 2021. [Online]. https://arxiv.org/abs/2106.09898

[42] Y. HaCohen-Kerner, D. Miller, and Y. Yigal, "The influence of preprocessing on text classification using a bag-of-words representation." *PloS one*, Vol. 15, 2020, p. e0232525.

[43] J.J. Webster and C. Kit, "Tokenization as the initial phase in NLP," in *The 14th international conference on computational linguistics*, 1992.

[44] N. Rahimi, F. Eassa, and L. Elrefaei, "An ensemble machine learning technique for functional requirement classification," *symmetry*, Vol. 12, No. 10, 2020, p. 1601.

[45] A. Humphreys and R.J.H. Wang, "Automated text analysis for consumer research," *Journal of Consumer Research*, Vol. 44, 2018, pp. 1274–1306. [Online]. https://api.semanticscholar.org/CorpusID:168854843

[46] L. Tian, C. Lai, and J.D. Moore, "Polarity and intensity: The two aspects of sentiment analysis," *arXiv preprint arXiv:1807.01466*, 2018.

[47] A. Ali, S.M. Shamsuddin, and A.L. Ralescu, "Classification with class imbalance problem," *Int. J. Advance Soft Compu. Appl*, Vol. 5, No. 3, 2013, pp. 176–204.

[48] X.Y. Liu, J. Wu, and Z.H. Zhou, "Exploratory undersampling for class-imbalance learning," *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, Vol. 39, No. 2, 2009, pp. 539–550.

[49] Q. Liu, M.J. Kusner, and P. Blunsom, "A survey on contextual embeddings," 2020.

[50] P. Bahad, P. Saxena, and R. Kamal, "Fake news detection using bi-directional lstm-recurrent neural network," *Procedia Computer Science*, Vol. 165, 2019, pp. 74–82, 2nd International Conference on Recent Trends in Advanced Computing DISRUP-TIV INNOVATION. [Online]. https://www.sciencedirect.com/science/article/pii/S1877050920300806

[51] C. Zhou, C. Sun, Z. Liu, and F.C.M. Lau, "A C-LSTM neural network for text classification," *CoRR*, Vol. abs/1511.08630, 2015. [Online]. http://arxiv.org/abs/1511.08630

[52] Y. Zhang, "Research on text classification method based on lstm neural network model," in *2021 IEEE Asia-Pacific Conference on Image Processing, Electronics and Computers (IPEC)*, 2021, pp. 1019–1022.

[53] R. Adarsh, A. Patil, S. Rayar, and K. Veena, "Comparison of VADER and LSTM for sentiment analysis," *International Journal of Recent Technology and Engineering*, Vol. 7, No. 6, Mar. 2019, pp. 540–543.

[54] S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory," *Neural Computation*, Vol. 9, No. 8, 11 1997, pp. 1735–1780. [Online]. https://doi.org/10.1162/neco.1997.9.8.1735

[55] B. Lindemann, T. Müller, H. Vietz, N. Jazdi, and M. Weyrich, "A survey on long short-term memory networks for time series prediction," *Procedia CIRP*, Vol. 99, 2021, pp. 650–655, 14th CIRP Conference on Intelligent Computation in Manufacturing Engineering, 15–17 July 2020. [Online]. https://www.sciencedirect.com/science/article/pii/S2212827121003796

[56] H. Batra, N.S. Punn, S.K. Sonbhadra, and S. Agarwal, "BERT-based sentiment analysis: A software engineering perspective," in *Database and Expert Systems Applications*. Springer International Publishing, 2021, pp. 138–148.

[57] J. Devlin, M. Chang, K. Lee, and K. Toutanova, "BERT: pre-training of deep bidirectional transformers for language understanding," *CoRR*, Vol. abs/1810.04805, 2018. [Online]. http://arxiv.org/abs/1810.04805

[58] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones et al., "Attention is all you need," *CoRR*, Vol. abs/1706.03762, 2017. [Online]. http://arxiv.org/abs/1706.03762

[59] Y. Al Amrani, M. Lazaar, and K.E. El Kadiri, "Random Forest and Support Vector Machine based hybrid approach to sentiment analysis," *Procedia Computer Science*, Vol. 127, 2018, pp. 511–520, proceedings of the first International Conference On Intelligent Computing in Data Sciences, ICDS2017. [Online]. https://www.sciencedirect.com/science/article/pii/S1877050918301625

[60] L. Breiman, "Random forests," *Machine Learning*, Vol. 45, No. 1, 2001, pp. 5–32. [Online]. https://doi.org/10.1023/A:1010933404324

[61] M. Wu, Y. Yang, H. Wang, and Y. Xu, "A deep learning method to more accurately recall known lysine acetylation sites," *BMC Bioinformatics*, Vol. 20, No. 1, 2019, p. 49. [Online]. https://doi.org/10.1186/s12859-019-2632-9

[62] Z. Feng, D. Guo, D. Tang, N. Duan, X. Feng et al., "Codebert: A pre-trained model for programming and natural languages," *arXiv preprint arXiv:2002.08155*, 2020.

[63] J. Ramos et al., "Using tf-idf to determine word relevance in document queries," in *Proceedings of the first instructional conference on machine learning*. Citeseer, 2003, pp. 29–48.

[64] C. Tantithamthavorn, S. McIntosh, A.E. Hassan, and K. Matsumoto, "An empirical comparison of model validation techniques for defect prediction models," *IEEE Transactions on Software Engineering*, Vol. 43, No. 1, 2016, pp. 1–18.

[65] J. Romano, J.D. Kromrey, J. Coraggio, and J. Skowronek, "Appropriate statistics for ordinal level data: Should we really be using t-test and cohen'sd for evaluatng group differences on the nsse and other surveys," in *annual meeting of the Florida Association of Institutional Research*, Vol. 177, 2006, p. 34.

[66] L. Baldwin, "Internal and external validity and threats to validity," in *Research concepts for the practitioner of educational leadership*. Brill, 2018, pp. 31–36.

[67] X. Ying, "An overview of overfitting and its solutions," *Journal of Physics: Conference Series*, Vol. 1168, No. 2, feb 2019, p. 022022. [Online]. https://dx.doi.org/10.1088/1742-6596/1168/2/022022

[68] L.A. Cabrera-Diego, N. Bessis, and I. Korkontzelos, "Classifying emotions in Stack Overflow and JIRA using a multi-label approach," *Knowledge-Based Systems*, Vol. 195, 2020, p. 105633. [Online]. https://www.sciencedirect.com/science/article/pii/S0950705120300939

Authors and affiliations

Didi Awovi Ahavi-Tete
e-mail: didi.ahavitete@gmail.com
ORCID: https://orcid.org/0009-0001-7348-2763
School of Computer Science and Mathematics,
Keele University, United Kingdom

Sangeeta Sangeeta
e-mail: s.sangeeta@keele.ac.uk
ORCID: https://orcid.org/0000-0002-3734-7871
School of Computer Science and Mathematics,
Keele University, United Kingdom

e-Informatica Software Engineering Journal
Volume 19, Issue 1, pages: 1–19
DOI: 10.37190/e-Inf250105

BibTeX

Guidelines for Conducting Action Research Studies in Software Engineering

Miroslaw Staron*
*Corresponding author: miroslaw.staron@gu.se

Article info

Keywords:
action research
guidelines
empirical methods

Submitted: 25 Jun. 2024
Revised: 2 Jan. 2025
Accepted: 2 Jan. 2025
Available online: 8 Jan. 2025

Abstract

Context: Action research is popular in software engineering due to its industrial nature and promises of effective technology transfers. Yet, the methodology is still gaining popularity, and guidelines for conducting quality action research studies are needed.
Objective: This paper aims to collect, summarize, and discuss guidelines for conducting action research in academia-industry collaborations. The guidelines are designed for researchers and practitioners alike.
Method: I use existing guidelines for empirical studies and my own experiences to define guidelines for researchers and host organizations for conducting action research.
Results: I identified 22 guidelines for conducting action research studies. They provide actionable recommendations on identifying the relevant context, planning and executing interventions (actions), reporting them, and reasoning around the ethics of action research.
Conclusions: The paper concludes that the best way of engaging with action research is when we can be embedded in the host organization and when the collaboration leads to tangible change in the host organization and the generation of new scientific results.

1. Introduction

After the software engineering research crisis of the 1990s, empirical software engineering gained popularity [1] as one of the remedies to the challenges with the adoption of research in industry. Experimentation was the first to receive a proper treatment with the seminal book by Wohlin et al. [2] with case studies following after [3] and design science research gaining popularity afterward [4]. Now, almost 30 years after the paper by Glass [1] that coined the term software research crisis, the research landscape is much more diverse. The major conferences and journals are significantly more mature in assessing publications, and the need for explicit research methodology for every study is obvious. In 2020, the ACM published guidelines for reviewers of empirical work in software engineering [5], which contained 17 distinct research methodologies. These are just a selection of evidence that indicates that software engineering has matured as a field, although the evolution and development never stops.

Action research gained popularity in the 2000s [6, 7] when its abilities to strengthen industrial collaborations became evident for academics and practitioners alike. Although it was initially taken from the field of information systems [8], the emphasis on collaborative research and development appealed to software engineering researchers. The action research addresses the challenges and dilemmas many software engineering researchers face – how to introduce and simultaneously study new technology.

Case studies in software engineering are often meant to be observatory or participant – observatory and, therefore, prescribe objectivity in investigations of the studied phenomena. Researchers must be more observers or participants but not executors of actions and interventions at the company. The primary focus in such studies is to understand phenomena in their natural context, but sometimes changes are needed in addition to the observations. Therefore, the guidelines designed for case studies do not always apply to action research.

Experiments are driven by hypotheses and, therefore, prescribe controllability, which favors isolated, small (even toy) problems as the core of experimentation. Design science research focuses on the artifacts rather than improving the practice of the collaborating company. But when a researcher is embedded in a host company, introduces a new technology, and wants to critically and systematically evaluate it, action research is the only methodology that provides the necessary toolkit for this researcher and the required discourse of analysis of the obtained results.

In this paper, I dive into the question of *What are the necessary guidelines for planning, conducting, and reporting action research studies in software engineering?*

Action research is a well-known research methodology in information systems, pedagogy, and nursing. Therefore, to define the guidelines, I started by reviewing the existing guidelines of Davidson [9], Baskerville [8], and Bleijenbergh [10]. These three were selected as they are used as methodological sources of guidelines by being cited and because they include guidelines and recommendations explicitly. The existing guidelines were refined, adopted, and rewritten to fit the context of software engineering, where software development is the focus. Both practitioners and researchers are from within software engineering. They were also aligned with the preliminary recommendations in my previous work on action research's theory and applications [11].

The remainder of the paper is structured as follows. Section 2 presents an overview of a selection of existing studies in action research and existing guidelines for other research methodologies in software engineering (e.g., case studies). Section 3 describes what action research in academia-industry collaboration context is and which elements are essential for a successful collaboration. Section 4 presents the guidelines, and Section 5 concludes the paper.

2. Related work

Before I discuss action research principles and how they shape this research methodology, I overview the state-of-the-art in action research and guidelines for empirical studies. These provide a fundamental overview of what is recommended for researchers and practitioners today in software engineering.

2.1. Studies in action research

Although action research is not a new methodology, it entered software engineering in the 1990s. Dos Santos and Travassos [6] and their subsequent work on using action research in software engineering [7] have found that this research methodology has the potential to address the challenges of technology transfer. They identified action research as a methodology that can benefit both industry and academia but did not identify concrete guidelines on how to make such a collaboration successful. The existing guidelines for such collaborations often focus on organizational support and management engagement [12, 13].

My book on action research theory and practice in software engineering [11] followed the same principles and focused on the action research methodology to increase the impact of industry-academia collaborations. The action research methodology presented in that book focuses on a five-step cyclic model, which is similar to the view of action research of Baskerville et al. [8]. This paper follows the same model, although the guidelines are applicable for the action research models with an arbitrary number of phases.

Wieringa [4] presented Design Science Research as a research methodology for software engineering. Even there, he mentioned using the so-called "Technical Action Research," which is a kind of action research that primarily focuses on developing artifacts (e.g., programs and tools). The technical action research emphasizes the importance of the collaborative development of artifacts and the need to be applied directly in software development organizations.

Dittrich et al. [14] presented experiences from another type of action research project focused on method development, not technology development. Their experiences show how good the adoption of research results is when conducting action research. The work by Dittrich et al. has been used to guide research on the co-development of new methods in software engineering.

Petersen et al. [15] presented another set of experiences from two action research studies. They identified several positive aspects of action research studies. For example, the deeper impact of the contributions and the more manageable the transfer of the results to industrial practices. However, they also acknowledged such validity threats as context-dependency of the results.

2.2. Existing action research guidelines

Almost any publication that introduces action research as a methodology, whether within software engineering or other fields, includes a reference to the methodology. However, such references do not refer to publications that contain explicit guidelines, are applicable to software engineering, or are directly actionable. The guidelines presented in this paper are based on three other guidelines outlined below, which contain such explicit guidelines.

The most extensive guidelines available are included in the work of Davidson et al. [9]. These guidelines are developed for information systems, where the focus is put on organizational development with the help of an information system. In their work, the authors introduced the concept of *Canonical Action Research* and described five principles, each broken down into several checks:

1. The principle of researcher-client agreement
   a. Did both the researcher and the client agree that CAR was the appropriate approach for the organizational situation?
   b. Was the focus of the research project specified clearly and explicitly?
   c. Did the client make an explicit commitment to the project?
   d. Were the roles and responsibilities of the researcher and client organization members specified explicitly?
   e. Were project objectives and evaluation measures specified explicitly?
   f. Were the data collection and analysis methods specified explicitly?

2. The principle of cyclical process model (CPM)
   a. Did the project follow the CPM or justify any deviation from it?
   b. Did the researcher conduct an independent diagnosis of the organizational situation?
   c. Were the planned actions based explicitly on the results of the diagnosis?
   d. Were the planned actions implemented and evaluated?
   e. Did the researcher reflect on the outcomes of the intervention?
   f. Was this reflection followed by an explicit decision on whether or not to proceed through an additional process cycle?
   g. Were both the exit of the researcher and the conclusion of the project due to either the project objectives being met or some other clearly articulated justification?
3. The principle of theory
   a. Were the project activities guided by a theory or set of theories?
   b. Was the domain of investigation, and the specific problem setting, relevant and significant to the interests of the researcher's community of peers as well as the client?
   c. Was a theoretically based model used to derive the causes of the observed problem?
   d. Did the planned intervention follow from this theoretically-based model?
   e. Was the guiding theory, or any other theory, used to evaluate the outcomes of the intervention?
4. The principle of change through action
   a. Were both the researcher and client motivated to improve the situation?
   b. Were the problem and its hypothesized cause(s) specified as a result of the diagnosis?
   c. Were the planned actions designed to address the hypothesized cause(s)?
   d. Did the client approve the planned actions before they were implemented?
   e. Was the organization situation assessed comprehensively both before and after the intervention?
   f. Were the timing and nature of the actions taken clearly and completely documented?
5. The principle of learning through reflection
   a. Did the researcher provide progress reports to the client and organizational members?
   b. Did both the researcher and the client reflect upon the outcomes of the project?
   c. Were the research activities and outcomes reported clearly and completely?
   d. Were the results considered in terms of implications for further action in this situation?
   e. Were the results considered in terms of implications for action to be taken in related research domains?
   f. Were the results considered in terms of implications for the research community (general knowledge, informing/re-informing theory)?
   g. Were the results considered in terms of the general applicability of CAR?

These guidelines are from the field where software is seen as a tool for organizations. This distinction implies that studies within IS do not study how software is developed but how it impacts organizations. These guidelines also do not consider the necessity for researchers to be embedded in the organizations but often imply that the researchers are external and introduce a change in the organization. This is not how action research is done in software engineering – research must be embedded in the organization, as without it, the change is often abandoned. On the other hand, the change that is adopted leads to technology transfer and wider technology adoption. It also leads to more relevant research in academia, which is often overlooked by the Canonical Action Research and Technical Action Research.

Another explicit set of guidelines is available in the work of Baskerville [8]. These guidelines are conceptually closer to software engineering, but chronologically before Davidson et al.'s guidelines described above. The seven guidelines of Baskerville related to action research projects as a whole:
1. Consider the paradigm shift.
2. Establish a formal research agreement.
3. Provide a theoretical problem statement.
4. Plan data collection methods.
5. Maintain collaboration and subject learning.
6. Promote iterations.
7. Generalize accordingly.

Baskerville's guidelines are at a high abstraction level, which makes them prone to interpretation errors. For example, the third guideline – providing a theoretical problem statement – can be mistaken by conducting a diagnosis phase and identifying a gap in research and theory. Therefore, it must be clarified that action researchers must use theory when designing the entire study, as Davidson prescribes – being guided by a theory.

The third set of guidelines comes from action research applications in human resource management by Bleijenbergh [10]. These guidelines are peripheral to software engineering, but still contain good advice, e.g., the first one.
1. Make sure to involve all relevant organizational stakeholders in the research process.
2. Propose a research strategy that involves empirical observation of real-life settings to get a systemic perspective on the organizational problem.
3. Propose a research strategy that involves participatory research methods, such as workshops, focus groups, participatory modeling, and other decision support systems in developing a plan for action in close collaboration between researchers and (HRM) practitioners.
4. Communicate the need for a relatively large time investment of employees and (HRM) practitioners in the research process.
5. Communicate the need to perform action research over a relatively long period since potentially several cycles of stages are needed to understand the organizational problem and solve it.
6. Make a habit of continuously reflecting on decisions during the research process by making observational, methodological, and theoretical memos.
7. To potentially get high-impact publications out of your action research, consider combining observation of real-life settings with a field experiment.
8. In reporting, create a clear structure of the various stages and cycles of the action research process.
9. To publish action research, authors should put considerable effort into not only describing the rigor of the data collection and analysis, , but also make the contribution to scholarly knowledge explicit.

The above guidelines focus on projects where organizational change focuses on the human side of processes and methods. They are also the set of guidelines that have been cited the least. Similarly to Baskerville's guidelines, they are very abstract and need contextualization.

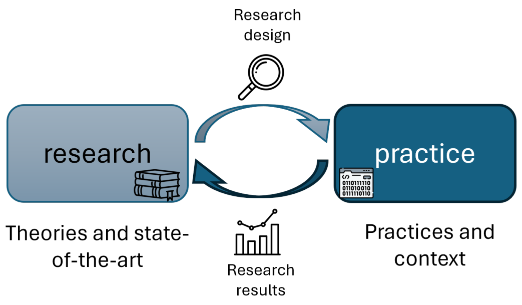
The fourth set of guidelines comes from Staron [11], in form of a checklist for making the results valid for more than one project. Staron [11] provided a rudimentary set of guidelines for conducting action research studies, focused on both researchers and practitioners, mostly in the context of method and tool development. These guidelines/checks are as follows:

1. Why do we need to have this research project?
2. What is the perception of the company about the research project?
3. In general terms, what need is the research project addressing?
4. Is the scope of the project's cycle well defined?
5. Is the scope of the research project well delimited?
6. Is the cycle well delimited in time?
7. What is/are the deliverable(s) of the research project?
8. Is there a stakeholder appointed? Does the stakeholder have the mandate to drive and implement the results of the research project?
9. Have security issues been addressed prior to the start of the research project?
10. Are employees identified that are going to support the research project?
11. If additional colleagues are participating, how are they going to be kept informed of the progress and results of the research project?
12. Are there regular meetings taking place between the stakeholder and the researcher?
13. During these regular meetings, is the time plan discussed/followed up on?
14. During these regular meetings, is the scope/deliveries/security discussed/followed up on?
15. Is data collected and stored orderly?
16. Does the storage of the data guarantee easy access and, at the same time, fulfillment of security rules?
17. First, the data is refined, analyses are performed, results are presented, and preliminary conclusions are drawn. So, how are the preliminary analyses and results handled?
18. During the second phase, results are put together to support findings and conclusions. So, are results from the evaluation rigorous enough to support the outcome of the research project?
19. How do you specify the results and knowledge to maximize the impact?
20. How should the company be informed about the outcome and findings of the research project?
21. How can the company learn, implement, and utilize the knowledge gained from research projects performed?

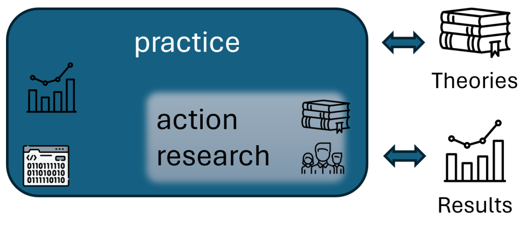
The above four guidelines cover different aspects of action research, as they are developed for different purposes. The subsequent sections describe action research in software engineering and its specifics. However, before I start with the guidelines, I need to explore what action research in software engineering is.

3. Context of action research

Action research is a collaborative research methodology [11], where researchers and practitioners work together in an action research team. On the left-hand side (Fig. 1a), the collaboration in ex vivo means that the researchers are outside of the studied context, mostly by design. Case studies are this research method, and as the researcher's objectivity in understanding the studied phenomena, the unit of analysis is prioritized. On the right-hand side (Fig. 1b), the collaboration is done in vivo, meaning that the researchers are embedded in the context that they study. This means that the researchers report to the same organization as the practitioners. Action research and design science research are examples of the latter.



(a) Ex vivo studies, where the research (and the researcher) are outside of the studied context, e.g., case studies



(b) In vivo studies, where the research is embedded in the context and the organization

Figure 1. Two ways of conducting empirical studies with industry in software engineering

In action research, the fact that the researchers are embedded in the organization means that they are either directly employed at the company or that they have the same access to the company as the employees – for example, they are employed at a research organization (a university), but they have access rights, cards, computers from the company and they are part of a team at the company. In Figure 2, the action research is placed in the context of the inputs – theories and practices, as well as the outputs – new methods, processes, and new knowledge.

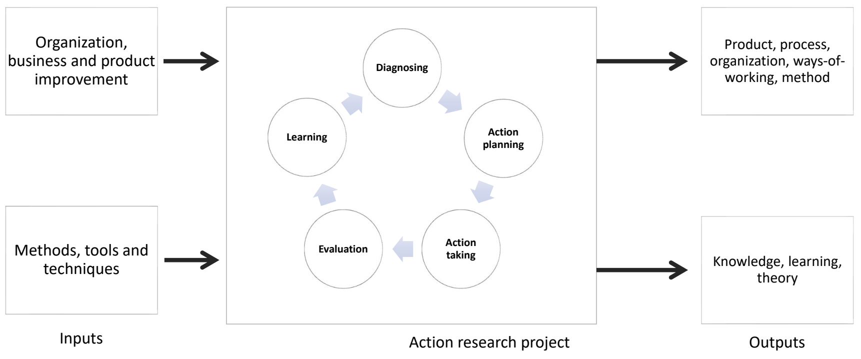

Organization, business and product improvement
Methods, tools and techniques
Inputs
Diagnosing
Action planning
Action taking
Evaluation
Learning
Action research project
Product, process, organization, ways-of-working, method
Knowledge, learning, theory
Outputs

Figure 2. Action research cycles embedded in their context – industry and academia

The in vivo embedding of the research is extremely important as it is the only way to perform interventions (actions) that are the central part of the action research [8]. The fact that the researchers are part of the intervention allows them to understand all aspects of the change they are part of. In other words, they get a first-hand experience of the intervention, not a second-hand perception of it (usually obtained through interviews and observations).

Consequently, this embedding of researchers and practitioners in the host organization (forming an action team, see [11]), the cyclical nature of action research, and explicit focus on interventions and changes to the organization call for different guidelines than the ones for case studies, surveys, or experiments.

On the left-hand side of Figure 2, both the organization and the researchers must come with explicit inputs – the researchers bring in theories, and the organizations (and practitioners) bring concrete challenges to address. Both parties must explicitly describe these inputs, explain them to one another, and agree on them. It is often an iterative process, and theories or challenges can be adjusted during the action research cycles.

However, achieving actionable and scientifically valid results is almost impossible without this understanding. Both Davidson [9] and Baskerville et al. [8] call for theories, but the software engineering context also calls for explicit specification of the initial needs of organization, business, and product improvements from the host organization.

The upfront design and the reliance on theories improve the construct validity of the action research studies. The action team is more confident that the results obtained are based on the correct observations and interventions, not due to chance or confounding factors.

3.1. What constitutes action research

Action research studies build upon three pillars, as shown in Figure 3. We need all of them to make the action research studies work.

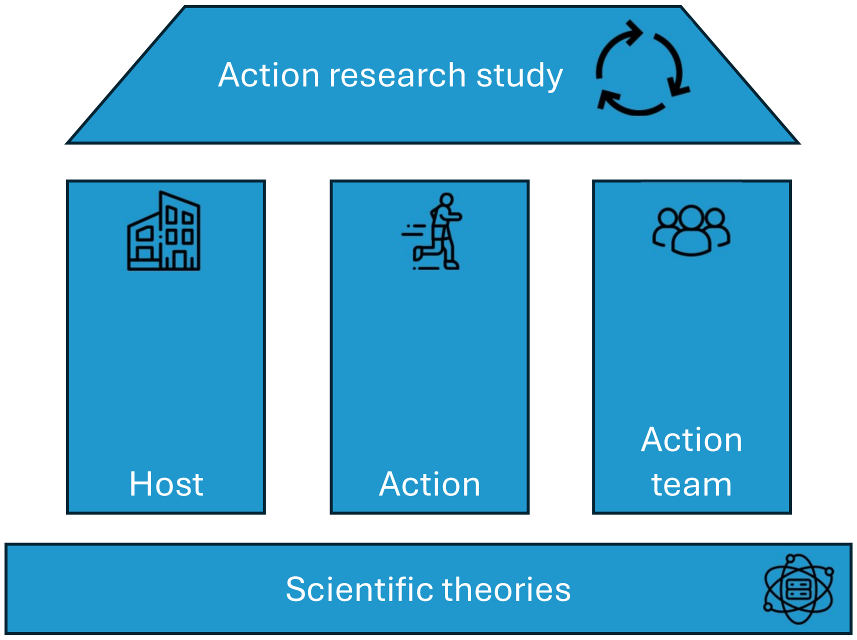


Figure 3. Pillars of action research – host organization, intervention, and the action team

The most important element of any action research study is the **action/intervention**. The action, a synonym for the intervention, is when we change the practice in the host organization. We do that to observe its results and rigorously create new knowledge and insights from that action. The interventions often include improvements to the operations, such as introducing new tools or changing the ways of working in the organization. An example of such an intervention can be changing how the company writes its commit messages by adding the ID of the user story that is implemented; the effect of this action would be to find how many commits are done per user story to understand how to accelerate software development.

The host organization, which directly benefits from the action research project, holds significant responsibilities for implementing the intervention. It must accommodate the action team by granting access to external researchers and ensuring that internal team members have the necessary time and resources to conduct the study. The organization should also allow the action team to obtain time from the reference team and the management team because they need to help shape and guide high-level goals for the action research project.

We should remember that different organizations and management/governance structures exist. In hierarchical organizations, we must always ensure that the management is informed and involved in the study. They make the decisions, and their permissions are crucial in such organizations. We should engage with our research partners to find the appropriate management level. In more agile, self-organized, or empowered organizations, we must ensure that we have sufficient engagement from the teams and individuals, as they are the ones who are in control of their time and, to some extent, work assignments. In such organizations, we should engage with management after we have engaged the teams.

Finally, the action team must be assembled with researchers and practitioners with the necessary competence. First, at least some research team members should have a research degree because they need skills in planning and conducting quality research studies. The researchers should also be able to identify the novelty of the studied topic – not all topics are equally important for the research community. Researchers need to consider the research value of any action research project. If the topic is not new, the study should consider that by planning for a replication (often with modification) or by expanding on the existing research results by modifying the context to increase the value for the company (do not re-do the work of others) and for the research community (create new findings). Second, at least some action team members should be practitioners from the host company. They have domain competence, which is required for accurate planning, executing, and interpreting the action research project results. Per definition, the ways of working at the company and the company's ability to use the results are in the hands of the host organization and, by extension, by the practitioners of that organization.

3.2. Situations where action research is not appropriate

One of the main challenges when conducting action research studies is understanding when it is inappropriate and when to pivot and change the research methodology. When engaging in action research, there are a few cues. First, if we do not have access to the company and no company representative is part of the research team, we must choose another research methodology. The lack of access means we cannot conduct necessary interventions for action research.

In the context of industry-academia collaborative action research, we should not engage in action research if we cannot guarantee transparency and the ability to share results (to a degree, we cannot demand that from our industrial partners). Instead, we should focus on conducting offline studies to recreate the company's context, allowing us to conduct experiments.

When researchers cannot commit to the project for the long term, we should also choose another methodology. Irregular meetings, short presentations, and a lack of systematic collaboration lead to poor or no results. We should also not engage when we cannot contribute to the practice and we are after publications, not impacting the industry. The lack of direct industrial impact (more than just publications) indicates that we are not engaging in action research.

4. Guidelines

The guidelines presented in this section have been designed based on guidelines for collaboration with companies from my own research [11, 16], as well as the guidelines from others [3, 8–10]. These guidelines relate to both the Canonical Action Research [9] and Technical Action Research [4].

4.1. Project set-up

I can now introduce the first guideline, which is based on two existing recommendations from Davidson [9] and one from Baskerville et al. [8]. It is about the starting of the action research project.

Guideline 1: Engage in action research only when it can be embedded in the host organization.

Collaboration with an industrial partner or being a part of a software development organization is necessary for an action research project, but it is not a sufficient condition. First, both parties must agree that action research is appropriate and allowed in the organization [9]. We need to be able to apply research results to the organization directly. We must be able to make interventions in the organization and observe their effects.

From the organization's perspective, the researchers must be embedded in it because they must understand the details of the company's operations. To understand it, they must have access to the documents, processes, and products in the same way as the company's employees. This means there is a paradigm shift in the organization – research is done in parallel to the normal operations [8]. If we cannot embed actions/interventions in the host organization, I recommend choosing other types of research methodologies, in particular, design science research or a case study. We should choose the case study based on whether the goal is to observe the organization and keep the distance from its operations and whether our goal is to understand these operations without influencing them. We should choose design science research when our goal is to create and develop a new artifact and to study its use in an industrial context. In that case, we can and should influence the company's operations, but we should focus on the qualities of this artifact.

From the researcher's perspective, the actions should be guided by theories in software engineering. We must be able to find existing ones or formulate new ones; otherwise, scientific investigation will not be guided by theories. As software engineering theories are formulated as strictly as in other fields, e.g., mathematics or physics, we can substitute them with existing empirical results. We can use existing empirical evidence to guide us when planning actions. We can also use existing research results when we explicitly want to validate them in our action research project. The next guideline makes it explicit:

Guideline 2: The objectives of the research project must be specified upfront.

When conducting action research projects, we must be clear about the project's aims, goals, and objectives. The researchers should specify their research results, publications, and engagement expectations. The practitioners should be transparent about their expectations regarding resources, the ability to conduct interventions, and collaboration with academic partners. When discussing these expectations, one or both parties may realize that action research is not the best methodology to study the phenomena in question or deliver the desired results. Therefore, we need to be aware that there is always the possibility to pivot, hence the next guideline.

Guideline 3: Be ready to pivot on your research methodology if the situation is not optimal for action research.

Every action research cycle has the perfect activity where we decide upon the continuation of the project – the learning phase. Davidson [9] explicitly calls for evaluating whether it is appropriate to continue. However, discontinuing action research can mean pivoting on the research methodology. We should be prepared for it because a non-optimal environment for action research can lead to failed projects and a lack of tangible results. In most cases, to get more familiar with the organization, it is better to start with a case study and follow the guidelines by Runeson et al. [3] or follow the guidelines by Wohlin et al. [17] to find even a better fit.

4.2. Host environment

The host environment must be organized to ensure both parties a positive action research experience. The academic side should be able to test theories and ideas. In contrast, the industrial side should perceive value in increased knowledge, enhanced competence, improved products, new features, or operational improvements. This balance is essential for successful collaboration.

Action research projects must be embedded within the host organization. Company-employee researchers often conduct these projects internally, frequently leading to applied results. External members of the action team must ensure they are treated equally. This consideration leads to the following recommendation.

Guideline 4: Action team should comprise both practitioners and researchers alike.

Just as the input values are both theories and the organization's challenges, the action team must reflect that. However, it is strongly advised to use the host organization's computers and infrastructure to ensure information security. This ensures that any information intended for publication is thoroughly scrutinized before being taken outside the organization. Therefore, here is my next guideline:

Guideline 5: Understand the host organization and secure appropriate engagement in the correct order: approach management first in hierarchical organizations, and start with the team in empowered organizations.

All previous recommendations have similar guidelines, although they are presented more fragmentedly. They either call for the involvement of all relevant parties (Bleijenbergh [10]), explicit commitment (Davidson [9]), and formal research agreement (Baskerville et al. [8]). Modern software development organizations are not as hierarchical as they used to be (or as non-software organizations are). Therefore, the action research collaboration agreement stretches over several levels of these organizations.

Therefore, the following guideline, which is for researchers from an external organization, is equally important.

Guideline 6: The researchers must respect the rules, principles, and obligations of the host organization as if they were employed there.

This guideline means that researchers must read, understand, and adhere to the host organization's rules, principles, and obligations as if they were employees. These obligations include maintaining confidentiality, upholding the organization's standards and practices throughout the research project, and being transparent and loyal to the organization.

Although it may be considered challenging for academic freedom to pursue any research problem or publish the results based on the findings, it is not. Software development companies' intellectual property is often soft and needs protection. Revealing parts of the source code, tests, or requirements (to name a few examples) can lead to leaking intellectual property to competitors. Therefore, researchers must be able to generalize their results to such a level that does not harm the company and is transferable to other contexts.

4.3. Interventions

One of my long-term collaborators once aptly summarized the essence of interventions in action research by saying, "Somebody has to do something." This captures the fundamental principle that action research must involve meaningful, tangible actions within the host company. To make these tangible observations and improvements, we must plan the actions and plan how to observe their results. Therefore, we need to follow the next guideline.

Guideline 7: Every intervention should be planned, lead to observable effects, and properly evaluated.

The observable effects must be both quantitative and qualitative because, without any observable effect, it's impossible to understand what the action/intervention changed. Quantitative effects can be collected by measuring products, processes, or designs, while qualitative effects can be observed through interviews, workshops, and post-mortem analyses. The quantitative aspect allows measuring the intervention's effects, while the qualitative component captures the broader context.

Consequently, we must prepare (plan) for the intervention together in the action team. We must measure the baselines and conduct qualitative data collection before the intervention. After the intervention, we must evaluate the effect and collect the exact measurements and qualitative data to understand the scope, size, and magnitude of the impact. Davidson [9] has a similar recommendation but without the emphasis on the observability and measurability of the effect, which I find very important.

Implementing concrete changes is the starting point and a prerequisite for its success. The boldface accent indicates that the emphasis is on the implementation.

Guideline 8: Interventions in each cycle should be atomic.

Long interventions are prone to interruptions; the longer the intervention, the higher the risk that other priorities will take precedence over the research study. Common disruptors, such as implementing new features or investigating new defects, can cause the host organization to shift focus. These disruptions are confounding factors, making it difficult to determine whether the observed results are due to the intervention or these other changes. Davidson [9] recommends specifying the focus up-front, but we can go even further and ensure that the focus is as atomic as possible.

If the intervention's nature requires longevity, the action team should include measures to capture the effects of potential confounding factors. The action team must also be prepared to adjust the data collection methods amidst changes to the context.

The host organization must be prepared for this, which means allocating additional resources, adjusting the project schedule, and accounting for the risk of failure in the current plan, such as in the ongoing sprint. Conducting a mock-up study or creating a pilot product to demonstrate a tool outside of current practices is not action research, as it does not introduce observable changes; it is merely an offline study. The next guideline is therefore:

Guideline 9: Practitioners in the action team must be able to conduct the intervention.

Practitioners should intervene within the scope of their work rather than attempting to change or advocate for the practices of other teams or organizations. The intervention should focus on their methods, tools, and infrastructure, ensuring that the context of the intervention is specific to their environment and not external to it. If other teams are affected, they should be included in the action team. Davidson [9] recommends that researchers and practitioners are motivated, but only motivation is insufficient; they must be able to conduct it, given their daily work, the scope of their industrial project, or competence. When it is impossible to perform an intervention, we can pivot to design science research and focus on the artifact instead, as we did in one of our research studies [18].

4.4. Planning, conducting and analyzing

Since the host organization engages in various activities, including organizational changes, it is essential to distinguish between interventions and routine operations.

Guideline 10: The scope of the intervention must be specified up-front based on theories.

Although interventions are atomic, their scope must be thoroughly planned and based on existing theories. Otherwise, we cannot contribute to the existing body of scientific knowledge. We must set the stop criteria and ensure we reach them. The same is true for the deliverables – they also need to be defined up-front, and their quality goals must be specified beforehand. We need to know when the intervention was successful and when it was not. Davidson and Baskerville et al. recommend having a solid theoretical backing of actions, but in software engineering, we can also identify new theories based on empirical observations. If so, our initial theory must be formulated upfront, validated, refined, and changed during the action research study.

Changes in ways of working that lack such planning cannot be considered interventions in action research. Properly distinguishing these activities ensures the validity and reliability of the research outcomes – if we do not have a baseline, we do not know if we improved. Additionally, this approach helps maintain clarity and focus on the research objectives, preventing confounding planned interventions with everyday business activities, hence our next guideline:

Guideline 11: Every intervention must be compared to a baseline, so ensure that there is one.

Inspired by the recommendation to plan data collection methods by Baskerville et al. [8], we can go one step further. We must establish a proper baseline for comparison. For instance, when introducing a new method, ensure that data from the old method is available for comparison. This may require collecting data before initiating the intervention. We should focus on quantitative and qualitative data in this data collection. The first provides us with facts, whereas the second allows us to get a deeper understanding of the reasons behind the phenomena observed in the quantitative data.

If pre-intervention data collection is necessary, we should be prepared to mitigate the Hawthorne effect [19]. We collect the data based on the theoretical models we bring to the project (see Guideline 2). For example, when we want to validate the effect of introducing modern code reviews, we should measure the performance of code reviews before this introduction in terms of speed, duration, and number of reviews. We also need to be prepared to adjust if the data that is needed cannot be collected:

Guideline 12: Explicitly plan for the intervention and data collection based on the theoretical model.

Since organizations are constantly changing, ensure that when the intervention is done, there are no confounders like reorganizations, new team onboarding, product changes, or other events that could affect the result. If they are, be prepared to document these events and study their effects or pivot (if the situation for action research is no longer optimal).

Our data collection methods must originate from the theories we apply in the action research. We must follow a theoretical model, as Davidson [9] points out. Measurements and data collection are essential parts of such a model.

Nevertheless, we must use the knowledge we gain in the diagnosing phase and pre-defined theories.

Guideline 13: Design and plan interventions based on causes identified in the diagnosing phase.

We can do it in two ways: either select the theory that aligns with the needs identified in the diagnosing phase or complement the existing theory with concepts and relations identified in the diagnosing phase. Davidson [9] recommends basing actions on the diagnosis, but we should try to reconcile both theories and empirical findings in the diagnosing phase.

Unlike other research methodologies, adjusting the study's setup during each cycle allows for significant flexibility. The action team can introduce new theories (e.g., from processes to products), change the scope of the cycle (e.g., from methods to tools), adjust the team (e.g., introduce new roles), or re-design the intervention so that it fits both the organization and the theory.

Sometimes, it is difficult to conduct action research studies for practical reasons. We may introduce harm to the organization by revealing some information or making a change that will cause problems for the company's operations. Therefore, I recommend also the following:

Guideline 14: Be ready to stop the intervention if it causes harm to the organization

Since the outcome of research activities is always unknown, we take the risk that the activity does not go according to plan. Therefore, we must, at all costs, prevent damage or harm to the organization, its employees, and its business. Otherwise, we do not follow the Nuremberg convention that all scientists should comply with – "Do no harm." This also means that we must have a contingency plan in place.

The contingency plan means we must continuously keep track of our collected data and how to address the research questions using it. We can also prepare alternative research questions to address using the data we have to maximize the chance of reporting results. We should remember that even negative or inconclusive results are of certain value to the research community. Additionally, publishing negative experiences, if they are generalizable, helps inform others of the risks encountered, contributing to a broader understanding and awareness within the field. This approach enhances the learning process and builds a repository of knowledge that can guide future interventions. Furthermore, documenting these experiences fosters a culture of transparency and continuous improvement within the organization.

When we engage in action research, it is easy to forget that sometimes we must change the topic or even stop the study. Therefore, my next guideline comes directly from my experiences in measurement system development, which was conducted as an action research study.

Guideline 15: The management and the action team should continuously assess the need for further studies.

We must remember that action research is a collaborative endeavor, requiring continuous involvement from the host organization's management to define and potentially finalize the studies. Depending on the specific context and goals of the research, the process might conclude after two or three cycles or extend to ten or more cycles.

Long-lived action research projects can address multiple challenges and test several theories. A good action research team does not happen often, but when it does, it usually lasts for many iterations. However, there is a danger with long-lived, static action research teams; they can get stuck in solving no longer important problems. Conducting a post-mortem workshop after the intervention is always a good idea to understand what to do next and how. If the change is not possible or needed by the host organization, the action team can find other problems to address, change the team's setup, or even change organizations [11].

4.5. Reporting

We often think that research must be conducted from start to finish to generate knowledge. In action research studies, however, even intermediate results are important. Since action research is done in cycles, we should be able to present the results from each cycle independently from each other as well as in the context of each other. Hence, the next guideline:

Guideline 16: Package intermediate results in a reusable way.

We must ensure that our results, tools, and methods are reusable, making them accessible to the reference team, other teams within the host organization, or even other organizations (see Davidson's fifth recommendation [9] about the impact of the action research projects). Researchers must remember that the value for the company is not found in a published paper, confusion matrix, or statistics alone. The real value for the host organization lies in tangible and actionable improvements, new products, features, increased operational efficiency, or better architectures. This focus on practical outcomes ensures that the research has a meaningful and lasting impact on the organization's success. It is often these kinds of tangible results that the research community values the most.

For the organization, the results of the action research cycles must be documented and disseminated even more frequently. Immediate publication on internal forums, web pages, podcasts, and webinars helps to spread good practices. Therefore, the action team should follow the next guideline to make communication efficient.

Guideline 17: The results should be continuously documented for academia and industry.

This internal documentation should include hands-on guides on how to apply the new knowledge, contact points for internal expertise, and locations of stored information from the action research. Action research projects often develop tools and instruments specifically configured for the host organization's infrastructure, which must be readily accessible internally. This approach ensures that the practical benefits of the research are fully realized and integrated within the organization. Davidson [9] recommends timely documentation of the research results, although it does not distinguish between the industrial and scientific audience.

When conducting action research, it's crucial to communicate the actions and results to management and the reference group. They must know if the actions provide value, and we need to know if the actions are going according to the plan. We must remember that our work should be about generating new knowledge, not conducting activities—the activities are a means to achieve the results.

Defining the context of the research is almost as important as the results. In action research, describing the organization by its name is tempting, but most often, it should be complemented with more details. Describing the organization's characteristics is more important, e.g., [20]. If the action team has to choose whether to reveal the organization's name or its characteristics, the characteristics provide more value to other researchers.

Guideline 18: Characterize the context of the study, including undertaken actions/interventions.

We should include as much relevant information about the context as possible when generalizing empirical and action research studies. In particular, such information as:
- Type of the organization and its process – e.g., web development organization working according to agile principles.
- Size and other characteristics of the product/service – e.g., 1,000,000 LOC written in C#.
- Context of the project – e.g., part of an internal reorganization or a larger research project.
- Host organization – e.g., the software development team of 20 persons.

This way of conveying characteristics helps others identify whether their context is similar or different from the reported study.

4.6. Ethical considerations for action research studies

Our ethical stance should be to "do no harm" when researching. In action research, this includes two aspects: direct harm to people (and, by extension, the organization) and indirect harm through new insights (and, by extension, products). This is included in Guideline 14: Be ready to stop the intervention if it causes harm to the organization.

We have access to the data of the host organization, which means that we can also access information that was not meant for the purpose of this study or even information that was part of the study but is not meant to be spread. This can include private conversations, e-mails, or commit messages containing sensitive information. Revealing such sensitive information can harm individuals, as they can lose their roles or even jobs. This can also be detrimental to the organization's business as the customers may lose trust in the company's products and services. Therefore, we should always consult the company's management and legal department before spreading information that could be sensitive – both internally and externally.

At the same time, we must recognize that our products and knowledge can potentially harm organizations and individuals. Our aim should always be to foster improvement rather than cause problems. When our findings pertain to individual improvements, we should direct our results to those individuals, inviting them to knowledge-sharing sessions and seminars. When presenting such results in a larger forum, it is essential to anonymize the data by removing names and identifying details of individuals or teams. We can emphasize best practices and effective working methods, but we must avoid singling out specific individuals or organizations.

Guideline 19: Reflect on the results objectively and avoid overinterpretation.

One of the main problems that I observed in action research collaborations is the tendency to overinterpret the results. Both researchers and practitioners tend to generalize the results too quickly. The fact that an improvement worked for one organization does not guarantee it will work for another. Overinterpreting the results is ethically problematic because we are in danger of speculation – we do not know if the same methods would work elsewhere. Both Davidson [9] and Baskerville et al. [8] warn about this too.

When reflecting on the results, it is better to follow the next guideline:

Guideline 20: Interpret the results based on theory

The theoretical foundation of action research cycles helps us ensure objective results. When we specify the theory upfront, we use it to interpret the results, which helps avoid overinterpretation, speculation, or even the Hawthorne effect.

5. Conclusions

Action research has been identified as one of the upcoming research methods in the last two decades. An increasing number of software engineering papers have been published in which this research methodology has been used, which indicates that there is a growing interest in it. The rise of industrial software engineering research, pioneered by large software development companies, nourishes this methodology.

This paper identified 20 guidelines for conducting action research studies in the context of academia-industry collaboration. Although these guidelines are probably relevant to industrial action research (in vivo action research), they have not yet been validated in that context. The guidelines should be seen as the basics and either a reference for action research or a starting point for designing such studies. They can be complemented by studying existing articles about action research, e.g., Kantola et al. [21] or Natarajan et al. [22] and Kemell et al. [23].

The guidelines presented in this paper are specifically targeted towards software engineering, i.e., the discipline of development of software and the development of the processes for developing software. What makes software engineering specific in this context is the fact that software development is the main context, not the use of IT technology or software in other domains, which is often the case of studies in information systems; it is the information systems research where action research historically has been very popular.

However, despite the growing popularity of action research, it is not for every type of research. We still need to use experimentation, case studies, and surveys to understand the fundamental phenomena of software engineering, to study industrial cases in software engineering, or to understand the spread of phenomena in the entire community. It's important to use the best methods and tools suited for the task and not revive to a specific one all the time. Action research works best when it is done in collaboration between academia and industry and when the entire action team can be embedded in the host organization.

Acknowledgments

I would like to thank Software Center, a collaboration between five universities and 17 companies, for providing and financing the context of this work. I also thank the anonymous reviewers for constructive, actionable feedback.

CRediT authorship contribution statement

Miroslaw Staron: Conceptualization, data collection, analysis, writing.

Declaration of competing interest

I declare that I have no competing interests.

Funding

This paper has been partially sponsored by Software Center, www.software-center.se

References

[1] R.L. Glass, "The software-research crisis," *IEEE Software*, Vol. 11, No. 6, 1994, pp. 42–47.
[2] C. Wohlin, P. Runeson, M. Höst, M.C. Ohlsson, B. Regnell et al., "Experimentation in software engineering: An introduction," *The Kluwer International Series In Software Engineering*, 2000.
[3] P. Runeson and M. Höst, "Guidelines for conducting and reporting case study research in software engineering," *Empirical Software Engineering*, Vol. 14, 2009, pp. 131–164.
[4] R.J. Wieringa, *Design science methodology for information systems and software engineering.* Springer, 2014.
[5] P. Ralph, N. bin Ali, S. Baltes, D. Bianculli, J. Diaz et al., "Empirical standards for software engineering research," *arXiv preprint arXiv:2010.03525*, 2020.
[6] P.S.M. dos Santos and G.H. Travassos, "Action research use in software engineering: An initial survey," in *2009 3rd international Symposium on empirical software Engineering and measurement.* IEEE, 2009, pp. 414–417.
[7] P.S.M. Dos Santos and G.H. Travassos, "Action research can swing the balance in experimental software engineering," in *Advances in computers.* Elsevier, 2011, Vol. 83, pp. 205–276.
[8] R.L. Baskerville, "Investigating information systems with action research," *Communications of the Association for Information Systems*, Vol. 2, No. 1, 1999, p. 19.
[9] R. Davison, M.G. Martinsons, and N. Kock, "Principles of canonical action research," *Information Systems Journal*, Vol. 14, No. 1, 2004, pp. 65–86.
[10] I. Bleijenbergh, J. van Mierlo, and T. Bondarouk, "Closing the gap between scholarly knowledge and practice: Guidelines for HRM action research," *Human Resource Management Review*, Vol. 31, No. 2, 2021, p. 100764.
[11] M. Staron, *Action research in software engineering.* Springer, 2020.

[12] Q. Song and P. Runeson, "Industry-academia collaboration for realism in software engineering research: Insights and recommendations," *Information and Software Technology*, Vol. 156, 2023, p. 107135.

[13] P. Runeson, "It takes two to tango – an experience report on industry–academia collaboration," in *Fifth International Conference on Software Testing, Verification and Validation.* IEEE, 2012, pp. 872–877.

[14] Y. Dittrich, K. Rönkkö, J. Eriksson, C. Hansson, and O. Lindeberg, "Cooperative method development: Combining qualitative empirical research with method, technique and process improvement," *Empirical Software Engineering*, Vol. 13, 2008, pp. 231–260.

[15] K. Petersen, C. Gencel, N. Asghari, D. Baca, and S. Betz, "Action research as a model for industry-academia collaboration in the software engineering context," in *Proceedings of the 2014 international workshop on Long-term industrial collaboration on software engineering*, 2014, pp. 55–62.

[16] M. Staron and W. Meding, *Software development measurement programs*, Vol. 10. Springer, 2018.

[17] C. Wohlin and P. Runeson, "Guiding the selection of research methodology in industry – Academia collaboration in software engineering," *Information and Software Technology*, Vol. 140, 2021, p. 106678.

[18] M. Staron, J. Strom, A. Karlsson, and W. Meding, "Using generative AI to support standardization work – The case of 3GPP," *International Conference on Product-Oriented Software Process Improvement, PROFES*, 2024.

[19] P. Sedgwick and N. Greenwood, "Understanding the Hawthorne effect," *The BMJ*, Vol. 351, 2015.

[20] M. Staron and W. Meding, "Mesram – A method for assessing robustness of measurement programs in large software development organizations and its industrial evaluation," *Journal of Systems and Software*, Vol. 113, 2016, pp. 76–100.

[21] K. Kantola, J. Vanhanen, and J. Tolvanen, "Mind the product owner: An action research project into agile release planning," *Information and Software Technology*, Vol. 147, 2022, p. 106900. [Online]. https://www.sciencedirect.com/science/article/pii/S0950584922000611

[22] T. Natarajan and S. Pichai, "Behaviour-driven development and metrics framework for enhanced agile practices in scrum teams," *Information and Software Technology*, Vol. 170, 2024, p. 107435. [Online]. https://www.sciencedirect.com/science/article/pii/S0950584924000405

[23] K.K. Kemell, A. Nguyen-Duc, M. Suoranta, and P. Abrahamsson, "Startcards – a method for early-stage software startups," *Information and Software Technology*, Vol. 160, 2023, p. 107224. [Online]. https://www.sciencedirect.com/science/article/pii/S0950584923000782

Authors and affiliations

Miroslaw Staron
e-mail: miroslaw.staron@gu.se
ORCID: https://orcid.org/0000-0002-9052-0864
Computer Science and Engineering,
University of Gothenburg and Chalmers University of
Technology, Sweden

e-Informatica Software Engineering Journal
Volume 19, Issue 1, pages: 1–32
DOI: 10.37190/e-Inf250106

BibTeX

A Novel Approach for Enhancing Code Smell Detection Using Random Convolutional Kernel Transform

Mostefai Abdelkader*, Mekour Mansour
*Corresponding author: abdelkader.mostefai@univ-saida.dz

Article info

Dataset link: https://zenodo.org/records/15602620

Keywords:
Code smell
Maintenance
Classification
Random convolutional transform
Deep learning

Submitted: 05 Sep. 2024
Revised: 20 Jun. 2025
Accepted: 7 Jul. 2025
Available online: 8 Jul. 2025

Abstract

Context: In software engineering, the presence of code smells is closely associated with increased maintenance costs and complexities, making their detection and remediation an important concern.
Objective: Despite numerous deep learning approaches for code smell detection, many still heavily rely on feature engineering processes (metrics) and exhibit limited performance. To address these shortcomings, this paper introduces CSDXR, a novel approach for enhancing code smell detection based on Random Convolutional Kernel Transform – a state-of-the-art technique for time series classification. The proposed approach does not rely on a manual feature engineering process and follows a three-step process: first, it converts code snippets into numerical sequences through tokenization; second, it applies Random Convolutional Kernel Transform to generate pooled models from these sequences; and third, it constructs a classifier from the pooled models to identify code smells.
Method: The proposed approach was evaluated on four real-world datasets and compared against four state-of-the-art methods – DeepSmells, AE-Dense, AE-CNN, and AE-LSTM – in detecting Complex Method, Multifaceted Abstraction, Feature Envy, and Complex Conditional smells.
Results: Empirical results demonstrate that CSDXR outperformed the four state-of-the-art methods – DeepSmells, AE-Dense, AE-CNN, and AE-LSTM – in detecting Complex Method and Multifaceted Abstraction smells. Specifically, the enhancement rates in terms of F_1-score were 1.99% and 6.09% for Complex Method and Multifaceted Abstraction smells, respectively. In terms of MCC, the improvement rates were 0.82% and 35.64% for these two smells, respectively. The results also show that while DeepSmells achieves superior overall performance on Feature Envy and Complex Conditional smells, CSDXR surpasses AE-Dense, AE-CNN, and AE-LSTM in detecting these two types of smells.
Conclusions: The paper concludes that the proposed approach, CSDXR, demonstrates significant potential for effectively detecting various types of code smells.

1. Introduction

Software plays an increasingly pivotal role in many aspects of modern life. As these systems become more complex and reliance on them continues to grow, maintaining them becomes ever more critical. To maintain their expected value, software systems require regular upkeep. In a highly competitive environment, developers often employ design and implementation strategies aimed at speeding up time-to-market. However, such practices can exacerbate technical debt [1], which refers to the long-term costs associated with suboptimal design and implementation decisions. While these decisions may offer immediate benefits, such as faster product releases or enhanced client satisfaction, they often undermine the software's quality and lead to costly future maintenance.

Code smells are indicators of poor code design that contribute to technical debt. They manifest in various parts of the code, such as classes or method statements, and arise from inadequate design or implementation choices. These decisions can be intentional, where developers are aware of the trade-offs, or unintentional [2, 3]. Extensive research has highlighted the detrimental impact of code smells on software quality, identifying them as a significant manifestation of technical debt [2] and emphasizing the need for effective detection, filtration and prioritization approaches [4, 5].

Manual detection of code smells is challenging [6], prompting the development of various automatic detection techniques. These methods are generally categorized into deep learning, machine learning, heuristics, and metrics-based approaches [4, 7, 8]. Metric-based and heuristic-based approaches are popular but often rely on costly manual processes involving designed heuristics and selected features.

These processes require significant manual intervention, including configuring and customizing analysis tools to suit specific needs, determining which code aspects to measure, selecting the appropriate metrics, setting thresholds, and fine-tuning these thresholds for specific contexts and projects. Additionally, interpreting the results of these metrics to classify a piece of code as a "smell" requires domain expertise, making the task labor-intensive.

Consequently, these manual interventions make the processes time-consuming and costly, highlighting the inherent limitations of such approaches.

Machine learning methods, on the other hand, depend on external tools to compute features (e.g., metrics) from the source code, making their effectiveness contingent upon these tools and the expert-defined features. However, different tools can yield varying results for the same metric, even when the metric is conceptually the same (e.g., lines of code or cyclomatic complexity). These discrepancies arise from variations in how each tool defines, computes, or interprets the metric. Since tools may apply slightly different algorithms, rules, or default settings, the values they produce can differ.

For example, tools calculating cyclomatic complexity may handle control flow constructs, such as loops or exception handling, differently, leading to variations in the final complexity score. Similarly, tools measuring lines of code (LOC) might differ in their definitions of what constitutes a line.

Additionally, variations in results can stem from the parsers or lexers used. Each parser may interpret code differently based on how it handles syntax, grammar, or language-specific rules, which in turn affects the features extracted for metric computation.

In the absence of a standardized tool, these differences in the tools used can significantly impact the results, and consequently, the performance of code smell detectors.

Despite the variety of existing techniques, many remain underdeveloped and ineffective [3, 4, 7, 9]. Thus, there is a pressing need for advanced techniques to enhance code smell detection models. Recent studies have explored deep learning models that minimize manual feature engineering by automatically learning features from source code [7, 9]. However, these models face limitations, including specificity to particular code smells and overall limited performance [7, 9, 10]. Xu and Zhang [10] argue that these limitations stem from token-based representations of code, which lose rich semantic and structural information. Nevertheless, we believe that deep learning models can still effectively extract meaningful representations from raw token sequences for code smell detection, as demonstrated by Ho et al. [9]. We propose that advanced time series classification (TSC) techniques could address these shortcomings.

The TSC field has seen significant advancements, with numerous techniques for time series representation and classification achieving success in various domains, including finance, Internet of Things, cloud computing, energy, transportation, code clone detection and social networks [11–16]. We propose that source code can be converted to an ordered sequence of tokens and treated as a time series. By leveraging TSC algorithms, we aim to improve code smell detection, inspired by their success in code clone detection [16].

Our thesis is that in kernel-based approaches such as XRocket (e.g., MiniRocket and Rocket), kernels serve as tools for detecting specific patterns by convolving them over the time series. The result of convolving each kernel is an activation map that indicates the location and strength of this pattern. A pooling operator then is used to summarize this activation map into a single feature (i.e., a single number). With n kernels and m pooling operators, a time series representation consisting of $n \times m$ features is created. The resulting representations is then used to train a classifier to classify new instances. Consequently, for a code smell characterized by identifiable patterns, it becomes feasible to detect such smells using carefully designed kernels, appropriate pooling operators, and a suitable classifier. This detection process works on time series data derived from source code to be checked for "smelliness."

Drawing inspiration from the success of TSC methods and guided by our thesis, we introduce CSDXR, a novel approach for code smell detection based on MINImally RandOm Convolutional KErnel Transform (MiniRocket) and RandOm Convolutional KErnel Transform (Rocket). Our approach hypothesizes that using MiniRocket or Rocket to pool representations of code snippets will outperform previous methods. CSDXR converts source code snippets into time series, then uses MiniRocket (CSDMR) or Rocket (CSDR) to pool representations. A classifier is then trained on these representations, labeled as either smelly or non-smelly, and used to predict the presence of code smells in new source code. The proposed CSDXR method does not rely on a manual feature engineering process.

Main Contributions:

– Introduction of a novel method does not rely on feature engineering process for code smells detection.
– Introduction of a novel method based on MiniRocket and Rocket for modelling source code.
– Evaluation of the method's effectiveness in detecting four specific code smells: Complex Method, Complex Conditional, Feature Envy, and Multifaceted Abstraction.

The rest of this paper is organized as follows: Section 2 provides background on code smells, Rocket, and MiniRocket. Section 3 reviews the state-of-the-art code smell detection approaches. Section 4 details our proposed approach. Section 5 presents our empirical study, Section 6 presents and discuss the results of the empirical study. Section 7 addresses threats to validity, and Section 8 concludes the paper and outlines directions for future work.

2. Background

This section provides the background information necessary for understanding the approach.

2.1. Code smell

The term *code smell* as first introduced by Kent Beck in the 1990s [17]. Code smells are indicators of poor code quality in various code elements such as classes, methods, or statements, and they often lead to increased technical debt. These smells typically signal violations of design principles and best practices, arising from suboptimal design and implementation decisions.

The concept of code smells gained wider recognition through Martin Fowler's book, which detailed 22 types of code smells and their corresponding refactoring solutions [17]. Examples of common code smells include Feature Envy, God Class, Duplicated Code, Long Method, Long Switch, and Long Parameter List. For more comprehensive information about code smells, refer to [17, 18].

In this paper, we evaluate the proposed approach on four specific code smells [18]:
- **Complex Method (CM):** A method characterized by high cyclomatic complexity.
- **Complex Conditional (CC):** A conditional statement with a complex condition expression (e.g., an intricate if statement).
- **Feature Envy (FE):** A method that is more interested in the details of a different class than the class it is in.
- **Multifaceted Abstraction (MA):** A class that has multiple, unrelated responsibilities.

Complex Method and Complex Conditional are implementation-level smells, while Feature Envy and Multifaceted Abstraction are design-level smells.

2.2. Rocket and MiniRocket

Time series data consists of ordered sequences, such as temporal data, where each data point is associated with a specific time. Time Series Classification (TSC) involves predicting the class of a time series based on previously classified series. According to Bagnall et al. [11], the order of attributes in time series data differentiates TSC from traditional classification problems, necessitating that the representation process creates discriminative and meaningful features by accounting for this temporal structure.

Two state-of-the-art techniques for time series classification are the RandOm Convolutional KErnel Transform (ROCKET) [19] and its variant, MINImally RandOm Convolutional KErnel Transform (MiniROCKET) [20]. Both methods are inspired by convolutional neural networks (CNNs) but differ in their approach to kernel generation and application.

The MiniRocket and Rocket methods compute a representation of a time series by first convolving it with a set of k kernels. In the case of Rocket, these kernels are randomly generated, whereas in MiniRocket, they are designed based on predefined rules.

Second, the activation map, which results from the convolution of each kernel with the time series, is summarized using pooling operators. Rocket utilizes two pooling operators, Proportion of Positive Values (PPV) and Max, to generate two features per kernel, resulting in a feature vector with $2k$ features. In contrast, MiniRocket uses only the PPV operator, producing a representation with k features.

ROCKET uses a large set of randomly generated convolutional kernels, typically 10,000 by default. These kernels vary in length and dilation, and are employed to transform the input time series into a feature vector.

The kernel initialization in ROCKET follows a random process with the following parameters:

1. **Kernel Length (l)**: Randomly selected from $\{7, 9, 11\}$.
2. **Kernel Weights (w)**: Randomly initialized from a normal distribution.
3. **Bias Term (b)**: Added to the result of the convolution operation.
4. **Dilation (d)**: Determines the spread of the kernel weights over the input time series. Dilation allows similar kernels with different dilation values to detect patterns at various frequencies and scales. For example, a kernel [2 −1 1] with $d = 1$ becomes [2 0 −1 0 1], and with $d = 3$, it becomes [2 0 0 0 −1 0 0 0 1].
5. **Padding (p)**: Adds zeros to the start and end of the input series to ensure the activation map and the input series are of the same length.

The result of applying a kernel ω with dilation d to a time series T at offset i is defined as:

$$T_{i:(i+l)} * w = \sum_{j=0}^{l-1} T_{i-\left(\left\lfloor \frac{m}{2} \right\rfloor \times d\right)+(j\times d)} \times w_j \tag{1}$$

MiniROCKET is a variant of ROCKET that retains the core principles but is designed to be more computationally efficient. It uses a smaller number of kernels, reducing the computational burden while maintaining performance [20].

MiniROCKET utilizes a set of predefined kernels with a fixed length of 9 and two possible weight values $\{-1, 2\}$, applying 84 fixed convolutions. Unlike ROCKET, which computes two features per kernel, MiniROCKET computes only one feature per kernel. These design choices make MiniROCKET significantly faster – up to 75 times – compared to ROCKET, while maintaining performance comparable to other models.

3. State of the art

Many approaches have been proposed to detect smells in software systems, classified into deep learning [3, 21], machine learning [8], heuristics, and metrics-based methods [4, 7–10, 22].

3.1. Metrics-Based Smell Detection Methods [21, 23]

Metrics-based approaches, in particular, are widely used for code smell detection. Software metrics are a common method for assessing software quality, evaluating factors and attributes such as cohesion and coupling within a system [24]. A clean codebase typically exhibits metric values within ranges defined by experts [Livre Software Quality], reflecting adherence to software design principles and best practices.

Code fragments are considered smelly if they violate these principles. Such violations can be identified by measuring the design attributes of the code fragment and comparing them to values from clean (non-smelly) code. For example, the Feature Envy smell is an indicator of poor cohesion and coupling [17]. Metrics-based approaches detect smells by applying formulas that use filters and thresholds on a set of metrics computed from the source code [25]. For instance, a God Class smell can be detected using metrics such as

ATFD (Access to Foreign Data), WMC (Weighted Methods per Class), and TCC (Tight Class Cohesion) [26].

An example of such formulas is defined by Marinescu [27] for detecting ten different code smells. Macia et al. [28] proposed thresholds and formulas that combine eight metrics for detecting aspect-oriented smells. Fard and Mesbah [29] introduced a method called JSNOSE to detect JavaScript code smells, which is metrics-based and combines static and dynamic analysis. Chen et al. [30] defined ten code smells specific to the Python language and proposed a metrics-based method to detect them. Their study utilized and compared thresholds specified by three filtering strategies: the Experience-Based Strategy, the Statistics-Based Strategy, and the Tuning Machine Strategy. This research was conducted on a dataset of 106 Python projects.

3.2. Rules/heuristic-based smell detection methods

In this category, the method's input is a source code model and, optionally, a set of software metrics. Detection is performed using a set of predefined rules or heuristics. These methods rely on specified rules or heuristics and leverage source code models, and optionally metrics, for detecting code smells and, principally, design smells [18, 31]. Rule-based approaches depend on manually specified rules [32]. For instance, DÉCOR [27] relies on expert-designed rules, which must be expressed in a domain-specific language. However, this design process is costly. The DÉCOR approach was validated on the software XERCES v2.7.0.

3.3. Machine learning-based smell detection

In this category, classifiers such as Support Vector Machines (SVM) or Naïve Bayes (NB) are trained on datasets specific to a particular smell. The dataset typically consists of computed models (representations) of code fragments. Once trained, these models are used to predict the class of new code fragments (i.e., whether they are smelly or not). Metrics-based representations are commonly employed in these methods [4, 7, 10].

Maiga et al. [33] proposed an approach called SVMDetect to detect anti-patterns in software systems. This approach leverages SVM, a well-known machine learning algorithm. An empirical study conducted on three systems and four anti-patterns demonstrated that SVMDetect is more accurate than DETEX.

Khomh et al. [34] introduced a process to transform detection rules into a probabilistic model, with a demonstration conducted on the Blob anti-pattern.

Kreimer [35] proposed a method based on decision trees to detect design flaws (code smells) in object-oriented software.

3.4. Deep learning-based smell detection

Sharma et al. [7] proposed an approach for code smell detection based on a deep learning model that combines Convolutional Neural Networks (CNN), Recurrent Neural Networks (RNN), and autoencoder models. The authors built a dataset from 922 C# and 922 Java repositories downloaded from GitHub. The proposed approach aims to leverage the power of these models without relying on the feature engineering process commonly used by most code smell detection methods (e.g., metrics). The authors also investigated the potential of transfer learning in code smell detection by training the model on C# projects and evaluating the results on Java projects. The empirical study conducted to detect

Feature Envy, Complex Method, Complex Conditional, and Multifaceted Abstraction smells indicated that the results are smell-specific. This means there is no simple, universal solution for detecting all types of smells. The results also indicated that deep learning models without a feature engineering process still require improvement in performance.

Ho et al. [9] proposed an approach called DeepSmells to address the limitations of the method proposed by Sharma et al. [7]. DeepSmells incorporates both structural and semantic features from software units and mitigates the effects of imbalanced data distribution. To achieve this, the method combines Convolutional Neural Networks (CNN) with Long Short-Term Memory networks (LSTM) to learn hierarchical representations of source code, preserve semantic information, and improve the quality of context encoding. The output of this stage is fed into a deep neural network classifier with a weighted loss function to counteract the effects of skewed data distribution. Empirical studies demonstrated that this approach outperforms state-of-the-art tools.

Skipina et al. [36] evaluated and compared machine learning models using code representations based on metrics versus representations based on neural code embeddings (CodeT5 and CuBERT) for detecting Data Class and Feature Envy smells. The evaluation was conducted on the MLCQ dataset, and the results showed no significant differences in performance between the two approaches. However, code embeddings were found to be more scalable and have the potential to adapt to new programming languages.

Hadj-Kacem and Bouassida [37] proposed a combined method using deep autoencoders and Artificial Neural Networks (ANN) for detecting God Class, Data Class, Feature Envy, and Long Method smells. In this method, the autoencoder reduces data dimensionality, and the ANN is used as the classifier. The empirical study indicated that this method is effective, achieving an F-measure of 98.93% for the God Class code smell.

Liu et al. [38] proposed a deep learning-based method for detecting the Feature Envy code smell. The model employed is a Convolutional Neural Network (CNN), where the input is a combination of text and numerical data. The text, consisting of a sequence of the method's name, the class name, and the target class name, is embedded to produce a numerical representation. The evaluation, conducted on seven well-known open-source projects, showed that the method outperforms state-of-the-art tools, achieving an F-measure of 34.32%.

Bo Liu et al. [39] proposed an approach called **feTruth** aimed at improving deep learning models dedicated to detecting the Feature Envy code smell. This objective is achieved by filtering out false positives produced by state-of-the-art tools using a set of heuristics and a decision tree classifier.

Das et al. [40] proposed a deep learning method based on Convolutional Neural Networks (CNN) to detect Brain Class and Brain Method code smells.

Yu et al. [41] proposed a method based on Graph Neural Networks (GNN) for Feature Envy detection. The method leverages code metrics and calling relationships to address the challenge posed by calling relationships between methods, which can hinder the detection process. The evaluation on five open-source projects showed that the performance, in terms of F_1-score, was 37.98% higher than state-of-the-art tools.

Hanyu et al. [42] introduced a deep learning approach based on a Graph Convolutional Network (GCN) for Long Method detection. This model builds a graph neural network by inputting two types of information: nodes and edges. The nodes represent methods and statements, while the edges represent include, control flow, control dependency, and data dependency relationships. The evaluation was based on five groups of datasets.

Zhang et al. [43] proposed a method called **DeleSmell** that combines deep learning and Latent Semantic Analysis (LSA) to detect Brain Class and Brain Method code smells. The deep learning model consists of two branches: a Convolutional Neural Network (CNN) branch and a Gated Recurrent Unit (GRU)-attention branch. A Support Vector Machine (SVM) classifier is used at the final stage. The approach aims to address the issues of incomplete feature representation and unbalanced distribution between positive and negative samples. The evaluation was conducted on a dataset built from 24 real-world projects, with the dataset balanced using a refactoring tool developed for this purpose. The results indicated an improvement of 4.41% in F_1-score compared to state-of-the-art methods.

Xu and Zhang [10] proposed a method for detecting Feature Envy, Insufficient Modularization, Empty Catch Block, and Deficient Encapsulation code smells. The method is based on a deep learning model and Abstract Syntax Trees (ASTs) and does not rely on a feature engineering process. The objective was to overcome the limitations of token-based approaches by leveraging the semantic and structural information of the source code. The experimental results indicate its superiority compared to state-of-the-art approaches for detecting code smells.

Zhang and Dong [44] proposed the **MARS** approach for detecting Brain Class and Brain Method smells. The approach aims to solve the gradient degradation problem using an improved residual network. It employs a metric-attention mechanism to increase the weight value of important code metrics. The approach was evaluated on the BrainCode dataset, which was built from 20 real-world applications. The experimental results show that the average accuracy of MARS is 2.01% higher than state-of-the-art tools.

Li and Zhang [45] proposed a method to optimize code smell detection through a hybrid model with multi-level code representation. In this approach, the result is a function of two predictions at the syntactic, semantic, and token levels. The prediction at the syntactic and semantic levels is computed using a Graph Convolution Network (GCN) that takes as input the AST with control and data flow edges of the source code. The token-level prediction is calculated using a bidirectional Long Short-Term Memory (LSTM) network with an attention mechanism. Experimental results demonstrate that the method performs better in both single code smell detection and multi-label code smell detection cases.

Liu et al. [46] proposed a method based on Convolutional Neural Networks (CNN) and a text embedding technique (i.e., Word2Vec). The CNN model is fed a representation of a code fragment computed using the Word2Vec approach.

For further details, see [21, 23].

4. The proposed approach

In this section, we present an overview of our proposed method, CSDXR, for code smell detection based on the random convolutional transform method. As illustrated in Figure 1, CSDXR combines MiniRocket (CSDMR) or Rocket (CSDR) with advanced classifiers. The proposed method consists of three basic steps. First, the method converts a code snippet into a time series using a tokenization technique. In the second step, MiniRocket (or Rocket) is employed to generate a model of the obtained sequence. Finally, in the third step, CSDXR uses the pooled models to build a classifier for detecting code smells.

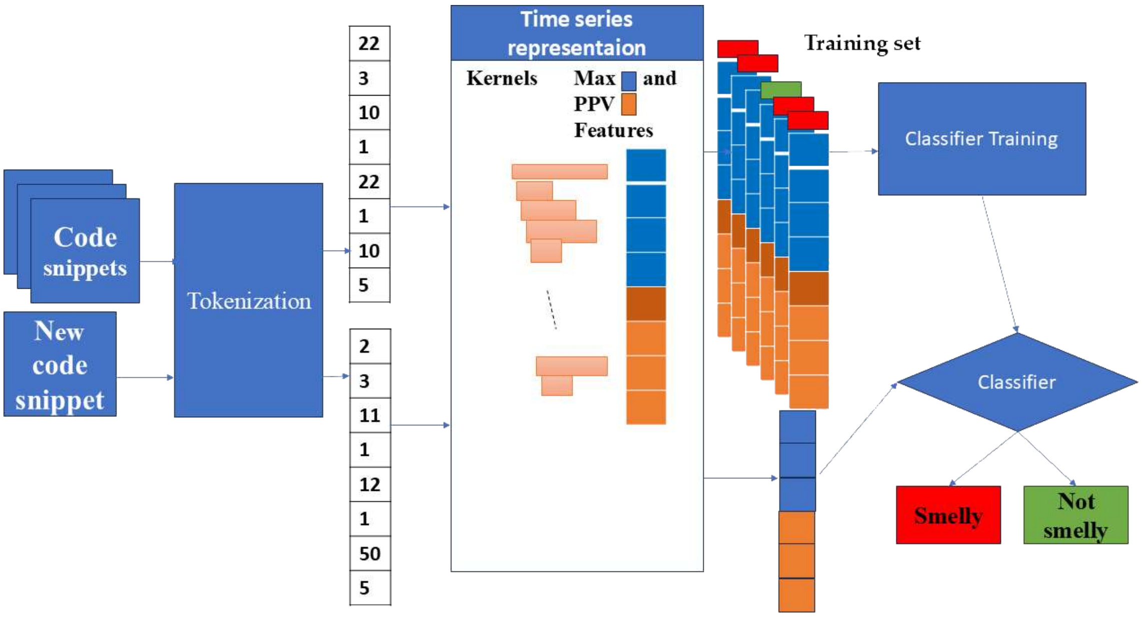


Figure 1. The proposed approach

4.1. Tokenization

The objective of this step is to convert a code snippet into a sequence of numbers that can be treated as a time series. The CSDXR method builds upon the tokenization algorithm provided by Sharma et al. [7]. Sharma et al. [7] released the full pipeline of their deep learning approach and encouraged researchers to extend it, aiming to fully explore the potential of code smell detection methods that do not rely on feature engineering.

The tokenization process works as follows: the code snippet is first decomposed into a sequence of tokens using a lexical analyzer. Each unique token is then assigned a specific number. For example, the token "{" might be assigned the number 123, the token "(" might be assigned the number 40, and so forth. An example of the result of the tokenization of a code snippet is illustrated in Figure 2.

The output of this step is a sequence of numbers (i.e., a time series).

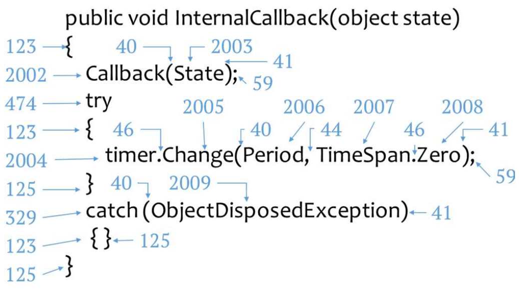

public void InternalCallback(object state)
123 {
40
2003
41
2002 Callback(State);
59
474 try
2005
2006
2007
2008
123 {
46
40
44
46
41
2004 timer.Change(Period, TimeSpan.Zero);
59
125 }
40
2009
329 catch (ObjectDisposedException)
41
123 { }
125
125 }

Figure 2. An example of the result of the tokenization of a code snippet [7]

4.2. Time series modelling

The objective of this phase is to create a representation of the code snippet from the time series produced in the previous step. This is achieved by employing the MiniRocket algorithm or a similar algorithm (e.g., Rocket).

The modelling process works as follows:

1. The steps involved in this process are collectively referred to as the representation phase. In this phase, first, a fixed set of k kernels is produced. Next, each kernel is convolved over the time series. The result of convolving each kernel is an activation map that indicates the location and strength of the pattern. Subsequently, pooling operators are used to summarize this activation map into a set of features. With n kernels and m pooling operators, a time series representation consisting of $n \times m$ features is created. For example, in the case of ROCKET, two pooling operators are used: PPV and MAX. The kernels are randomly generated from the set { 7, 9, 11 }, and their weights are randomly sampled from a normal distribution. In contrast, for MiniROCKET, this process is quasi-deterministic, and only one pooling operator is used (PPV). The kernels have a size of 9, and their weights are randomly selected from the set {-1,2}. For example, the application of the kernel $\mathbf{w} = [-1, 0, 1]$ to the sequence $[0, 1, 3, 2, 9, 1, 1, 15, 4, 9]$ is illustrated in Figure 3. This Figure shows that the obtained activation map by convolving the kernel $\mathbf{w} = [-1, 0, 1]$ over this time series is $[3, 1, 6, -1, -8, 14, 3, -6]$. The feature obtained using the PPV operator is $5/8$, and for the MAX pooling operator, it is 14. Finally, all the features obtained by convolving the k kernels are concatenated to form the time series model.

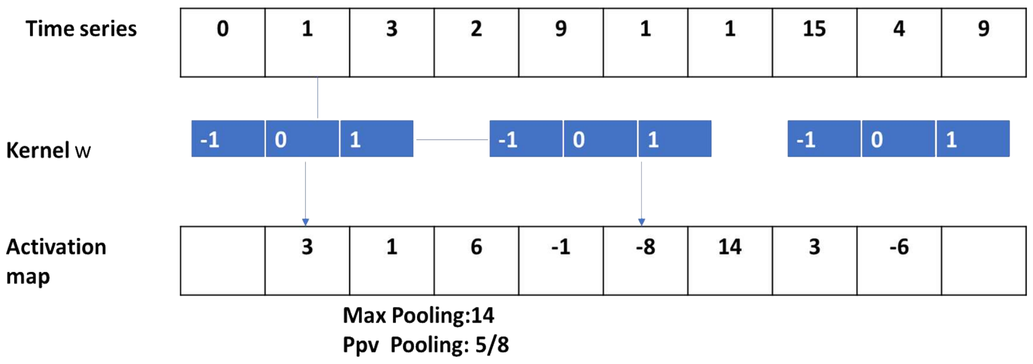


Figure 3. An example of sequence transformation

4.3. Classifier learning

In this step, the feature vectors representing smelly and non-smelly code snippets are used to train a classifier to differentiate between smelly and non-smelly code fragments. Various classifiers have been proposed for this purpose, and this step can be fulfilled using any of these classifiers.

In this paper, we employ the following classifiers: Naïve Bayes, Decision Tree, Logistic Regression, Random Forest, and XGBoost. The trained classifier model is then used to determine whether a new code snippet is smelly or not.

5. Empirical study

This section presents an empirical study on the use of CSDXR for code smell detection. The aim of this experiment is to evaluate the performance of CSDXR in detecting code smells. Specifically, the study addresses the following research objectives and questions:

5.1. Research objectives

The goal of this research is twofold:
1. To explore the feasibility of applying state-of-the-art time series representation methods in the context of code smell detection.
2. To investigate the effectiveness of these representations when used with advanced classifiers.

Based on these goals, the study aims to answer the following research questions:

RQ1: How do variations in the configuration of CSDXR, specifically using MiniRocket and Rocket transformations combined with advanced and standard classifiers, affect the prediction performance in detecting code smells?

We use MiniRocket, Rocket, standard and advanced classifier models in this exploration. MiniRocket and Rocket are fed with time series representing source code snippets. The output is then used with standard and advanced classifiers such as Naïve Bayes, Logistic Regression and XGboost.

Hypothesis 1: It is feasible to detect code smells using classifiers trained on representations pooled by well-configured MiniRocket or its variants from time series representing source codes. The rationale behind this hypothesis is that prior research in time series classification suggests that MiniRocket and Rocket transformations yield distinct feature representations, while classifier choices further modulate performance. Exploring these combinations helps identify optimal configurations for detecting code smells.

RQ2: How does the CSDXR method compare to state-of-the-art baseline tools (DeepSmells, AE-Dense, AE-CNN, and AE-LSTM) in terms of classification metrics such as Precision, Recall, F_1-score, and MCC? Are the performance differences in term of F_1 and MCC statistically significant?

We evaluate how well the CSDXR method performs in comparison to four baseline models presented in Section 4.3.

Hypothesis 2: The CSDXR method can improve the performance of code smell detection. This hypothesis is justified by our thesis that a source code snippet can be viewed as a time series and that Rocket and its variants, including MiniRocket, have proven to be powerful techniques for time series classification. By leveraging these methods, we aim to enhance the effectiveness of detecting code smells.

RQ3: How do the performance (in terms of Precision, Recall, F_1-score, and MCC) and computational cost (transformation time) of CSDXR models with MiniRocket compare to those with Rocket? Are the performance differences in terms of F_1, MCC and transformation time statistically significant?

In this exploration, we replace the MiniRocket-based transformation (CSDXR, corresponding to CSDMR) with the Rocket-based transformation (CSDR). The rationale for this change is supported by existing literature that shows comparable performance of these two transformations in other domains. Since computational cost is a critical factor when selecting the appropriate transformation method in practical applications [19, 20], this question investigates the efficiency of the two designs of the CSDXR method: the

MiniRocket-based CSDMR and the Rocket-based CSDR, with both variants being derived by varying the classifiers used.

Hypothesis 3: We hypothesize that the performance of CSDMR variants (based on MiniRocket) is comparable to CSDR variants (based on Rocket) in the context of code smell detection, with CSDMR variants being faster than CSDR variants. Both variants differ in their classifier selection, which influences their performance and computational efficiency.

This hypothesis is supported by literature showing that MiniRocket and Rocket have comparable performance in other domains and that MiniRocket is faster than Rocket.

To answer RQ1, RQ2, and RQ3, the CSDXR method was trained on a training set and subsequently evaluated on a test dataset. This evaluation used a dataset curated by Sharma et al. [7]. Details of the dataset are provided in the following section.

5.2. Datasets

We conduct our experiments on datasets containing four types of code smells[1]: Complex Method (CM), Complex Conditional (CC), Feature Envy (FE), and Multifaceted Abstraction (MF). Notably, the last two smells, Feature Envy and Multifaceted Abstraction, are particularly challenging to detect [7]. These datasets were curated by Sharma et al. [7] and have been utilized in other studies [10].

The datasets are composed of a total of 416,445 instances, with the following breakdown:

1. **Number of Smelly Instances:** 20,753
2. **Number of Non-Smelly Instances:** 395,692

These datasets are imbalanced, with an average imbalance rate of 4.21%, meaning that on average, positive instances make up around 4.21% of the total instances for each smell. Table 1 presents the Statistics of the Datasets.

Table 1. Statistics of the datasets

Smell	Alias	# Positive	# Negative
Complex Method	CM	12,489	144,460
Complex Conditional	CC	6,186	149,767
Feature Envy	FE	1,788	51,260
Multifaceted Abstraction	MA	290	50,205

5.3. Hardware specification

All experiments are conducted on Google Colab Pro, which provides the necessary computational resources for running the time series transformation and classifier training processes efficiently.

5.4. Evaluation plan

The performance of CSDXR is compared with four baseline models. The baseline models include three variants of an auto-encoder model for code smell detection, introduced by Sharma et al. [7]:

[1] https://github.com/tushartushar/DeepLearningSmells

1. **AE-Dense:** An auto-encoder model using dense layers for both the encoder and decoder.
2. **AE-CNN:** An auto-encoder model employing Convolutional Neural Networks (CNNs) for the encoder and decoder.
3. **AE-LSTM:** An auto-encoder model utilizing Long Short-Term Memory (LSTM) networks.

The fourth baseline model is **DeepSmells**, introduced by Ho et al. [9].

The evaluation process involves comparing the performance metrics of CSDXR with those reported for the baseline models in the study by Ho et al. [9]. This comparison is carried out across four datasets that include Complex Method (CM), Complex Conditional (CC), Feature Envy (FE), and Multifaceted Abstraction (MF).

Each dataset is shuffled and then is split into training and testing subsets with a 70%/30% ratio. The training set is used to train the models, while the testing set is used to evaluate their performance.

The experiments utilize the Sktime library, a Python framework for time series analysis, to implement the time series transformation process required for model training and evaluation.

5.5. Performance

Given the heavy imbalance in code smells datasets [47], using accuracy alone to evaluate classifier performance can lead to misleading results [48]. Therefore, this study uses Precision, Recall, F-measure (F_1), and Matthews Correlation Coefficient (MCC) to assess the performance of the CSDXR method. These metrics are commonly used in code smell detection studies [7, 9, 10, 22, 47] and provide a more reliable evaluation of classifier performance in imbalanced datasets.

– **Precision** measures the proportion of true positive predictions among all positive predictions made by the classifier. It indicates how many of the detected positives are actually true positives.
– **Recall** measures the proportion of true positives that were correctly identified by the classifier out of all actual positives. It reflects the classifier's ability to identify all relevant instances.
– **F-measure** (F_1-score) is the harmonic mean of *precision* and *recall*. It provides a single metric that balances the trade-off between Precision and Recall, making it useful when there is an uneven class distribution.
– **Matthews Correlation Coefficient** (MCC) is a more robust metric compared to accuracy and F-measure. It provides a balanced measure that takes into account all four categories of the confusion matrix: True Positives (TP), False Negatives (FN), False Positives (FP), and True Negatives (TN). The MCC is particularly useful for evaluating performance on imbalanced datasets. It ranges from -1 to $+1$, where $+1$ indicates a perfect prediction, -1 indicates total disagreement, and 0 indicates no better than random prediction.

The confusion matrix is used to calculate these metrics and is summarized as follows:

– **True Positives** (TP): Instances that are actually positive and correctly classified as positive.
– **False Negatives** (FN): Instances that are actually positive but incorrectly classified as negative.
– **False Positives** (FP): Instances that are actually negative but incorrectly classified as positive.

– **True Negatives** (TN): Instances that are actually negative and correctly classified as negative.
  These metrics are calculated using the following formulas:

$$precision = \frac{TP}{TP + FP} \tag{2}$$

$$recall = \frac{TP}{TP + FN} \tag{3}$$

F_1 (F-measure): F_1 is the harmonic mean of precision and recall

$$F_1 = \frac{2 \cdot recall \cdot precision}{recall + precision} \tag{4}$$

$$MCC = \frac{TP \cdot TN - FP \cdot FN}{\sqrt{(TP + FP) \cdot (TP + FN) \cdot (TN + FP) \cdot (TN \cdot FN)}} \tag{5}$$

Table 2. Confusion matrix

	Positive (predicted)	Negative (predicted)
True (Actual)	*TP*	*FN*
False (Actual)	*FP*	*TN*

6. Results and discussion

This section presents and discusses the experimental results for the CSDXR method in the context of code smell detection.

6.1. Results for RQ1

RQ1: How do different CSDXR configurations affect the prediction performance?

Approach The CSDXR method leverages either MiniRocket or a similar method such as Rocket for pooling a time series model. These methods consist of two main components:

1. **Pooling a Model**: This involves transforming the time series into a feature vector using MiniRocket or Rocket.
2. **Classification**: This involves using a classifier to predict the class of the time series based on the pooled feature vector.

While the literature typically employs MiniRocket and Rocket with linear classifiers, these methods can theoretically be used with any classifier [19]. Therefore, the CSDXR method's performance depends on various hyperparameters related to both the pooling method and the classifier.

Hyperparameter Tuning Identifying the optimal parameters for the CSDXR approach is a challenging task, as it involves searching across a vast space of possible parameter combinations. This problem, commonly referred to in the literature as hyperparameter tuning, has been extensively studied, with proposed solutions ranging from basic methods, such as grid search, to more advanced metaheuristic-based approaches [49, 50]. In the case of CSDMR, the MiniRocket hyperparameters, such as the number of kernels and dilation, were systematically varied during the training process using a simple grid search approach. The process began by setting dilation to the widely used value of 32, as reported in the literature, and varying the number of kernels across the values {84, 168, 252, 1000, 10000, 10120}. Once the optimal number of kernels was identified, the dilation size was varied across {1, 22, 32, 44} to further refine the hyperparameters for best performance.

In the case of CSDR only the number of kernels was varied, as dilation is randomly set within the Rocket algorithm. The best hyperparameters found during training were then applied in the testing phase to ensure consistency and fairness in evaluation. Classifier hyperparameters were set to their default values as provided by the software packages used.

Design Alternatives To evaluate how different configurations affect the performance of CSDXR, the following design alternatives were studied:

1. **CSDXR with Logistic Regression** (CCDMR_LR) [51]: A linear classifier that models the relationship between features and the target class.
2. **CSDXR with XGBoost** (CCDMR_XGB) [52]: An ensemble method that combines multiple decision trees to improve predictive performance.
3. **CSDXR with Random Forest** (CCDMR_RF) [53]: An ensemble method that aggregates multiple decision trees to enhance robustness and accuracy.
4. **CSDXR with Naïve Bayes** (CCDMR_NB) [54]: A probabilistic classifier based on Bayes' theorem, assuming feature independence. It is important to note that time series data inherently contains correlations between consecutive data points, which violate the assumption of feature independence in models like Naïve Bayes. Despite this, the Naïve Bayes model was selected for its simplicity and efficiency.
5. **CSDXR with Decision Tree** (CCDMR_DT) [55]: A model that splits data based on feature values to make predictions.

The goal was to assess how each classifier, in combination with MiniRocket or Rocket, impacts the overall effectiveness of CSDXR in detecting code smells. The experiments aimed to determine the optimal configuration and hyperparameters for achieving the best performance. The performance of the CSDR design alternatives is detailed in Section 6.3.

Results

1. Effect of the number of kernels on the effectiveness of different design alternatives of the CSDXR model

   Regarding the effect of the number of kernels, Figure 4 shows the F_1-scores of different CSDXR design alternatives as the number of kernels vary, while Figure 5 illustrates the corresponding Matthews Correlation Coefficient (MCC) scores.

   The analysis of these figures reveals that the CSDXR variants achieved their highest performance with 10120 kernels. Specifically, CSDMR_XGB exhibited the highest F_1-score and MCC measures across the CC, CM, and MF datasets. This variant outperformed others with a significant margin, demonstrating its strong capability in detecting code smells effectively.

   In comparison, CSDMR_DT also showed robust performance, particularly on the CC, FE, and MF datasets, although it did not surpass CSDMR_XGB in overall metrics.

   The summary of average performance metrics across the different design alternatives is provided in Table 3. According to this table, CSDMR_XGB leads with an F_1-score of 0.41, followed by CSDMR_DT with 0.35. CSDMR_RF, CSDMR_NB, and CSDMR_-

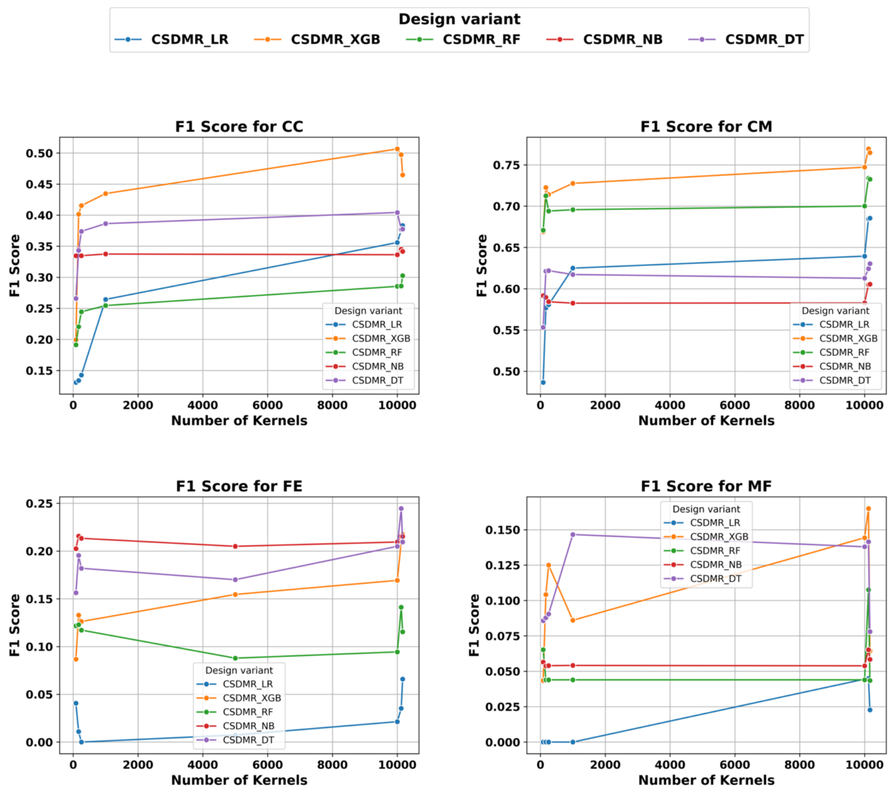


Figure 4. The F_1-score for each CSDMR variant across different smell types

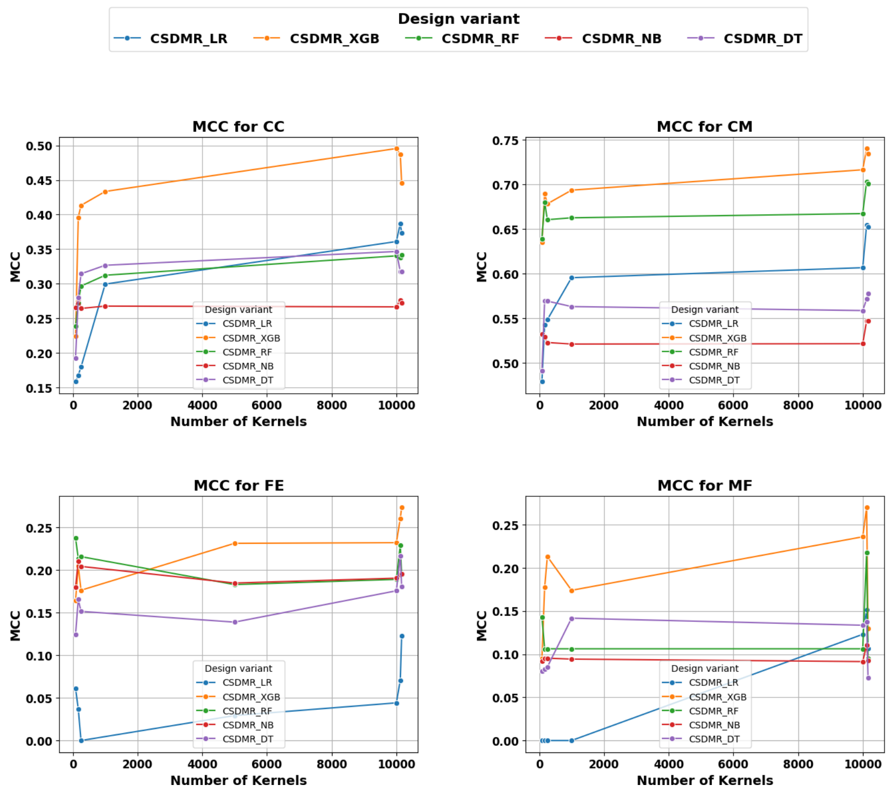


Figure 5. *MCC* score for each CSDMR variant across different smell types

Table 3. Average F_1 and MCC scores for each CSDMR variant across different smell types

Design variant	Smell	F_1-score	MCC
CSDMR_LR	CC	0.38	0.39
	CM	0.68	0.65
	FE	0.04	0.07
	MF	0.04	0.15
average		0.29	0.32
CSDMR_XGB	CC	0.50	0.49
	CM	0.77	0.74
	FE	0.21	0.26
	MF	0.16	0.27
Average		0.41	0.44
CSDMR_RF	CC	0.29	0.34
	CM	0.73	0.70
	FE	0.14	0.23
	MF	0.11	0.22
Average		0.32	0.37
CSDMR_NB	CC	0.35	0.28
	CM	0.60	0.55
	FE	0.22	0.19
	MF	0.07	0.11
Average		0.31	0.28
CSDMR_DT	CC	0.38	0.32
	CM	0.62	0.57
	FE	0.24	0.22
	MF	0.14	0.14
average		0.35	0.31

LR showed lower scores of 0.32, 0.31, and 0.29, respectively. For MCC, CSDMR_XGB achieved a score of 0.44, with CSDMR_RF next at 0.37. CSDMR_LR scored 0.32, CSDMR_DT 0.31, and CSDMR_NB 0.28.

The consistent trend across both F_1-score and MCC metrics indicates that the number of kernels plays a crucial role in the performance of CSDXR. The optimal kernel number of 10120 maximizes the feature representation capability of MiniRocket, thus enhancing the effectiveness of the classifiers. CSDMR_XGB's superior performance highlights its potential for robust code smell detection, while CSDMR_DT also proves to be a strong contender, particularly in certain datasets.

Overall, the results suggest that the choice of kernel number and the specific design variant significantly impact the performance of the CSDXR method. The figures and table provide a clear visualization of these effects, supporting the effectiveness of the CSDMR_XGB design in particular.

2. The effect of dilation

In this analysis, we examine how different dilation sizes impact the performance of the CSDXR model. The experiment explored four scenarios with dilation sizes set to 1, 22, 32, and 44, while maintaining the number of kernels at 10120, which was previously identified as optimal.

Figure 6 displays the performance of each CSDMR variant across the four datasets, demonstrating sensitivity to the dilation size. Figure 7 further illustrates the specific F_1 and MCC scores obtained for various dilation sizes.

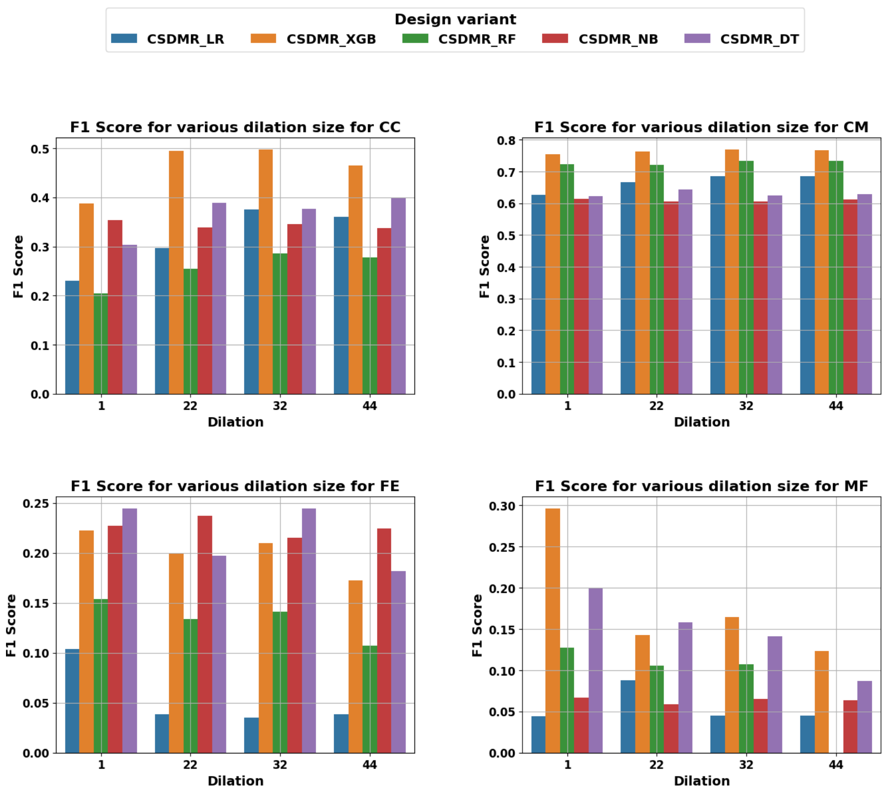


Figure 6. F_1-score by dilation size

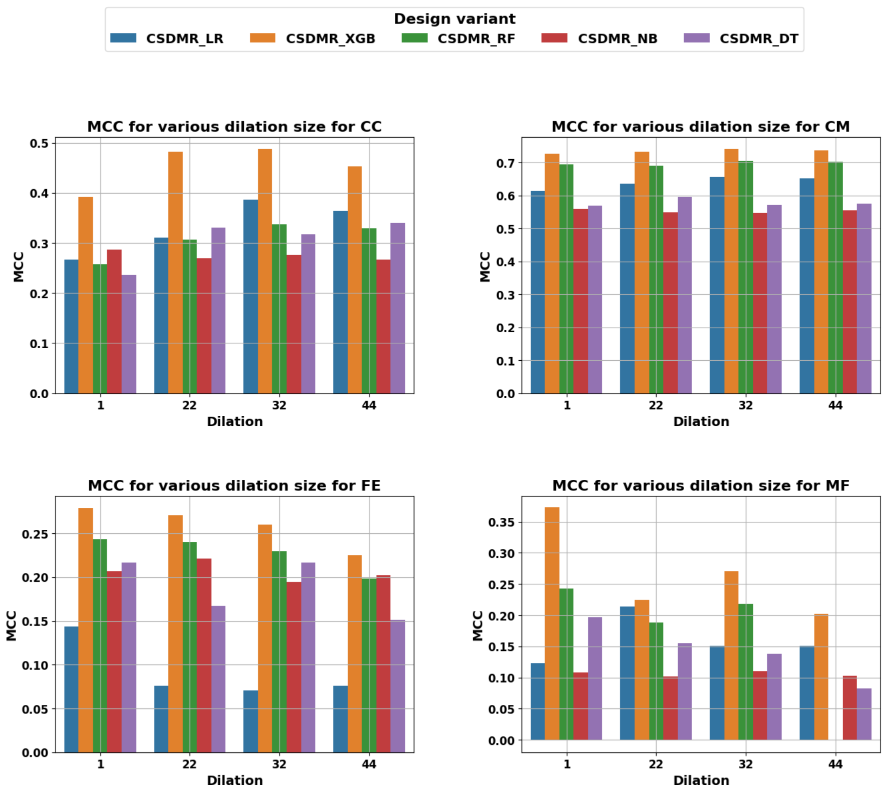


Figure 7. *MCC* score by dilation size

Table 4. Best performance of each CSDMR variant across different smell types by dilation size

Smell	Design variant	Dilation	Precision	Recall	F_1-score	*MCC*
CC	CSDMR_LR	32	0.67	0.26	0.38	0.39
	CSDMR_XGB	32	0.70	0.39	**0.50**	**0.49**
	CSDMR_RF	32	**0.74**	0.18	0.29	0.34
	CSDMR_NB	32	0.30	0.41	0.35	0.28
	CSDMR_DT	44	0.38	**0.42**	0.40	0.34
CM	CSDMR_LR	32	0.32	0.02	0.04	0.07
	CSDMR_XGB	32	**0.81**	**0.73**	**0.77**	**0.74**
	CSDMR_RF	32	0.80	0.68	0.73	0.70
	CSDMR_NB	1	0.59	0.65	0.61	0.56
	CSDMR_DT	22	0.64	0.65	0.64	0.59
FE	CSDMR_LR	1	0.40	0.06	0.10	0.14
	CSDMR_XGB	1	0.61	0.14	0.22	**0.28**
	CSDMR_RF	1	**0.73**	0.09	0.15	0.24
	CSDMR_NB	22	0.17	0.40	**0.24**	0.22
	CSDMR_DT	32	0.23	**0.26**	**0.24**	0.22
MF	CSDMR_LR	22	**1.00**	0.05	0.09	0.21
	CSDMR_XGB	1	0.76	0.18	**0.30**	**0.37**
	CSDMR_RF	1	0.86	0.07	0.13	0.24
	CSDMR_NB	1	0.04	**0.43**	0.07	0.11
	CSDMR_DT	44	0.08	0.10	0.20	0.20

The results reveal that CSDMR_XGB consistently achieved the highest F_1 and *MCC* scores on the CC and CM datasets across all dilation sizes. For the MF dataset, CSDMR_XGB attained the best F_1-scores for dilation sizes of 1, 32, and 44. On the FE dataset, CSDMR_XGB excelled in *MCC* score with a dilation size of 1. However, for the FE dataset's F_1-score, CSDMR_DT and CSDMR_NB were the top performers for dilation sizes of 1 and 32, respectively.

To better understand the optimal dilation size for each variant, Table 4 presents the dilation sizes that achieved the highest F_1 and *MCC* scores for each CSDMR variant. The table indicates that CSDMR_XGB outperforms other variants in terms of F_1-score on most datasets, except for FE, where its F_1-score of 0.22 is lower compared to the 0.24 achieved by CSDMR_DT and CSDMR_NB. Nonetheless, the *MCC* scores for CSDMR_XGB were superior across all datasets, with values of 0.49 for CC, 0.74 for CM, 0.28 for FE, and 0.37 for MF, reflecting a more comprehensive and reliable measure of performance.

The best dilation sizes for each smell type were found to be 32 for CC, 32 for CM, 1 for FE, and 1 for MF. The F_1-scores ranged from 0.22 to 0.77, and the *MCC* values ranged from 0.28 to 0.74.

Overall, the dilation size significantly affects the performance of the CSDXR model, with the optimal size varying depending on the dataset and the specific variant used. The findings suggest that fine-tuning dilation sizes is crucial for achieving the best performance in code smell detection.

Moreover, this table shows that, firstly, for CSDMR_DT, the best performance was observed with dilation sizes of 44, 22, 32, and 44 for CC, CM, FE, and MF smells, respectively. Consequently, the F_1-scores for these settings ranged from 0.20 to 0.64, and the *MCC* values ranged from 0.20 to 0.59.

In contrast, **CSDMR_RF** achieved its best performance with dilation sizes of 32, 32, 1, and 1 for CC, CM, FE, and MF smells. Therefore, the F_1-scores ranged from 0.13 to 0.75, and the *MCC* values varied between 0.24 and 0.70.

Furthermore, **CSDMR_NB** showed optimal performance with dilation sizes of 32, 1, 22, and 1 for CC, CM, FE, and MF smells. In this case, the F_1-scores ranged from 0.07 to 0.61, with *MCC* values between 0.11 and 0.56.

On the other hand, **CSDMR_LR** performed best with dilation sizes of 32, 32, 1, and 22 for CC, CM, FE, and MF smells. The F_1-scores for these sizes ranged from 0.04 to 0.38, and the *MCC* values ranged from 0.07 to 0.39.

Regarding precision, **CSDMR_RF** achieved the highest scores for CC, **CSDMR_-XGB** for CM, **CSDMR_RF** for FE, and **CSDMR_LR** for MF, with precision values of 0.74, 0.81, 0.73, and 1.00, respectively.

In terms of recall, **CSDMR_DT** led for CC, **CSDMR_XGB** for CM, **CSDMR_-DT** for FE, and **CSDMR_NB** for MF, with recall values of 0.42, 0.73, 0.26, and 0.43, respectively.

In summary, the results underscore that dilation size significantly impacts model performance. Different variants exhibit varying sensitivities to dilation size, thus highlighting the need for careful tuning to optimize performance for specific code smells and variant configurations.

RQ1. Hypothesis 1: It is feasible to detect code smells using classifiers trained on representations pooled by well-configured MiniRocket or its variants from time series representing source codes.

It is evident from the study that the CSDMR variants can achieve F_1 and *MCC* scores of 0.74 or higher on certain datasets, demonstrating the feasibility of detecting code smells using classifiers trained on well-configured MiniRocket representations of time series from source code. However, the performance of these classifiers is not uniform across all datasets; for some datasets, the F_1-score may be as low as 0.3. This variability highlights the sensitivity of performance to the specific type of code smell being detected, the characteristics of the dataset, and the configuration of hyperparameters. These findings emphasize the importance of careful dataset selection and hyperparameter tuning in achieving optimal results.

6.2. Results of RQ2

RQ2: How efficient is the CSDXR method?

Approach The performance of CSDMR (Code Smell Detection using MiniRocket) is compared with four baseline models. The baseline models include three variants of an auto-encoder model for code smell detection, introduced by Sharma et al. [7]:

– **AE-Dense**: An auto-encoder model using dense layers for both the encoder and decoder.
– **AE-CNN**: An auto-encoder model employing Convolutional Neural Networks (CNNs) for the encoder and decoder.
– **AE-LSTM**: An auto-encoder model utilizing Long Short-Term Memory (LSTM) networks.

The fourth baseline model is **DeepSmells**, introduced by Ho et al. [9].

Results Regarding the performance of the baseline models, Table 5 presents the results for each type of smell. The data indicate that the CSDMR_XGB model surpasses the

Table 5. Performance of baseline models across different smell types

Smell	Model	Metric			
		P	R	F_1	*MCC*
CM	AE-Dense	0.483	0.630	0.547	0.508
	AE-CNN	0.472	0.582	0.521	0.478
	AE-LSTM	0.468	0.615	0.532	0.491
	DeepSmells	0.731	**0.779**	0.754	0.734
	CSDMR_LR	0.323	0.019	0.035	0.071
	CSDMR_XGB	**0.811**	0.731	**0.769**	**0.740**
	CSDMR_RF	0.802	0.676	0.734	0.703
	CSDMR_NB	0.585	0.648	0.615	0.559
	CSDMR_DT	0.643	0.645	0.644	0.594
CC	AE-Dense	0.170	0.387	0.237	0.211
	AE-CNN	0.194	0.276	0.228	0.193
	AE-LSTM	0.180	0.329	0.232	0.201
	DeepSmells	0.575	**0.604**	**0.589**	**0.568**
	CSDMR_LR	0.667	0.262	0.376	0.387
	CSDMR_XGB	0.697	0.387	0.497	0.487
	CSDMR_RF	**0.737**	0.177	0.286	0.337
	CSDMR_NB	0.298	0.410	0.345	0.277
	CSDMR_DT	0.382	0.420	0.400	0.340
FE	AE-Dense	0.170	0.387	0.237	0.211
	AE-CNN	0.157	**0.493**	0.238	0.235
	AE-LSTM	0.197	0.254	0.222	0.197
	DeepSmells	0.341	0.258	**0.294**	0.269
	CSDMR_LR	0.395	0.060	0.104	0.143
	CSDMR_XGB	0.613	0.136	0.223	**0.279**
	CSDMR_RF	**0.730**	0.086	0.154	0.243
	CSDMR_NB	0.168	0.405	0.237	0.221
	CSDMR_DT	0.230	0.261	0.245	0.217
MA	AE-Dense	0.031	**0.747**	0.060	0.135
	AE-CNN	0.031	0.678	0.060	0.127
	AE-LSTM	0.033	0.402	0.061	0.099
	DeepSmells	0.287	0.272	0.279	0.275
	CSDMR_LR	**1.000**	0.046	0.088	0.214
	CSDMR_XGB	0.762	0.184	**0.296**	**0.373**
	CSDMR_RF	0.857	0.069	0.128	0.242
	CSDMR_NB	0.036	0.425	0.067	0.109
	CSDMR_DT	0.076	0.103	0.199	0.197

other models in terms of both F_1 and *MCC* evaluation metrics for the CM and MF smells, achieving F_1-scores of 0.769 and 0.296, and *MCC* scores of 0.740 and 0.373, respectively. Specifically, the enhancement rates in terms of F_1-score were 1.99% and 6.09% for Complex Method and Multifaceted Abstraction smells, respectively. In terms of *MCC*, the improvement rates were 0.82% and 35.64% for these two smells, respectively. The CSDMR_XGB model outperformed all baseline models in term of *MCC* on Feature Envy dataset. The obtained score was 0.279.

Additionally, the CSDMR_XGB model excels in Precision for the CM and CC smells, with Precision values of 0.811 and 0.697, respectively. The CSDMR_LR model achieves a Precision score of 1.00 for the MF smell and also exhibits superior Precision on the CC smell with a value of 0.737 compared to all other models.

In terms of Recall, the AE-CNN model stands out, achieving the highest Recall scores for the FE and MF smells, with values of 0.493 and 0.747, respectively.

Table 5 also shows that DeepSmells outperforms all other models in terms of F_1 and *MCC* scores for the CC and FE smells. Although DeepSmells excels on these datasets, the CSDMR_XGBoost model surpasses AE-Dense, AE-CNN, and AE-LSTM on the CC dataset, achieving the best performance compared to AE-LSTM across all datasets. For the FE smell, CSDMR_DT outperforms AE-Dense, AE-CNN, and AE-LSTM. The performance of CSDMR_NB is comparable to that of AE-Dense.

Table 6. Mean F_1-scores and *MCC* values of CSDMR and baseline models

Model	Mean F_1	Mean *MCC*
CSDMR_LR	0.15	0.20
CSDMR_XGB	0.45	0.47
CSDMR_RF	0.33	0.38
CSDMR_NB	0.32	0.29
CSDMR_DT	0.37	0.34
AE-Dense	0.27	0.27
AE-CNN	0.26	0.26
AE-LSTM	0.26	0.25
DeepSmells	0.48	0.46

In terms of mean F_1 and mean *MCC* scores, the results in Table 6 demonstrate that the CSDMR_XGB model outperforms AE-Dense, AE-CNN, and AE-LSTM and achieves performance comparable to DeepSmells. In terms of mean F_1 and mean *MCC* scores, the results in Table 6 demonstrate that the CSDMR_XGB model outperforms AE-Dense, AE-CNN, and AE-LSTM and achieves performance comparable to DeepSmells. To validate this hypothesis and after confirming the normality of the data, we conducted a Student's *t*-test (t-test) [56] to detect whether performance differences between CSDMR variants and baseline models are statistically significant. The test used significance rate α equals to 0.05. In hypothesis testing, α denotes the probability of making a Type I error (falsely rejecting the null hypothesis, H_0). An α set to 0.05 means there is only a 5% probability of concluding that an effect exists when it does not. A Result is considered statistically significant when the obtained *p*-value from the Student's *t*-test is less than the alpha ($p < \alpha$).However, statistical significance alone is insufficient because *p*-values do not show the magnitude of the observed effect. Therefore, In addition to significance, we also assessed practical significance by reporting and interpreting effect sizes, which quantify performance differences between models. The magnitude of the difference is quantified using Hedges' g [57] with a 95% confidence interval (CI) and in terms of both the obtained F_1 and *MCC* scores. Hedges' g was chosen over Cohen's d due to our small sample size. The performance difference is quantified in terms of both the obtained F_1 and *MCC* scores. The effect sizes were interpreted using Cohen's d guidelines [58]:
- Negligible effect: < 0.2
- Small effect $= 0.2$
- Medium effect $= 0.5$
- Large effect $= 0.8$

Table 7 Shows *p*-values for F_1 and *MCC* scores when comparing CSDMR variants with baseline models, along with effect sizes for significant results (*p*-value < 0.05)and power values for non-significant results. Meanwhile, Table 6 shows Mean F_1-scores and *MCC* values of CSDMR and baseline models. These tables show that, in term of *MCC*:
1. CSDMR_XGB significantly outperformed AE_Dense, AE_CNN, and AE_LSTM with a large effect size.

Table 7. Comparison of models with statistical tests: p-value of the statistical test along with effect size (Hedges' g) in case of a significant test, and power values in case of non-significant test. S? indicates whether the result is significant (Yes) or not (No). Negative effect size indicates a performance superiority for the baseline model

		MCC				F_1			
Model 1	Model 2	p-value	Effect size	Power	S?	p-value	Effect size	Power	S?
CSDMR_LR	AE-Dense	0.67		0.08	No	0.46		0.13	No
CSDMR_LR	AE-CNN	0.71		0.07	No	0.48		0.12	No
CSDMR_LR	AE-LSTM	0.77		0.06	No	0.48		0.12	No
CSDMR_LR	DeepSmells	0.16		0.37	No	0.09		0.51	No
CSDMR_XGB	AE-Dense	0.02	0.97		Yes	0.07		0.16	No
CSDMR_XGB	AE-CNN	0.03	1.04		Yes	0.07		0.17	No
CSDMR_XGB	AE-LSTM	0.02	1.05		Yes	0.06		0.17	No
CSDMR_XGB	DeepSmells	0.84		0.05	No	0.33		0.05	No
CSDMR_RF	AE-Dense	0.04	0.52		Yes	0.39		0.06	No
CSDMR_RF	AE-CNN	0.07		0.12	No	0.37		0.06	No
CSDMR_RF	AE-LSTM	0.03	0.60		Yes	0.33		0.06	No
CSDMR_RF	DeepSmells	0.21		0.07	No	0.08		0.11	No
CSDMR_NB	AE-Dense	0.31		0.05	No	0.17		0.06	No
CSDMR_NB	AE-CNN	0.33		0.06	No	0.17		0.06	No
CSDMR_NB	AE-LSTM	0.07		0.06	No	0.13		0.06	No
CSDMR_NB	DeepSmells	0.04	−0.70		Yes	0.03	−0.61		Yes
CSDMR_DT	AE-Dense	0.07		0.08	No	0.06		0.09	No
CSDMR_DT	AE-CNN	0.12		0.09	No	0.05		0.10	No
CSDMR_DT	AE-LSTM	0.04	0.44		Yes	0.04	0.48		Yes
CSDMR_DT	DeepSmells	0.05		0.11	No	0.04	-0.43		Yes

2. CSDMR_RF significantly outperformed AE_Dense and AE_LSTM with a medium effect size.
3. DeepSmells significantly outperformed CSDMR_NB with a medium effect size.
4. CSDMR_DT significantly outperformed AE_LSTM with a small effect size.

These tables show also that in term of F_1:

1. DeepSmells significantly outperformed CSDMR_NB with a medium effect size.
2. AE_LSTM significantly outperformed CSDMR_NB with a small effect size.
3. DeepSmells significantly outperformed CSDMR_DT with a small effect size.

Additionally, a post-hoc power analysis (i.e., retrospective power analysis) is conducted for cases where statistically nonsignificant results were obtained to assess the reliability of these findings. It is important to mention that while researchers agree on the importance of prospective power analysis to determine an adequate sample size for a planned research study [59, 60], they disagree about the value of a post hoc power analysis. Some researchers still recommend that power analysis can be done retrospectively, especially when a statistically nonsignificant result is obtained [59, 60]. In this case, a post hoc power analysis is conducted to determine if the lack of significance is due to low statistical power or to a truly small effect. Power analysis is based on four related parameters (the sample size, the effect size, the significance level (α, often set to 0.05), and the statistical power. A power analysis is generally conducted to estimate one of these four parameters given the remaining three values. The statistical power of a test is the probability of rejecting H_0 when it is really false (i.e., the capacity to detect an effect if it is really there). This power is tied by an inverse relation with β (i.e., the probability of making a Type II error) and is equal to $1 - \beta$. A low power value indicates that there is a high risk of Type II errors, while a high value indicates a low risk of Type II errors. The literature shows that 0.20 is the acceptable level of β, so the desired power is 0.80. In our study, the estimated post hoc power was found to be low and range between 0.5 and 0.05 in each nonsignificant case, which leads to the conclusion that the non-significance is due to low power and suggests that more powerful research should be conducted.

RQ2. Hypothesis 2: The CSDXR method can improve the performance of code smell detection.

Overall, the results indicate that the CSDMR_XGB model achieves superior F_1 and MCC scores compared to the baseline models on two types of smells. Specifically, the enhancement rates in terms of F_1-score were 1.99% and 6.09% for Complex Method and Multifaceted Abstraction smells, respectively. In terms of MCC, the improvement rates were 0.82% and 35.64% for these two smells, respectively. Additionally, it demonstrates enhanced Precision for one specific type of smell. The CSDMR_LR model also excels in Precision for two types of smells. For each type of smell, at least one CSDMR variant surpasses the performance of the AE-Dense, AE-CNN, and AE-LSTM models.
In general, The Student's test indicate that CSDMR outperformed AE-Dense, AE-CNN, and AE-LSTM models while achieving performance comparable to DeepSmells.
Moreover, the use of a simple grid search strategy for hyperparameter tuning provides an initial baseline for the performance of the CSDXR approach. Therefore, we accept the hypothesis that CSDXR models can improve the performance of code smell detection. However, this strategy may not fully leverage the model's potential. Incorporating more advanced hyperparameter tuning methods, such as evolutionary algorithms, could lead to improved performance. Future work will be dedicated to the exploration of these advanced strategies to reveal the potential of CSDXR.
Finally, it is also important to note that the statistical power of the analysis suggests that future studies need to be carried out to validate the reliability of nonsignificant.

6.3. Results of RQ3

RQ3: How does the CSDMR's performance and computational cost compare to that of CSDR?

Approach The CSDXR method, which consists of transformation and classification components, is compared with the CSDR method. For this comparison, we implemented the transformation component using the Rocket technique (CSDR) and assessed the performance of CSDR variants against CSDMR variants in terms of F_1-score and MCC. Additionally, we compared the transformation times logged for both CSDMR and CSDR methods to convert the training dataset.
Results Table 10 presents the performance metrics for each design variant of CSDR and CSDMR, with configurations that produce feature vectors of size S for each type of smell(O.size). The table includes performance results in terms of F_1-score, MCC, and transformation time (Trans. Times) in seconds. It also highlights the differences in F_1 and MCC scores between each CSDR variant and its corresponding CSDMR variant. The final column shows the Transformation Time Ratio (TTR) CSDMR relative to CSDR for each smell.

The results indicate that the performance of CSDMR and CSDR variants is comparable, with differences in F_1 and MCC scores ranging from 0 to 0.16. In all cases, there were no significant performance differences between CSDR and CSDMR. There were no performance differences between CSDR and CSDMR in seven cases. CSDMR performed worse than CSDR in 17 cases, while it outperformed CSDR in 16 cases. Regarding transformation times, CSDMR is significantly faster than CSDR. Specifically, CSDMR demonstrated a speed rate approximately 16 times faster on the CC dataset with an output vector size of 84. The speed rate varied between 2 and 16 across different datasets, with an average speed rate of 12.7.

A statistical test was conducted to examine whether there are significant differences between the CSDMR and CSDR results, specifically in terms of F_1-scores, MCC scores, and transformation times.

The Kolmogorov-Smirnov test [61] was chosen for this analysis due to the non-normal distribution of the data. A p-value less than 0.05 indicates the presence of significant differences between the CSDMR and CSDR approaches. Table 8 presents the p-values for CSDMR and CSDR in terms of F_1 and MCC scores, while Table 10 presents the mean execution times for CSDMR and CSDR, along with the p-values for transformation times.

Table 9 shows that all the obtained p-values for F_1 and MCC scores are greater than 0.05, indicating that there are no significant differences between CSDMR and CSDR in these metrics. However, the p-value for transformation time is less than 0.05 (see Table 10), suggesting a significant difference between CSDMR and CSDR in terms of transformation time. Based on the mean execution times of CSDMR and CSDR and the p-value presented in this table, we can conclude that CSDMR is faster than CSDR in terms of transformation time.

This analysis shows that while the performance of CSDMR and CSDR is generally similar, CSDMR offers a considerable advantage in terms of transformation time, making it a more efficient choice for code smell detection.

RQ3. Hypothesis 3: CSDMR variants' performance is comparable to CSDR variants in the context of code smell detection, while CSDMR variants are faster than CSDR variants.

Therefore, we conclude that the performance of CSDMR variants is comparable to that of CSDR variants. Additionally, CSDMR variants are significantly faster than CSDR variants.

6.4. Discussion

This study demonstrated that the proposed method, CSDXR, have the potential to detect smells without the use an extensive feature engineering process. The study revealed also that MiniRocket combined with the XGBoost classifier outperforms other variants in terms of detection performance.

However, the obtained results demonstrate that the performance of the CSDXR method is highly sensitive to the type of code smell. While slight improvements were observed for two code smells (CM and MA smells), the results indicate that there is still room for significant enhancement, as the highest F_1 and MCC scores achieved were 0.769 and 0.740 respectively.

Table 8. Performance metrics for each design variant of CSDR and CSDMR

Smell	O. size	Model variant	CSDR F_1	CSDR *MCC*	CSDR Trans. Time(s)	CSDMR F_1	CSDMR *MCC*	CSDMR Trans. Time (s)	F_1 diff.	*MCC* diff.	TTR
CC	82	CSDXR_LR	0.00	0.00	27.82	**0.13**	**0.16**	1.74	-0.13	-0.16	**16.01**
	82	CSDXR_XGB	**0.30**	**0.29**		0.20	0.22		0.10	0.07	
	82	CSDXR_RF	**0.21**	**0.27**		0.19	0.24		0.02	0.03	
	82	CSDXR_NB	**0.34**	**0.28**		0.33	0.27		0.01	0.02	
	82	CSDXR_DT	**0.31**	**0.25**		0.27	0.19		0.05	0.05	
CM	82	CSDXR_LR	**0.60**	**0.55**	27.78	0.49	0.48	2.31	0.11	0.07	12.03
	82	CSDXR_XGB	**0.69**	**0.65**		0.67	0.63		0.02	0.01	
	82	CSDXR_RF	**0.69**	**0.65**		0.67	0.64		0.02	0.01	
	82	CSDXR_NB	**0.60**	**0.54**		0.59	0.53		0.01	0.01	
	82	CSDXR_DT	**0.57**	**0.51**		0.55	0.49		0.02	0.02	
FE	82	CSDXR_LR	0.00	0.00	198.58	**0.04**	**0.06**	14.71	-0.04	-0.06	13.50
	82	CSDXR_XGB	0.08	0.14		**0.09**	**0.16**		0.00	-0.02	
	82	CSDXR_RF	0.07	0.16		**0.12**	**0.24**		-0.05	-0.07	
	82	CSDXR_NB	**0.21**	**0.22**		0.20	0.18		0.01	0.04	
	82	CSDXR_DT	0.14	0.11		**0.16**	**0.12**		-0.01	-0.01	
MF	82	CSDXR_LR	0.00	0.00	218.23	0.00	0.00	16.27	0.00	0.00	13.42
	82	CSDXR_XGB	0.02	0.06		**0.04**	**0.09**		-0.02	-0.03	
	82	CSDXR_RF	0.02	0.11		**0.07**	**0.14**		-0.04	-0.04	
	82	CSDXR_NB	0.05	**0.14**		**0.06**	0.09		0.00	0.05	
	82	CSDXR_DT	0.15	0.14		0.09	0.08		0.06	0.06	
CC	1000	CSDXR_LR	0.10	0.13	261.60	**0.26**	**0.30**	17.07	-0.16	-0.17	15.32
	1000	CSDXR_XGB	0.42	0.40		**0.43**	**0.43**		-0.02	-0.03	
	1000	CSDXR_RF	**0.29**	**0.34**		0.25	0.31		0.04	0.02	
	1000	CSDXR_NB	**0.34**	**0.27**		**0.34**	**0.27**		0.00	0.00	
	1000	CSDXR_DT	0.36	0.30		0.39	0.33		-0.03	-0.03	
CM	1000	CSDXR_LR	**0.64**	**0.60**	286.03	0.62	**0.60**	20.68	0.01	0.00	13.83
	1000	CSDXR_XGB	0.72	**0.69**		**0.73**	**0.69**		0.00	0.00	
	1000	CSDXR_RF	**0.72**	**0.68**		0.70	0.66		0.02	0.02	
	1000	CSDXR_NB	**0.59**	**0.53**		0.58	0.52		0.00	0.01	
	1000	CSDXR_DT	0.60	0.55		0.62	0.56		-0.01	-0.02	
FE	1000	CSDXR_LR	**0.09**	**0.20**	1937.35	0.02	0.04	790.66	0.07	0.15	2.45
	1000	CSDXR_XGB	0.16	0.22		**0.17**	**0.23**		-0.01	-0.01	
	1000	CSDXR_RF	**0.13**	**0.23**		0.09	0.19		0.03	0.04	
	1000	CSDXR_NB	0.19	**0.21**		**0.21**	0.19		-0.02	0.02	
	1000	CSDXR_DT	0.17	0.14		0.20	0.18		-0.03	-0.03	
MF	1000	CSDXR_LR	**0.15**	**0.19**	2103.54	0.00	0.00	169.46	0.15	0.19	12.41
	1000	CSDXR_XGB	0.04	0.09		**0.09**	**0.17**		-0.04	-0.08	
	1000	CSDXR_RF	0.02	0.06		**0.04**	**0.11**		-0.02	-0.05	
	1000	CSDXR_NB	0.05	0.13		0.05	0.09		0.00	0.04	
	1000	CSDXR_DT	0.08	0.07		0.15	0.14		-0.07	-0.07	

Table 9. p-values for F_1 and *MCC* scores comparing CSDMR and CSDR

		CSDR			
	Smell	CC	CM	FE	MF
CSDMR	F_1	0.87	0.87	0.87	0.87
	MCC	0.5	0.87	0.99	0.99

Table 10. Mean execution time and p-values for transformation times comparing CSDMR and CSDR

Approach	Mean execution time (s)
CSDMR	129.11
CSDR	632.62
p-value	0.019

While, the CSDXR low performance obtained can be attributed to the imbalanced nature of the dataset used, which makes the detection process more challenging. In general, better performance could likely be achieved with more balanced datasets.

Additionally, three other primary reasons can explain these performance differences:

1. **Nature of the Code Smell**: The inherent characteristics of different code smells play a significant role. Some code smells exhibit identifiable patterns in their structure, while others do not, making them harder to detect.
2. **Nature of the CSDXR Approach**: The performance is influenced by the ability of the kernels and/or pooling operators used in the Rocket and the MiniRocket method to detect and summarize patterns present in the source code. This highlights the importance of kernel design in identifying meaningful patterns.
3. **Source Code Transformation**: The transformation approach used to convert source code into time series data may fail to adequately reveal the patterns present in the source code. This can impact the ability of the method to effectively detect certain code smells.

For instance, the superior results on the CM and MA datasets may be attributed to the presence of well-defined patterns in the smelly source code for these datasets. In contrast, the other datasets may lack such patterns, resulting in lower performance, or the patterns that identify the smell may be present, but the approach lacks the capability to detect them. Thus, our conjecture is that the variation in CSDXR performance across different types of smells can be explained by the fact that the CSDXR approach is fundamentally a token-based method. Code smells can be detected by leveraging various types of information present in a code fragment—such as lexical, structural, semantic, or contextual information. While some smells can be identified using just one of these types, others require a combination, making the detection process more complex. An effective detection method should ideally incorporate all of them. In our case, the CSDXR method relies primarily on lexical features and only a limited amount of structural information (i.e., tokens and their order of appearance), which may limit its effectiveness for certain smells. For example, detecting the Feature Envy smell also requires semantic information—like the meaning of identifiers, data and control dependencies, comments, and so on.

To clearly identify the reasons behind these results, it is crucial to design new experiments. These should investigate the potential of novel kernel designs, improved pooling operators, and alternative source code transformation approaches to better capture the underlying patterns in source code.

Additionally, while this study primarily focused on evaluating efficiency in terms of transformation time, we recognize that cost associated with memory usage is another critical parameter that significantly influences scalability and practical applicability. Addressing cost related to memory usage will be essential for enhancing the approach's performance in real-world scenarios.

7. Threats to validity

7.1. Internal validity

The internal validity of the study may be affected by several factors. First, the use of the Sktime library for implementing the time series transformation module and setting classifier hyperparameters to package default values may introduce biases or limit the method's performance. Second, while the number of kernels and dilation parameters were varied until satisfactory results were achieved, this approach may not fully capture the robustness of the findings. Additionally, the absence of cross-validation could impact the reliability of the reported results. To address these limitations, future work should include a more comprehensive experimental design involving extensive hyperparameter tuning and the use of cross-validation to ensure a clearer understanding of the method's capabilities and more reliable conclusions.

7.2. External validity

External validity concerns the generalizability of the study's findings. This study evaluated the CSDXR approach on only four types of code smells, each with distinct patterns. However, code smells vary in their properties, and the CSDXR approach shows promise in detecting smells that exhibit clear patterns. It may not, however, be as effective for detecting smells with subtler or less distinct patterns. To enhance the generalizability of the results, it would be beneficial to test the CSDXR approach on a broader range of code smells, including those with less obvious patterns. This would help determine whether the approach can be applied effectively in different contexts and scenarios, ultimately assessing its broader applicability in code smell detection.

8. Conclusion

This paper introduces a novel approach for code smell detection using advanced time series classification techniques such as Rocket and MiniRocket. The proposed CSDXR method involves converting a code source snippet into a sequence of numbers through tokenization, generating a vectorial representation using a random convolutional transform method, and then training a classifier on these vector representations, labeled as smelly or non-smelly. This classifier is subsequently used to determine if new code snippets are smelly based on their representations.

The empirical study, conducted on a well-known dataset to detect four code smells, shows that the CSDXR approach outperforms four state-of-the-art methods, particularly in detecting Complex Method and Multi-Faceted Smells. Although the DeepSmells method performs better than CSDXR, the CSDXR approach surpasses the performance of AE-Dense, AE-CNN, and AE-LSTM models.

Future work will focus on exploring advanced time series representations to further improve code smell detection capabilities.

Acknowledgment

The authors acknowledge the use of language models, including ChatGPT and DeepSeek, which assisted in enhancing the grammatical accuracy and overall clarity of the manuscript during the writing process. All content was thoroughly reviewed, verified, and edited by the authors to ensure it accurately reflects the scientific contributions of this work.

CRediT authorship contribution statement

Dr. Mostefai Abdelkader – conceptualization, methodology, software, data curation, investigation, visualization, writing – original draft, writing – writing – review & editing.
Dr. Mekour Mansour – conceptualization, writing – review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

The code smell datasets used in this research is provided on the link: https://zenodo.org/records/15602620.

Funding

No funding was received to assist with the preparation of this manuscript.

References

[1] P. Avgeriou, P. Kruchten, I. Ozkaya, and C. Seaman, "Managing technical debt in software engineering (Dagstuhl seminar 16162)," *Dagstuhl Reports*, Vol. 6, No. 4, 2016.

[2] P. Kruchten, R.L. Nord, and I. Ozkaya, "Technical debt: From metaphor to theory and practice," *IEEE Software*, Vol. 29, No. 6, 2012, pp. 18–21.

[3] A. Alazba, H. Aljamaan, and M. Alshayeb, "Deep learning approaches for bad smell detection: A systematic literature review," *Empirical Software Engineering*, Vol. 28, No. 77, 2023.

[4] T. Sharma and D. Spinellis, "A survey on software smells," *Journal of Systems and Software*, Vol. 138, 2018, pp. 158–173.

[5] N. Sae-Lim, S. Hayashi, and M. Saeki, "How do developers select and prioritize code smells? a preliminary study," in *Proceedings IEEE 33rd International Conference Software Maintenance and Evolution (ICSME)*, 2017, pp. 484–488.

[6] M. Hozano, A. Garcia, B. Fonseca, and E. Costa, "Are you smelling it? Investigating how similar developers detect code smells," *Information and Software Technology*, Vol. 93, 2018, pp. 130–146.

[7] T. Sharma, V. Efstathiou, P. Louridas, and D. Spinellis, "Code smell detection by deep direct-learning and transfer-learning," *Journal of Systems and Software*, Vol. 176, 2021, p. 110936.

[8] M.I. Azeem, F. Palomba, L. Shi, and Q. Wang, “Machine learning techniques for code smell detection: A systematic literature review and meta-analysis,” *Information and Software Technology*, Vol. 108, 2019, pp. 115–138.

[9] A. Ho, A.M. Bui, P.T. Nguyen, and A.D. Salle, “Fusion of deep convolutional and LSTM recurrent neural networks for automated detection of code smells,” in *Proceedings 27th International Conference Evaluation and Assessment in Software Engineering*, 2023, pp. 229–234.

[10] W. Xu and X. Zhang, “Multi-granularity code smell detection using deep learning method based on abstract syntax tree,” in *Proceedings International Conference Software Engineering and Knowledge Engineering*, 2021.

[11] A. Bagnall, J. Lines, A. Bostrom, J. Large, and E. Keogh, “The great time series classification bake off: A review and experimental evaluation of recent algorithmic advances,” *Data Mining and Knowledge Discovery*, Vol. 31, No. 3, 2017, pp. 606–660.

[12] M. Middlehurst, P. Schäfer, and A. Bagnall, “Bake off redux: A review and experimental evaluation of recent time series classification algorithms,” *Data Mining and Knowledge Discovery*, 2024, pp. 1–74.

[13] J. Zhang, F.Y. Wang, K. Wang, W.H. Lin, X. Xu et al., “Data-driven intelligent transportation systems: A survey,” *IEEE Transactions on Intelligent Transportation Systems*, Vol. 12, No. 4, 2011, pp. 1624–1639.

[14] Y. Zhou, Z. Ding, Q. Wen, and Y. Wang, “Robust load forecasting towards adversarial attacks via Bayesian learning,” *IEEE Transactions on Power Systems*, Vol. 38, No. 2, 2023, pp. 1445–1459.

[15] A.A. Cook, G. Mısırlı, and Z. Fan, “Anomaly detection for IoT time-series data: A survey,” *IEEE Internet of Things Journal*, Vol. 7, No. 7, 2019, pp. 6481–6494.

[16] M. Abdelkader, “A novel method for code clone detection based on minimally random kernel convolutional transform,” *IEEE Access*, Vol. 12, 2024, pp. 158 579–158 596.

[17] M. Fowler and K. Beck, *Refactoring: Improving the Design of Existing Code*. Addison-Wesley Professional, 1999.

[18] G. Suryanarayana, G. Samarthyam, and T. Sharma, *Refactoring for Software Design Smells: Managing Technical Debt*. Morgan Kaufmann, 2014.

[19] A. Dempster, F. Petitjean, and G.I. Webb, “ROCKET: Exceptionally fast and accurate time series classification using random convolutional kernels,” *Data Mining and Knowledge Discovery*, Vol. 34, No. 5, 2020.

[20] A. Dempster, D.F. Schmidt, and G.I. Webb, “MiniRocket: A very fast (almost) deterministic transform for time series classification,” in *Proceedings International Conference Knowledge Discovery and Data Mining*, 2021, pp. 248–257.

[21] B. Nyirongo, Y. Jiang, H. Jiang, and H. Liu, “A survey of deep learning based software refactoring,” *arXiv preprint arXiv:2404.19226*, 2024.

[22] T. Sharma and M. Kessentini, “QScored: a large dataset of code smells and quality metrics,” in *Proceedings IEEE/ACM 18th International Conference Mining Software Repositories (MSR)*, 2021, pp. 590–594.

[23] Y. Zhang, C. Ge, H. Liu, and K. Zheng, “Code smell detection based on supervised learning models: A survey,” *Neurocomputing*, Vol. 565, 2024, p. 127014.

[24] R.S. Pressman, *Software Engineering: A Practitioner’s Approach*, 6th ed. Palgrave Macmillan, 2005.

[25] N.E. Fenton, *Software Metrics – A Rigorous Approach*. London: Chapman and Hall, 1991.

[26] R. Marinescu, “Detection strategies: Metrics-based rules for detecting design flaws,” in *Proceedings 20th International Conference Software Maintenance (ICSM)*, Chicago, IL, USA, 2004, pp. 350–359.

[27] R. Marinescu, “Measurement and quality in object-oriented design,” in *Proceedings 21st IEEE International Conference Software Maintenance (ICSM)*, Budapest, Hungary, 2005, pp. 701–704.

[28] I.M. Bertran, A. Garcia, and A. von Staa, “Defining and applying detection strategies for aspect-oriented code smells,” in *24th Brazilian Symposium on Software Engineering, SBES*, Salvador, Bahia, Brazil, 2010, pp. 60–69.

[29] A.M. Fard and A. Mesbah, "JSNOSE: Detecting JavaScript code smells," in *Proceedings 13th IEEE International Working Conference Source Code Analysis and Manipulation (SCAM)*, Eindhoven, Netherlands, 2013, pp. 116–125.

[30] Z. Chen, L. Chen, W. Ma, X. Zhou, Y. Zhou et al., "Understanding metric-based detectable smells in Python software: A comparative study," *Information and Software Technology*, Vol. 94, 2018, pp. 14–29.

[31] N. Moha, Y.G. Guéhéneuc, L. Duchien, and A.F.L. Meur, "DECOR: A method for the specification and detection of code and design smells," *IEEE Transactions on Software Engineering*, Vol. 36, No. 1, 2010, pp. 20–36.

[32] F.A. Fontana, M.V. Mäntylä, M. Zanoni, and A. Marino, "Comparing and experimenting with machine learning techniques for code smell detection," *Empirical Software Engineering*, Vol. 21, No. 3, 2016, pp. 1143–1191.

[33] A. Maiga, N. Ali, N. Bhattacharya, A. Sabané, Y.G. Guéhéneuc et al., "Support vector machines for anti-pattern detection," in *Proceedings 27th IEEE/ACM International Conference Automated Software Engineering (ASE)*, 2012, pp. 278–281.

[34] F. Khomh, S. Vaucher, Y.G. Guéhéneuc, and H. Sahraoui, "A Bayesian approach for the detection of code and design smells," in *Proceedings 9th International Conference Quality Software (QSIC)*, 2009, pp. 305–314.

[35] J. Kreimer, "Adaptive detection of design flaws," *Electronic Notes in Theoretical Computer Science*, Vol. 141, No. 4, 2005, pp. 117–136.

[36] M. Škipina, J. Slivka, N. Luburić, and A. Kovačević, "Automatic detection of code smells using metrics and codeT5 embeddings: A case study in C," *Neural Computing and Applications*, 2024, pp. 1–18.

[37] M. Hadj-Kacem and N. Bouassida, "A hybrid approach to detect code smells using deep learning," in *Proceedings International Conference Evaluation of Novel Approaches to Software Engineering*, 2018.

[38] H. Liu, Z. Xu, and Y. Zou, "Deep learning-based feature envy detection," in *Proceedings 33rd IEEE/ACM International Conference Automated Software Engineering (ASE)*, 2018, pp. 385–396.

[39] B. Liu, H. Liu, G. Li, N. Niu, Z. Xu et al., "Deep learning-based feature envy detection boosted by real-world examples," in *Proceedings 31st ACM Joint European Software Engineering Conference and Sympossium Foundations of Software Engineering (ESEC/FSE)*, 2023, pp. 908–920.

[40] A.K. Das, S. Yadav, and S. Dhal, "Detecting code smells using deep learning," in *Proceedings TENCON – IEEE Region 10 Conference*, 2019, pp. 2081–2086.

[41] D. Yu, Y. Xu, L. Weng, J. Chen, X. Chen et al., "Detecting and refactoring feature envy based on graph neural network," in *Proceedings IEEE 33rd International Symposium Software Reliability Engineering (ISSRE)*, 2022, pp. 458–469.

[42] H. Zhang and T. Kishi, "Long method detection using graph convolutional networks," *Journal of Information Processing*, Vol. 31, Aug. 2023, pp. 469–477.

[43] Y. Zhang, C. Ge, S. Hong, R. Tian, C.R. Dong et al., "DeleSmell: Code smell detection based on deep learning and latent semantic analysis," *Knowledge-Based Systems*, Vol. 255, 2022, p. 109737.

[44] Y. Zhang and C. Dong, "MARS: Detecting brain class/method code smell based on metric – attention mechanism and residual network," *Journal of Software: Evolution and Process*, 2021.

[45] Y. Li and X. Zhang, "Multi-label code smell detection with hybrid model based on deep learning," in *Proceedings International Conference Software Engineering and Knowledge Engineering*, 2022.

[46] H. Liu, J. Jin, Z. Xu, Y. Zou, Y. Bu et al., "Deep learning based code smell detection," *IEEE Transactions on Software Engineering*, Vol. 47, No. 9, 2019, pp. 1811–1837.

[47] K. Alkharabsheh, S. Alawadi, V.R. Kebande, Y. Crespo, M.F. Delgado et al., "A comparison of machine learning algorithms on design smell detection using balanced and imbalanced dataset: A study of god class," *Information and Software Technology*, Vol. 143, 2022, p. 106736.

[48] P. Probst, B. Bischl, and A.L. Boulesteix, "Tunability: Importance of hyperparameters of machine learning algorithms," *arXiv preprint arXiv:1802.09596*, 2018.
[49] L. Li, K. Jamieson, G. DeSalvo, A. Rostamizadeh, and A. Talwalkar, "Hyperband: A novel bandit-based approach to hyperparameter optimization," in *Proceedings International Conference Learning Representations*, 2018, pp. 1–48.
[50] L. Li, K. Jamieson, A. Rostamizadeh, E. Gonina, J. Ben-Tzur et al., "A system for massively parallel hyperparameter tuning," in *Proceedings Machine Learning and Systems*, Vol. 2, 2020, pp. 230–246.
[51] S.H. Walker and D.B. Duncan, "Estimation of the probability of an event as a function of several independent variables," *Biometrika*, Vol. 54, No. 1–2, 1967, pp. 167–179.
[52] T. Chen and C. Guestrin, "XGBoost: A scalable tree boosting system," in *Proceedings International Conference Knowledge Discovery and Data Mining*, 2016, pp. 785–794.
[53] T.K. Ho, "Random decision forests," in *Proceedings 3rd International Conference Document Analysis and Recognition*, 1995, pp. 278–282.
[54] J.G.H. John and P. Langley, "Estimating continuous distributions in Bayesian classifiers," *arXiv preprint arXiv:1302.4964*, 2013.
[55] F. Yang, "An extended idea about decision trees," in *International Conference on Computational Science and Computational Intelligence (CSCI)*, 2019, pp. 349–354.
[56] Student, "The probable error of a mean," *Biometrika*, Vol. 6, 1908, pp. 1–25.
[57] L.V. Hedges, "Estimation of effect size from a series of independent experiments," *Psychological Bulletin*, Vol. 92, No. 2, 1982, pp. 490–499.
[58] J. Cohen, *Statistical power analysis for the behavioral sciences*, 2nd ed. Routledge, 1988.
[59] P.D. Ellis, *The essential guide to effect sizes: Statistical power, meta-analysis, and the interpretation of research results.* Cambridge University Press, 2010.
[60] J.M. Hoenig and D.M. Heisey, "The abuse of power: The pervasive fallacy of power calculations for data analysis," *The American Statistician*, Vol. 55, No. 1, 2001, pp. 1–6.
[61] F.J. Massey, "The Kolmogorov–Smirnov test for goodness of fit," *Journal of the American Statistical Association*, Vol. 46, No. 253, 1951, pp. 68–78.

Authors and affiliations

Mostefai Abdelkader
e-mail: abdelkader.mostefai@univ-saida.dz
ORCID: https://orcid.org/0000-0003-0408-331X
Department of Computer Science,
University of Saida, Dr. Taher Moulay, Algeria

Mekour Mansour
e-mail: mansour.mekour@univ-saida.dz
ORCID: https://orcid.org/0000-0001-8486-9114
Department of Computer Science,
University of Saida, Dr. Taher Moulay, Algeria

e-Informatica Software Engineering Journal
Volume 19, Issue 1, pages: 1–29
DOI: 10.37190/e-Inf250107

BibTeX

Challenges of Requirements Communication and Digital Assets Verification in Infrastructure Projects

Waleed Abdeen*, Krzysztof Wnuk, Michael Unterkalmsteiner, Alexandros Chirtoglou
*Corresponding author: waleed.abdeen@bth.se

Article info

Keywords:
infrastructure
requirements
digital assets
verification
validation

Submitted: 11 Mar. 2025
Revised: 22 Jun. 2025
Accepted: 23 Jul. 2025
Available online: 30 Jul. 2025

Abstract

Background: Poor communication of requirements between clients and suppliers contributes to project overruns,in both software and infrastructure projects. Existing literature offers limited insights into the communication challenges at this interface.
Aim: Our research aim to explore the processes and associated challenges with requirements activities that include client-supplier interaction and communication.
Method: we study requirements validation, communication, and digital asset verification processes through two case studies in the road and railway sectors, involving interviews with ten experts across three companies.
Results: We identify 13 challenges, along with their causes and consequences, and suggest solution areas from existing literature.
Conclusion: Interestingly, the challenges in infrastructure projects mirror those found in software engineering, highlighting a need for further research to validate potential solutions.

1. Introduction

Developing software products and releasing them to the market is a complex process that requires efficient communication [1]. Inefficient communication manifests itself as communication gaps that lead to quality issues, wasted efforts, delays, and ultimately to failure to meet the customers' expectations [2]. Moreover, the inefficient use of requirements artifacts, i.e., information is not tailored to the needs of the involved stakeholders, may impede requirements communication in software projects [3]. What is common for large software and infrastructure projects is their complexity that often implies the use of suppliers [4] to deliver significant parts of the solution, e.g., automotive industry often uses suppliers to develop software for cars.

The friction-free economic principles of the software industry help involve suppliers from geographically distant locations, primarily for cost reduction and the availability of competence. One of the main challenges in involving suppliers is establishing good communication principles, especially agreed quality levels, and also communicating potential difficulties and issues early [5]. In particular, the verification of supplier deliverables against client requirements poses challenges [6]. A large contributor to this challenge is the inherent uncertainty in requirements, which increases the negative influence of coordination efforts on trust for new product development projects [7]. In project management theory, it has been suggested that both goal and resource interdependence between clients and suppliers have a positive influence on coordination and cooperation in new software development programs [8].

Infrastructure projects often last decades in planning and execution [9], and involve a large number of specialized project teams (e.g., owners, contractors, consultants, designers) with specific core competencies. These projects are also featured with uncertainty, fragmentation [10], and high complexity and inter-organizational task interdependence, which makes communication ever more important [11]. Because contracts of construction projects are inherently incomplete [12], understanding which challenges arise in client-supplier communication in design-build projects becomes critical.

In practice, the ability to communicate clients' needs and requirements to suppliers is a key success criterion in infrastructure projects [13–15]. Effective communication facilitates stakeholder engagement during the change management process [9] and clarifies realistic stakeholder expectations, while insufficient communication drives uncontrolled change and can contribute to project delays [16].

Several studies have explored the challenges associated with general communication in large construction projects [17], requirements allocation [18], verification and validation [19], system integration [20], the relationship between the communication-conflict interaction and project success [10], trust between firms and suppliers [21], encoding and decoding communication competencies in project management [22], or inter-cultural communication [23].

However, little research effort has been dedicated to exploring the processes and associated challenges with requirements activities that include client-supplier interaction and communication in infrastructure projects. Requirements communication is the process of communicating – both verbally and through documents – requirements and project-related documents between the client and the supplier. Requirements validation is the process of ensuring that the requirements reflect the client's needs before designing or investing in building the system. Digital assets verification ensures that the delivered assets (e.g., 3D drawings) correspond to the requirements specifications. Figure 1 illustrates the verification and validation processes. It is important to note that in the verification processes (requirements verification and system verification), the client is not directly involved. As for the validation processes, the client input is important to ensure delivery of what the

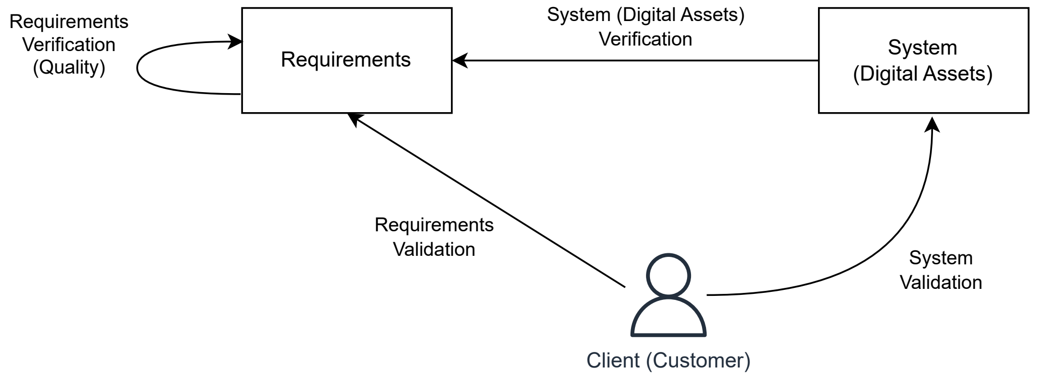


Figure 1. Verification and validation processes

client wants. In infrastructure projects, tender documents, written by the client, are the basis from which suppliers derive requirements specifications that need to be validated.

This study is part of a research project that aims to investigate methods that can improve functional requirements traceability at different stages in construction projects. Functional requirements should be linked to a digital twin [24], a collection of digital assets that represent the facility and are linked to the physical facility. Digital assets are defined as any digital file (textual, images, audio, or video) stored on any electronic device (e.g., computer, mobile phone, or cloud) with the right to own the file [25]. In infrastructure projects, Computer-Aided Design models (CAD), Building Information Models (BIM), and System Information Models (SIM) [26] are typical digital assets. Traceability between functional requirements and digital assets would allow for efficient and effective digital assets verification before construction has started, enable monitoring during construction, as well as make it possible to follow-up on the original requirements during the decades of operation and maintenance.

This study explores the nature of client-supplier requirements communication in large infrastructure design-build projects, where the supplier is responsible for the design, construction, and requirements validation and digital assets verification processes. Such a description can help to understand the challenges that practitioners encounter and either point to solutions that exist in the literature or describe a research gap.

We frame our research using the design science process [27,28], which aims to extend knowledge by identifying and addressing specific problems in practice. We conducted two case studies where we explored the client-supplier interface. In the two projects, Anonymous acted as client and two other companies acted as contractor and sub-contractor. The semi-structured interviews were designed to understand *how* requirements communication, requirements validation, and the verification of deliverables are conducted. We transcribed the audio recordings and coded the data to answer the research questions.

We present an overview of the client-supplier interface and develop process diagrams for each investigated case. Furthermore, we identify and describe the challenges faced by the managers and engineers working in these processes. Finally, we map these challenges to potential solutions from the system and software engineering domain. *Identify conflicts early*, *requirements abstraction*, *requirements are impossible to build* and *granularity of traces*, are challenges that are also faced in software engineering domain and are important to address.

The remainder of this paper is structured as follows. We present in Section 2 literature that is related to our work. In Section 3, we present our research methodology and study design. We present our research results in Section 4 and discuss them in Section 5. In Section 6, we draw a conclusion from this study. And finally we list future work directions in Section 7.

2. Related work

The focus of our investigation is the processes related to requirements engineering on the interface between clients and suppliers in the context of construction projects. In this section, we discuss, therefore, related work from systems engineering and communication in large construction project research areas.

2.1. Communication in software engineering

Communication in software engineering is essential in any project as it affects its outcome. Many researchers have investigated the associated challenges with communication in industry [3, 29, 30]. Bjarnason et al. [29] investigated geographical, cognitive, and psychological distances in communication between teams and identified ten factors that affect communication. Liskin [3] investigated how artifacts support or impede requirements communication. They identified five challenges related to link requirements. Liebel et al. [30] studied communication problems in automotive requirements engineering, concluding that it is important to establish communication channels outside the fixed organization structure. Other researchers [31–33] proposed solutions to tackle these challenges. Iqbal et al. [31] developed a framework to address communication issues during the requirements engineering process for software development outsourcing. Fricker et al. [32] suggested handshaking with implementation proposals that could be used to improve the communication between the client and the supplier. Pernstal et al. [33] developed the BRASS framework to support coordination and communication of inter-departmental requirements in the large-scale development of software-intensive systems. However, the framework does not focus on requirements validation but rather on supporting the selection of the most suitable implementation plan. Although these studies investigate (requirements) communication challenges and propose solutions to them, they mainly focus on inter-organization communication, in contrast to our study, where we focus on requirements communication in the client-supplier interface.

2.2. System engineering

RE and verification are of high interest to the system engineering community, which can be seen by the several conducted studies covering different system engineering processes and their challenges [18, 20, 34–36].

Madni et al. [20] explored the challenges in system integration in the defense and aerospace domain. They propose an ontology-based system integration approach to facilitate communication between the different stakeholders. Makkinga et al. [19] conducted semi-structured interviews in three mid-sized projects in the Netherlands. Their focus was on the verification and validation problems in construction projects and possible solutions to these problems. They advise more research work in the validation process. Vermillion et al. [18] compare the requirements allocation and objective allocation in projects with outsourced design. They propose an approach for requirements allocation for the design process. Moreover, they see an asymmetry in knowledge between the clients who outsource the design and agents designing the system, which needs to be addressed to reduce negotiation iterations.

System engineering processes and practices in the construction domain have also been of interest to researchers [35, 36]. Lynghaug et al. [35] explored the state of system engineering practices in the Norwegian construction domain. They conclude by providing recommendations on implementing and improving system engineering practices in the construction domain. One of the most critical system engineering processes is requirements analysis. Raatikainen et al. [37] highlighted the challenge of efficient communication and management of requirements in the nuclear energy domain. De Graaf et al. [36] investigated the state of practice of six cases of sub-contracted design work in civil engineering projects. They identified three factors that affect the sub-contracted work, mainly: Building Information Model (BIM) interoperability, time pressure, and employee availability.

2.3. Communication in large construction projects

In large construction projects, communication is essential to the success of the project, and conflicts could lead to communication breakdowns and project failure as illustrated in multiple studies [9, 10, 17, 38]. Wu et al. [10] investigated the relationship between the communication-conflict interaction and project success. Malik and Taqi [17] explored the relationship between communication and the success of construction projects. They found that process conflict and relationship conflict have a negative impact on communication and project success. Saxena and McDonagh [38] have investigated communication breakdowns and concluded that change management requires a multilevel communication approach. Furthermore, communication could be an enabler to other activities in the project. Butt et al. [9] have looked into how effective communication facilitates stakeholder engagement during the change management process and changing project culture. Communication is a powerful tool in ensuring participation in change management processes. Lack of communication leads to teams focusing on task performance and efficiency rather than empowerment and involvement.

Communication is even more essential in global projects, where cultural changes added more complexity, as it continues to shows [22, 23, 39–41]. Loosemore and Muslmani [23] investigated intercultural communication challenges in Persian Gulf projects, highlighting the need for a better understanding of cultural diversity. Similar recommendations were reported by Ochieng and Price, who studied the cultural variation of project managers in Kenya and the UK in communicating effectively on multicultural projects [39]. Henderson et al. [41] looked into the impact of communication norms on global project teams and individuals' project satisfaction and performance. In another study [22], the researchers also looked into encoding and decoding communication competencies in project management. Daim et al. [40] have also looked into communication breakdowns among global virtual teams, and these breakdowns tend to threaten project delivery. The authors found five factors impacting communication breakdowns: trust, interpersonal relations, cultural differences, leadership and technology.

These studies show the importance and consequences of communication in construction projects. However, they are mainly focused on people's communications rather than artifacts of communication, which we focus on in our study.

2.4. Research gap

Requirements validation, communication and digital assets verification are important in avoiding project failures [15, 42]. While researchers from the software engineering domain have investigated the area [43, 44], it remains greatly unexplored or neglected in the system engineering domain. Hence, due to the lack of studies that investigate requirements validation, requirements communication and digital assets verification in the system engineering domain, we are conducting this study to fill this gap.

3. Methodology

Due to the aim of the research project as a whole, which is to identify and address problems related to the requirements communication process and the traceability of functional requirements to the digital twin, we frame our research as design science problem [27, 28]. Hevner [45] has depicted design science as a process intertwining relevance, rigor, and design cycles. In the study reported in this paper, we investigate the challenges of client-supplier communication in system engineering projects; in other words, we identify the *relevance* of the problem in the infrastructure domain. Thereafter, we plan to design a solution for one of the problems we identify in this study, apply it to the problem, and further improve it (*design*). Finally, we plan to verify that the solution fits the problem through implementation to other problem instances from the software engineering domain (*rigor*). In this study, we focus only on determining the *relevance* of the problem.

We designed the study according to the case study guidelines by Runeson et al. [46], who advocate for defining a detailed case study protocol that reflects the changes made during the iterative process of data collection and analysis. The research questions with the motivation and alignment to our objective are listed in Table 1.

Table 1. Research questions

Id	Research question	Motivation
RQ1	What are the practices and challenges in system requirements validation?	One of the main documents that are communicated between the client and the supplier in design-build contracts are system requirements. Hence it is important to have those requirements validated. We explore the system requirements validation process and the challenges associated with it.
RQ2	What are the practices and challenges in system requirements communication between the client and the supplier?	Explore the requirements communication process between the client and the supplier, and what different formats the requirements take before they are translated into a design. Moreover, we want to explore the difficulties in the process and opportunities for improvements.
RQ3	What are the practices and challenges in digital assets verification?	Explore any quality checks done by the client or the supplier on the design documents and any challenges associated with the process.

3.1. Case description

The two cases were selected from *Anonymous'* projects based on availability and access to information. Both projects distinguish between project-specific requirements and regulatory requirements. Project-specific requirements describe needs that pertain to the particular facility, originating from diverse stakeholders such as the government, communes, and land owners. Regulatory requirements define needs that are relevant to all applicable facilities. While the client specifies the project-specific requirements, it is the responsibility of the supplier to identify and comply with the relevant regulatory requirements for the designed facility.

Case One is a road project that includes the design and building of roads and bridges for cars, pedestrians, and cyclists. At the time of the study (April-June 2020), the project was in construction (3 years duration). Case One, with a budget of 35M USD, is part of a larger road project with an estimated duration of 30 years (15 planning, 15 construction) and a budget of 4B USD. Case Two has specified approximately 700 project-specific requirements and is associated with 12 000 regulatory requirements.

Case Two is a railway project with the design and build of a high-speed train connection, including rail, bridges, and tunnels. The project was in its early stages (requirements specification and initial design) at the time of the study. The estimated project duration for Case Two is 34 years (9 pre-study, 10 planning, 15 construction) and a budget of 8.8B USD. Case Two has specified approximately 1 000 project-specific requirements and is associated with approximately 15 000 regulatory requirements[1].

3.2. Data collection

We conducted semi-structured interviews with the participants from all three companies listed in Table 2. Semi-structured interviews are fit for a study where a clear hypothesis does not exist and the research questions are of explorative nature [46]. In our case, we explored the system engineering processes that involve client-supplier communication, but we don't know to what extent these processes were implemented.

Table 2. Companies overview

Company	Industry	Role	Case	Company size
A	Construction	Contractor	Case One	45 000
B	Design	Sub-Contractor	Case One	6 000
C	Transportation and Infrastructure	Client	Case One, Case Two	9 400

The unit of measure for company size is the average number of employees.

We formulated the interview questions in three main themes, according to our stated research questions: *requirements validation*, *requirements communication*, and *digital assets verification*. Then we adapted the interview questions to each interviewee role. For example, when we interviewed a requirements engineer, we focused our questions on *requirements validation* and *requirements verification*. During each interview, we asked (on average) 14 pre-defined questions about the topics, excluding follow-up questions. We present here a few examples of the questions that we asked during the interviews: on the topic of *requirements communication* we asked questions such as "Q: How do you receive the requirements specified by the client?" and "Q: Do you experience any challenges in requirements' communication?", on the topic of *requirements validation* we asked "Q: Explain the requirements validation process?", and in the area of *digital assets verification* we asked "Q: What is the next step after the digital assets are produced?", and "Q: How do you ensure that the produced digital assets satisfy the requirements".

We interviewed ten people from three companies. Table 3 shows their roles, companies, and industry experience. The interviewees had between 7 and 24 years of experience and worked in roles related to requirements engineering, design, and project management. Convenience sampling was used when choosing the participants for the interviews. We asked our company contacts for people who work with requirements and system design, then they suggested people based on their availability. In Case 2, we only interviewed

[1]In both projects, the number of project-specific requirements varied over time. Furthermore, the number of associated regulatory requirements is an estimate by a requirements engineering lead at *Anonymous*. Determining the exact numbers would require that every contractor keeps track of the regulations they need to comply to, which is, as we shall see, not the case.

people from the client side as we didn't have access to suppliers. In this paper, we use the acronym XY to reference interviewees, where X is a letter referencing the company [A,B,C] and Y is a number referencing an interviewee in that company.

Table 3. Interviewees roles

Id	Role	Company	Experience
A1	Design Manager	A	20+ years
A2	Tender Manager	A	10 years
A3	BIM Manager	A	7 years
A4	Design Manager	A	10 years
B1	Discipline Leader	B	15 years
B2	Head of Design	B	25 years
C1	BIM Specialist	C	9 years
C2	Requirements Specialist	C	11 years
C3	Requirements Engineer	C	8 years
C4	Head of Tech and Environment	C	24 years

The experience presented in this table is the overall industry experience.

Two to three researchers attended the interviews (observer triangulation). The first author led the interview by asking questions, and the other two authors observed and asked follow-up questions if necessary. We conducted each interview as follows:

1. A week before the interview, we sent the informed consent letter to the participants and asked them to return a signed copy within two weeks after the interview.
2. Before the interview started, we briefly described the purpose of our study, and the expected outcome and explained how we are going to conduct the interview.
3. We started recording the interview and asked the questions.

After we finished conducting the interviews, the first author transcribed them. The third author listened to the interviews' recordings to verify the transcript. We sent each transcript out to the interviewees for comments. We received corrections from one participant, clarifying statements that were incomplete due to recording issues caused by the recording equipment.

3.3. Data analysis

We coded the interview transcripts with an initial set of codes that the first and third author created in alignment with the research questions. The first author quoted and labeled parts of the transcript using these codes. Feedback sessions were conducted between the first and third author to refine the codes. Several coding iterations were done until we had enough data to answer the research questions. A final verification of the quoted text was done by the third author. The codes were created on different levels of abstraction, as seen in Figure 2. Initially, we started with 14 abstract codes: *artifacts, roles, tolls, processes, important statements and challenges, digital assets implementation, digital assets verification, requirements elicitation, requirements specifications, requirements communication, requirements change, and requirements validation.* We derived compound codes by combining two or more abstract codes (e.g., requirements validation-process). Then, we added additional codes as needed. This helped us to use relevant codes when answering the research questions, as exemplified next.

Coding example When coding the answers to the interview questions related to the requirements validation process, we started with the codes: 1) *requirements validation*

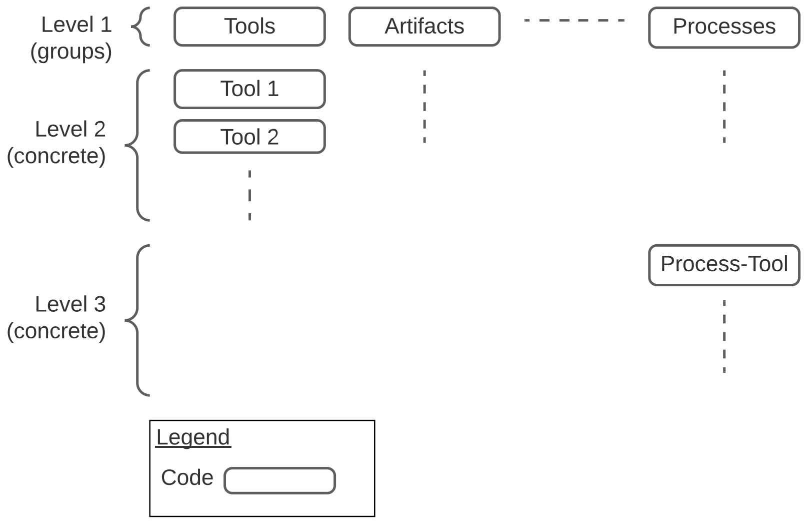


Figure 2. Coding levels

process to quote the activities associated with the requirements validation process and 2) *requirements validation challenge*, to quote the challenges of that process. These codes were part of the group *requirements validation.* Then we added more codes based on the content, e.g., *requirements documentation* and *cost estimation* to quote the artifacts used in the validation process. After that, we created codes on a higher abstraction level, e.g., *tools*, *artifacts*, and *processes.*

Using this way of coding resulted in codes on different abstraction levels. This helped retrieve all information in a whole area, e.g., the requirements validation process, and in detail in that process, e.g., actors in the requirements validation process.

4. Results

We present the results in three sections following the themes of the interview questions: requirements validation, requirements communication, and digital assets verification. In each section, we describe the process and the challenges faced in that process. A challenge is presented as the description, causes, and consequences of that challenge. Some challenges have no causes or/and consequences since we could not identify them from the interviews. We illustrate the results in figures that show the process flow, artifacts, and actors. Each figure is divided into columns representing the parties involved in the project: client, contractor, or subcontractor. In some figures, there is a question mark which is a placeholder for information that was not clear from the interviews.

4.1. Requirements validation

Figure 3 depicts the requirements validation process in Case One. At the contractor, the tender manager and their team conduct a review process. The inputs to the process are the technical description and the general regulations (local to the country) that apply to the project. The output of the process is a cost estimate spreadsheet, which estimates the project cost and is used when preparing the contract for the project. Another requirements review process is conducted at the subcontractor by the discipline leaders and designers.

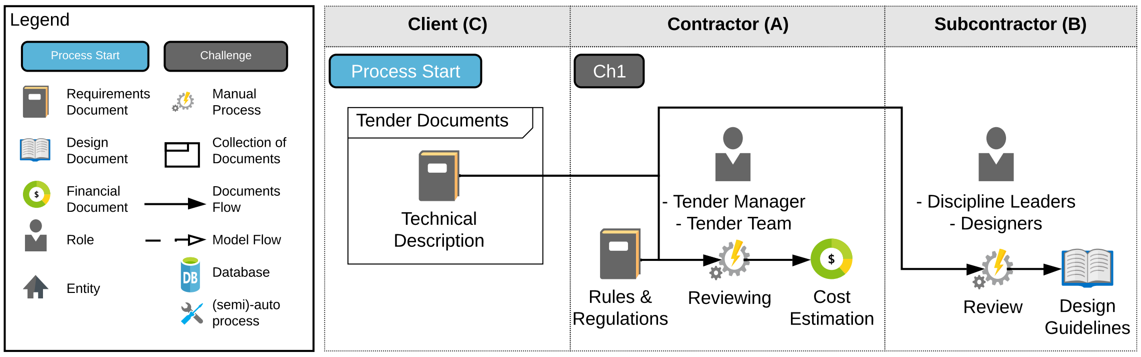


Figure 3. Requirements validation – Case one, the abbreviation (ch) refers to challenge, both the abbreviations and legend apply to all figures

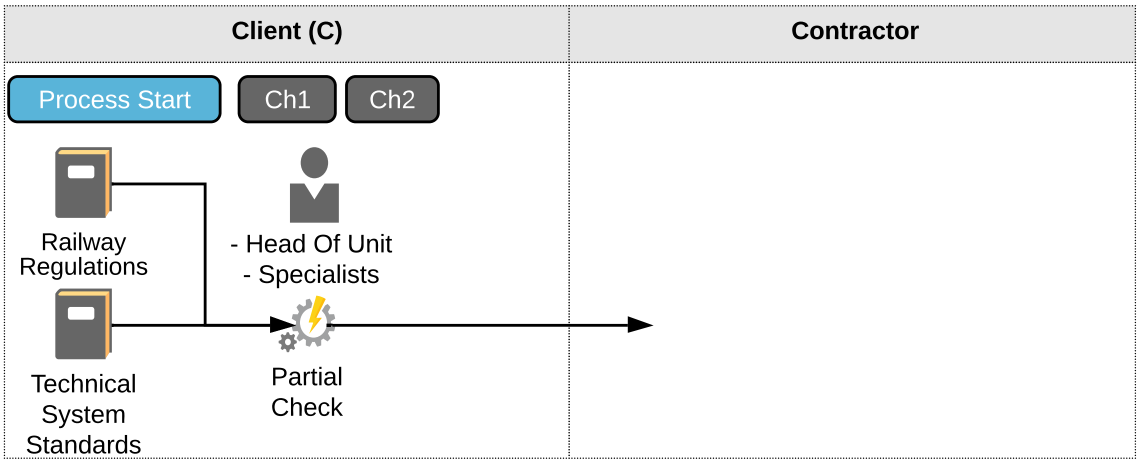


Figure 4. Requirements validation – Case two

They take as input the technical description from the contractor and the general regulations. The output of this process is the design guidelines document that helps designing the models for the project.

Figure 4 shows the requirements validation process in Case Two. The client conducts a partial check on the requirements. The inputs to this check are the railway regulations and technical system standards and the outputs are the validated requirements. We do not have information about the requirements validation process on the contractor side, because the people we interviewed in Case Two are from the client-side.

We have identified the following two challenges that are faced by the roles involved in the requirements validation process.

4.1.1. Challenge 1: Prioritizing the requirements validation process

Prioritizing the requirements validation process is a challenge faced by the contractor in Case One and the client in Case Two. The time to validate the requirements by the contractor is seen to be short in Case One, as reported by the tender manager (A2). When the tender project is announced, the contractor has to respond with the cost estimate in a matter of weeks. This makes the validation of the requirements challenging for the contractor who has to do it in a short time. “The client spends four years coming up with the design (presented by the requirements), and we have to price it in four or eight weeks, coupled with a tender model that is purely price-driven. It is a very unhealthy situation" (A2). The requirements validation process requires experience and knowledge in the discipline, which makes the prioritization of this process and finding the right people to perform it difficult for the client. "When you talk about validating requirements, it is not our top priority" (C2).

Ambiguous and conflicting requirements is a cause of this challenge in Case One. The requirements validation process could take more than a couple of months when the supplier finds requirements that need clarification or revision by the client. In Case Two, *lack of resources* is seen as a cause of this challenge. As C2 said, "there are not enough railway specialists in the company." The people involved in Case Two are involved in other projects and have other responsibilities at the client, making it difficult to find the right people to validate the requirements.

The requirements validation process is overlooked is seen by the client as a consequence of not prioritizing the said process. Since it is not always feasible to allocate resources for the validation process, it gets less priority over other activities, leading to the process being skipped in many cases. Furthermore, the contractor sees *added responsibility* as a consequence. Since the contractor submits the bid to the project based on the requirements, they take the responsibility if they have not validated the requirements properly: "Do not put all the risk on us if we can't find the problem in such a short time" (A2).

4.1.2. Challenge 2: Identify conflicts early

It could be difficult to identify conflicts in the requirements early as seen in Case Two by the requirements engineer (C3) and head of environment (C4). Although a partial check on the requirements is conducted at the client side, some conflicts may go undetected until later at the design stage. For example, when verifying tunnel-related requirements, one (requirements engineers or designers) may not be able to identify water pipes intersection with an obstacle due to the land geometry. "It is hard to see that before when you haven't drawn the lines for the pipes or the tunnel" (C3).

The main cause of this challenge is not clear. When we asked C3 whether the lack of information or having too much information is the cause of difficulties identifying those conflicts, they answered "it depends" (C3).

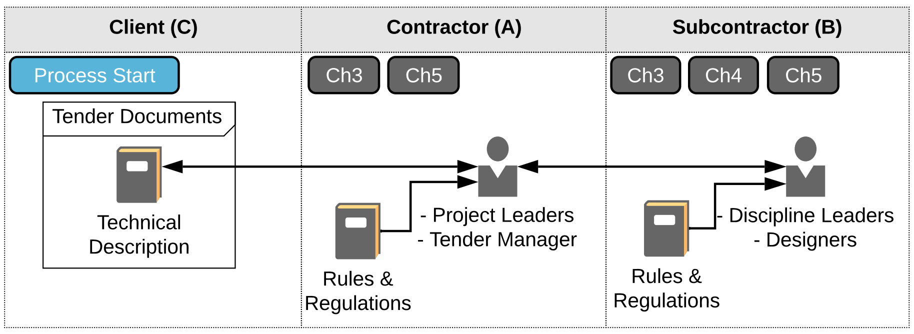


Figure 5. Requirements communication – Case one

4.2. Requirements communication

Figure 5 shows the requirements communication process in Case One. The project's technical requirements, called technical description document, are a part of the tender documents. The project leaders and tender manager on the contractor side get the technical description from the client, and then they communicate these documents to the discipline leaders and designers on the subcontractor side. Additional requirements from the rules and regulations apply to the project in Case One. The contractor and subcontractor are responsible for finding those additional requirements that apply to the project.

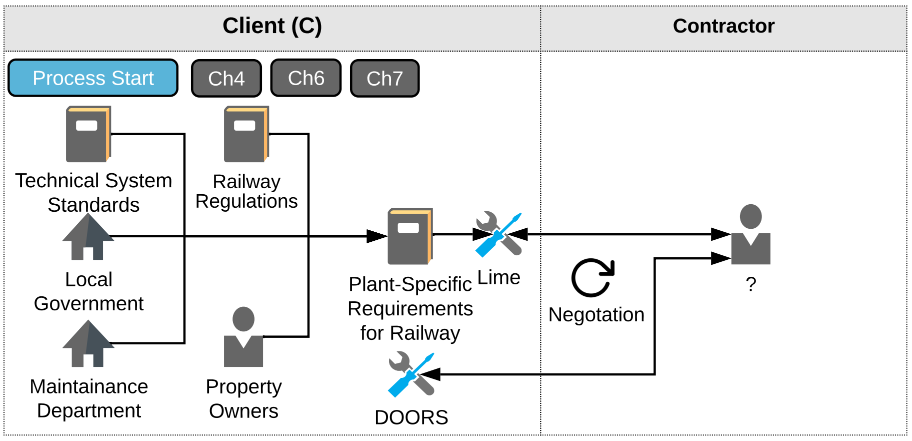


Figure 6. Requirements communication – Case two

Figure 6 shows the requirements communication process in Case Two. In this case, there are three main types of requirements sources: 1) documents, e.g., technical system standards, or railway regulations; 2) internal or external departments, e.g., local government or maintenance department; and 3) people who reside or work in the project area and who could be affected by the project outcome, e.g., property owners. The client compiles the requirements from all the mentioned sources into project-specific requirements for the railway. Those requirements are stored in a software called LIME, which is used to negotiate requirements between the client and the contractor. The documents stored in LIME are living documents, where both the client and supplier communicate the requirements and their feedback through an iterative process. After the client and the contractor have agreed on the requirements, those requirements are stored in DOORS[2].

The interviewees reported five challenges in relation to requirements communication.

4.2.1. Challenge 3: Misinterpretation of requirements

All parties working on the project have access to the same requirements documents, however, each party has their interpretation that could differ from the other parties' interpretation. This was reported in both cases by the design manager (A1), tender manager (A2), and discipline leader (B1). Moreover, the requirements can be unclear or difficult to understand. As one interviewee on the subcontractor side mentioned, "we have had a whole lot of discussions regarding these requirements in the technical description, and it is constantly not crystal clear, so it is very much up to interpretation" (B1). Also, the client thinks that

[2]Requirements management software.

they could improve their writing; as C4 said, “we often think that we are quite clear in our communication, but often it’s not the case” (C4).

Lack of knowledge in local projects and *requirements are open for interpretation* are of the causes of this challenge, as seen by the interviewees. Some people working on the project may lack experience in projects from the this specific country, which could lead to a different interpretation of the requirements. “If you have any country A engineer and you want to make a bridge design, his background is slightly or completely different from the country B engineer” (A2). The subcontractor sees it as one of the causes of misinterpretation of requirements as put by one interviewee “There is never really a correct answer. There is not just one solution that you can do” (B1).

The misinterpretation of requirements may lead to rework and extra costs. If the subcontractor’s interpretation is different from the client’s interpretation, then additional work should be done by the subcontractor. As B1 said, “if our interpretation of the requirements is not that what the client wants us to do, then there is some additional work for us” (B1). In special cases, the misinterpretation of requirements could lead to extra costs paid by the contractor, as confirmed by A1, “I was not informed at all what a requirement (temporary use of land for the project) means”.

4.2.2. Challenge 4: Long time to communicate requirements (changes and questions)

The requirements communication process between the different parties (client-supplier) takes a long time. This process includes requirements change requests and any requirements-related questions raised by the contractor or subcontractor. This challenge was seen in Case One and Case Two by the discipline leader (B1) and the head of environment (C4). All requirements-related communications between the client and subcontractor go through the contractor, which takes a long time: “it could take quite a bit of time” (B1). The client also sees that the change process takes a long time from the contractor side as well. C4 gave an example of a case, “one occasion we did a very big change of the requisition of the whole project… but then it took like a year for the consultants to respond and tell us how the changes would be interpreted and implemented” (C4).

The interviewees did not explicitly identify the causes or consequences of this challenge.

4.2.3. Challenge 5: Finding the correct information

The documents communicated by the client do not include all the requirements that apply to the project, and it is challenging to find all requirements that apply by the contractor and subcontractor. This challenge was reported in Case One by the design manager (A4) and head of design (B2). As seen in Figure 5 and Figure 6, there are additional requirements documents that the client does not communicate as part of the tender documents. Those documents are local rules and regulations. The client only refers to those documents, and the contractor and subcontractor need to find those documents. “The tricky task to make sure you have all requirements that have to be followed” (A4).

A cause of this challenge is *Lack of knowledge in local projects*, similar to that of miss-interpretation of requirements challenge explained in Section 4.2.1. Some people with different background could lack experience in projects in a specific country. In this case, people with experience in local projects are consulted to find the right information.

4.2.4. Challenge 6: Requirements elicitation and validation with non-technical stakeholders

The requirements elicitation and validation process is a challenging task since many stakeholders are involved. This challenge was reported in Case Two by a requirements engineer (C3). The client elicits requirements from many sources and stakeholders, as seen in Figure 6. Therefore, the way the client communicates the requirements needs to be adapted to the audience. "We can have a super model to communicate with the supplier, but we can't use that one when we meet the restaurant owner in the city" (C3).

The cause of seeing the requirements elicitation and validation process as a challenging task is because there are *different parties and many stakeholders with different backgrounds* involved in the process.

This challenge adds additional work to the client, as the client needs to adapt the requirements to the audience. For example, sketches and drawings need to be done so the client can communicate the requirements with the property owners.

4.2.5. Challenge 7: Requirements abstraction

The requirements communicated by the client would likely have variations in abstraction levels. The challenging part is to find the right level of granularity for these requirements. This is reported in Case Two by the head of environment (C4). A too specific requirement could constrain the supplier in developing the solution, "sometimes we were getting the wrong answer when we are too specific in demands" (C4), and a too abstract requirement is open for miss interpretation "often we have to explain more to make the requirements more specific" (C4).

We did not identify clear causes or consequences for this challenge from the interviews. The interviewee did not have a concrete answer about what could be causing such a challenge and what could be the consequences.

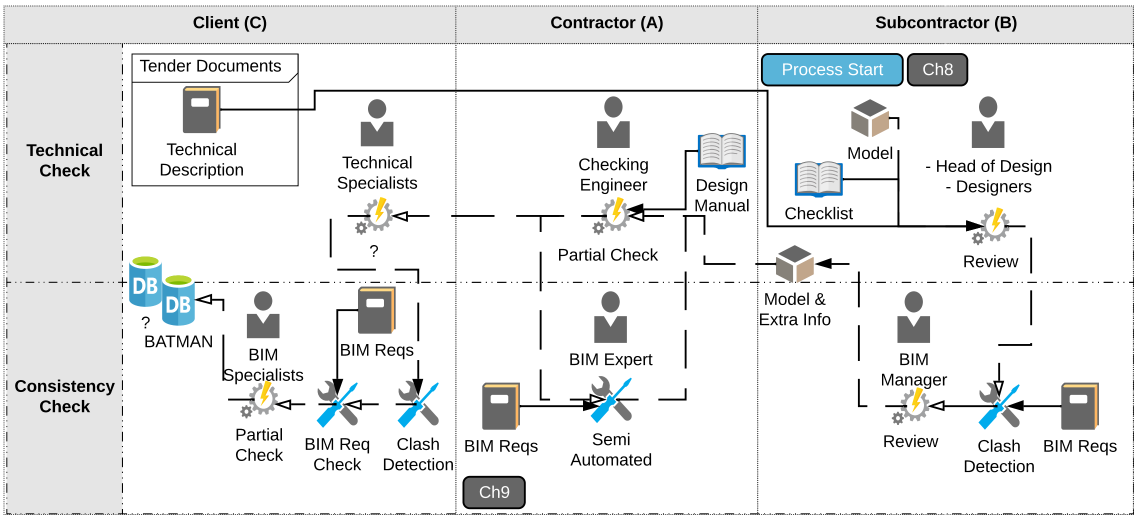


Figure 7. Digital assets verification – Case one

4.3. Digital assets verification

Figure 7 shows the digital assets verification process in Case One. There are two verification processes present in the figure. The technical check is the process of verifying the produced digital assets' conformity with the specified technical requirements. The consistency check is the process of verifying the conformity of digital assets with the BIM requirements. BIM requirements specifies how the design models should be delivered (e.g., the file's extension to be used for delivering a model, or the units of measure used).

As the subcontractor designers produce the model, a review process is conducted by those designers and the head of design. The process takes as input 1) the produced model, which could be 2D drawings/3D models/BIM, 2) a checklist that is prepared by the designers based on the technical requirements before the implementation, 3) the technical requirements of the project. The output model of this process, along with BIM requirements, goes into a clash detection tool, which checks whether the models have any conflict in design objects. After that, a manual review process is made by the BIM manager.

At the contractor, a similar verification process on the model is followed. First, a technical check is done with the design manual, which is produced at the beginning of the project. This check is done by a checking engineer and is mainly based on experience with verification of similar models. The model is sent to the BIM expert who conducts pre-configurated semi-automated checks on the attributes of the BIM objects based on the client's BIM requirements, using internal tools with customizable configuration possibilities. The technical and consistency check is an iterative process.

Then the model is sent to the client for approval. At the client, there are technical specialists who do their own technical checks while taking technical requirements as input. After that, a series of consistency checks are conducted: clash detection, BIM requirements check, and partial check by BIM specialists. When the model passes all those checks, it is stored in the appropriate database.

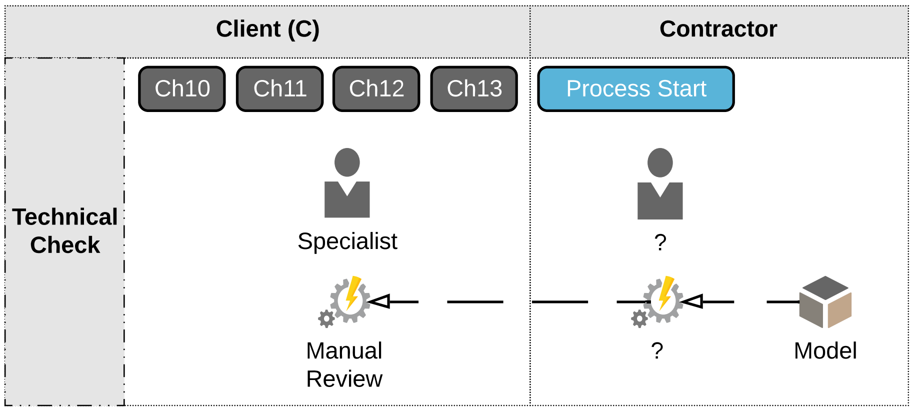


Figure 8. Digital assets verification – Case two

Figure 8 presents the digital assets verification process in Case Two. When the models are delivered to the client for approval, the specialist in the domain conducts a manual review process to verify those models. We also know that there is some kind of check being done by the contractor.

We have identified six challenges related to the process of digital assets verification.

4.3.1. Challenge 8: Requirements are impossible to build

In some cases, during the verification of digital assets, engineers detect requirements that are impossible to build. This challenge was reported in Case One by the discipline leader (B1). The contractor and subcontractor are limited by what they can change in the requirements. Then during the verification of digital assets, some requirements appear to be impossible to build. "The solution that company C stated simply can't be done" (B1). Although the client may be validating the requirements, it is still the contractor's and subcontractor's responsibility to make sure those requirements are sound and possible to build.

The client specifies solutions rather than requirements were the cause of this challenge. The requirements provided by the client do not have much room for the contractor or subcontractor to come up with a solution. Rather the requirements specify an actual solution that the contractor and subcontractor need to follow. Therefore, if those requirements/solutions have conflicts, they will likely show during the verification process of the digital assets (since they might be overlooked in the requirements validation process, as discussed in Section 4.1.1.

The consequence of this challenge is *extra effort spent on rework* to be done by the subcontractor. During the verification process, if some requirements were detected to be impossible to build, then the subcontractor needs to come up with a new solution, verify it, and request a change for requirements.

4.3.2. Challenge 9: Difficulty understanding BIM requirements

Some consultants working on the project are not fluent or familiar with the language in which the requirements documents are written, as reported in Case One by the BIM manager (A3). It is difficult for those consultants to understand the BIM requirements. Although the consultants translate those documents into their language, they are not confident that the translation is accurate.

Consultants are unfamiliar with the documents' language was identified as the cause of this issue. "The language is always a problem when you are an international consultant" (A3).

Long time spent at the beginning is the consequence of this challenge. At the beginning of the project, it took a while for both the client and the contractor to coordinate and make sure that they were on the same page; as mentioned by A3 "It took some time coordinating with the client to understand the requirements" (A3).

This challenge is not specified to BIM requirements only. The project has many people (requirements engineers and designers) with different language proficiency, whom may find it difficult to interpret the requirements if the translation is inaccurate.

4.3.3. Challenge 10: Requirements management tool related

In Case Two, one of the identified challenges is related to using the requirements management tool DOORS. This challenge was reported by the requirements specialist (C2). People would prefer to use conventional tools like spreadsheets over a specialized new system like DOORS; as mentioned by C2, "specialists don't really like new systems" (C2). It is challenging to get people working on the tool. Another part of this challenge is related to reaching models from within DOORS. "When specialists review stuff, they need to go to the specific document in the specific models to look, and they can't just click on the link, which is annoying" (C2). The delivered models are verified in a different system, and it could be difficult to find the model and the related requirements during the review process by the client's engineers.

The causes we identified for this challenge are *use of yet another new tool* and *the limitation of the tool DOORS*. The requirements management tool DOORS is new for many specialists working on the project, and they are not familiar with it; also, some specialists already maintain different systems. "It is a lot for people that already maintain like fifteen different systems so that was an issue to make them like DOORS" (C2). Also, DOORS has its limitations, e.g., "you can't do hyperlinks in the attribute in DOORS" (C2). Currently, using the tool to link the requirements to a specific attribute or part of the model is not possible.

This challenge leads to *unnecessary work for the requirements specialist* and *extra effort by the people verifying the models*. Since DOORS is a new system for people to use, the requirements specialist does spend time preparing views and arranging requirements to make it easy for people involved in the review process. "I have to be involved quite a bit just for them to know where to look" (C2). Additionally, since the tool has a limitation in linking requirements to the detailed model implementations, model verification requires extra effort. As explained by C2 "they need to be ready to open several different models which take time to load" (C2).

It is important to note here that this challenge does not point out the deficiencies of a specific tool, but rather the lack of integration of the many tools an engineer needs to use to perform their work.

4.3.4. Challenge 11: Granularity of traces

The traces created between the requirements and the models are of high abstraction. This was reported in Case Two by the requirements specialist (C2). There is information used in DOORS to trace requirements to the created models. However, this traceability information is not detailed enough, which makes it difficult to do the verification process. It is challenging to create those kinds of traces as it is seen to be an expensive practice.

Requirements are not linked to objects is seen to be as the cause of this challenge, "even though they give you a specific place to look there will be lots of places to look at" (C2). The current traces are created between requirements and models rather than requirements and the objects.

One consequence of this challenge is that an *extra effort is required to verify the model* as C2 explained "it just takes some time and effort" (C2). Since the requirements are linked to models rather than objects within the model, it takes extra effort and time from the specialist to verify the model.

4.3.5. Challenge 12: Lack of Experience Using Tools

There is a lack of experience in using the modeling tools by the people verifying the design models. This challenge was reported in Case Two by the requirements specialist (C2). The manual model review process, at the client in Case Two, depends mainly on the experience of the specialists. Although, DOORS contains traces between requirements and models, the specialists verifying the model must know where in the model the requirements apply. The specialists lack experience verifying those models, as (C2) put it "We are not very experienced using the model, so that's a challenge for all specialists" (C2).

The use of different models in the project is the cause of this challenge. There are too many model types used in the project; these models require different tools for viewing. It is difficult for specialists to cope with all these tools. C2 said, “we can not choose exactly what they are going to use; unfortunately, some consultants use different BIM modeling programs” (C2).

Difficulty navigating the models is a consequence of lacking the experience in verifying the models. Since the digital assets verification process at the client in Case Two is an experience-based process, it gets difficult to navigate the models to verify them. The specialist needs to figure out where in the model the specific requirements apply.

4.3.6. Challenge 13: Verifying all requirements

The client finds it challenging to verify all the requirements in the delivered model. This was reported in Case Two by the requirements specialist (C2). There are many generic and project-specific requirements that apply to the models; it becomes troublesome to verify whether the models conform to all requirements.

The *absence of risk analysis* for the requirements makes the verification of all requirements difficult. There exists no risk analysis nor classification of requirements based on severity or importance. Therefore, the client has no basis on which to base the prioritization for verification and resource allocation.

As a consequence, there is an *Uncertainty in the verification process* at the client side. Currently, it is not determined if a complete model verification, for all requirements that apply, is necessary or not. “There is a debate at the client whether or not we are supposed to do a complete verification or just sample verification and see” (C2).

5. Discussion

In this section, we discuss the implications of our study results for research and practitioners. We start by discussing the client-supplier communication cycle. Then we look at the challenges and identify potential solution areas from the literature. Finally, we discuss the role of the requirements documents in client-supplier communication.

5.1. Client-supplier communication

There are differences between the two cases in the way the requirements are communicated. In Case One, the client documented and released the requirements to the suppliers once, at the beginning of the project (Figure 5), while in Case Two, the requirements specification is a more mature process, as it adopts an iterative approach where the client and the contractor (i.e, supplier) negotiate the requirements before they are stored in a requirements management tool (Figure 6). Early requirements negotiation supports more correct and feasible requirements specification [47], and is part of the recipe of the system engineering best practices [48]. The requirements process in Case Two follows best practices and is, therefore, superior to the process in Case One.

The development of the requirements specifications in an infrastructure project is shared between the client and supplier, as the latter is expected to do their own analysis of the requirements, elicit the missing ones, and ensure that they align with the client’s expectations. This is in contrast to software engineering projects where the client is normally responsible for sharing a set of requirements that aligns with their needs. However, in the public domain, the development of the requirements specifications in software projects could be similar to infrastructure projects. In a case study conducted by Brataas et al., [49] to investigate an innovation partnership project in the Norwegian medical sector, the development of the requirements was a joint effort between the client procurement team and multiple suppliers, which allowed the procurement team to focus on innovation. Moe et al. [50] made a similar observation in their study on information systems requirements in public sector projects: the dialogue between the client and supplier complements the project's requirements specification.

In Case One, every party involved in the project does their verification for the delivered digital assets. The technical check, as we see in Figures 7 and 8, is a sample check done manually. There are two drawbacks of this check: 1) it is time consuming, and 2) it does not verify all parts of the delivered digital assets. The supplier takes responsibility for the issues that show later in the project due to unverified requirements. A similar observation was made by Makkinga et al. [19], where a supplier carried the responsibility of the verification when the contractor did not have enough resources to conduct a complete verification. Consistency checks, which verify whether the design conforms to the general design requirements such as correct use of units of measurement or file formats, are done (semi-)automatically, as shown in Figure 7. We believe that developing a common checklist would have been beneficial in the early identification of potential issues to requirements. Moreover, we speculate that communication between two parties with various levels of domain knowledge could be facilitated by communication brokers [51].

The life cycle model used in both cases is sequential. A sequential model is defined by INCOSE [52] as a systematic approach for the system development process where the system goes through a sequence of steps from goals definition to a complete system. This sequential model is beneficial to use in our cases which are large projects with different parties involved. However, this model also requires verified traceable requirements [52], which is challenging to achieve. As we saw in Section 4, the requirements validation may be overlooked (ch3), and the traceability information is not adequate (ch11), which makes checking all the requirements difficult (ch13).

5.2. Challenges and potential solutions

The mapping of challenges to solutions was done based on the following: (1) Analysis and synthesis of the results, as some possible solutions were identified during the interview process, e.g., feedback during RE process, policy change and tender process. (2) The authors' experience in requirements engineering research

Figure 9[3] depicts the identified challenges, their causes, and their consequences and shows the relations between them based on the analysis of the study results and complemented with references from the literature based on the author's experience in requirements engineering research. In addition, we propose solutions areas to tackle these challenges. In the problem analysis, arrows represent a *contributes to* relation between the connected elements. The arrows from the solution area to the problem area indicate whether a solution *potentially addresses* the connected cause and consequently the connected challenge(s). We differentiate between whether the relation originates from:

– our results (solid arrow)
– an observation made in another study (dashed arrow with reference)
– both from our results and from literature (solid arrow with reference)

[3]We will make the citations in the figure consistent with the citation in the paper before publishing.

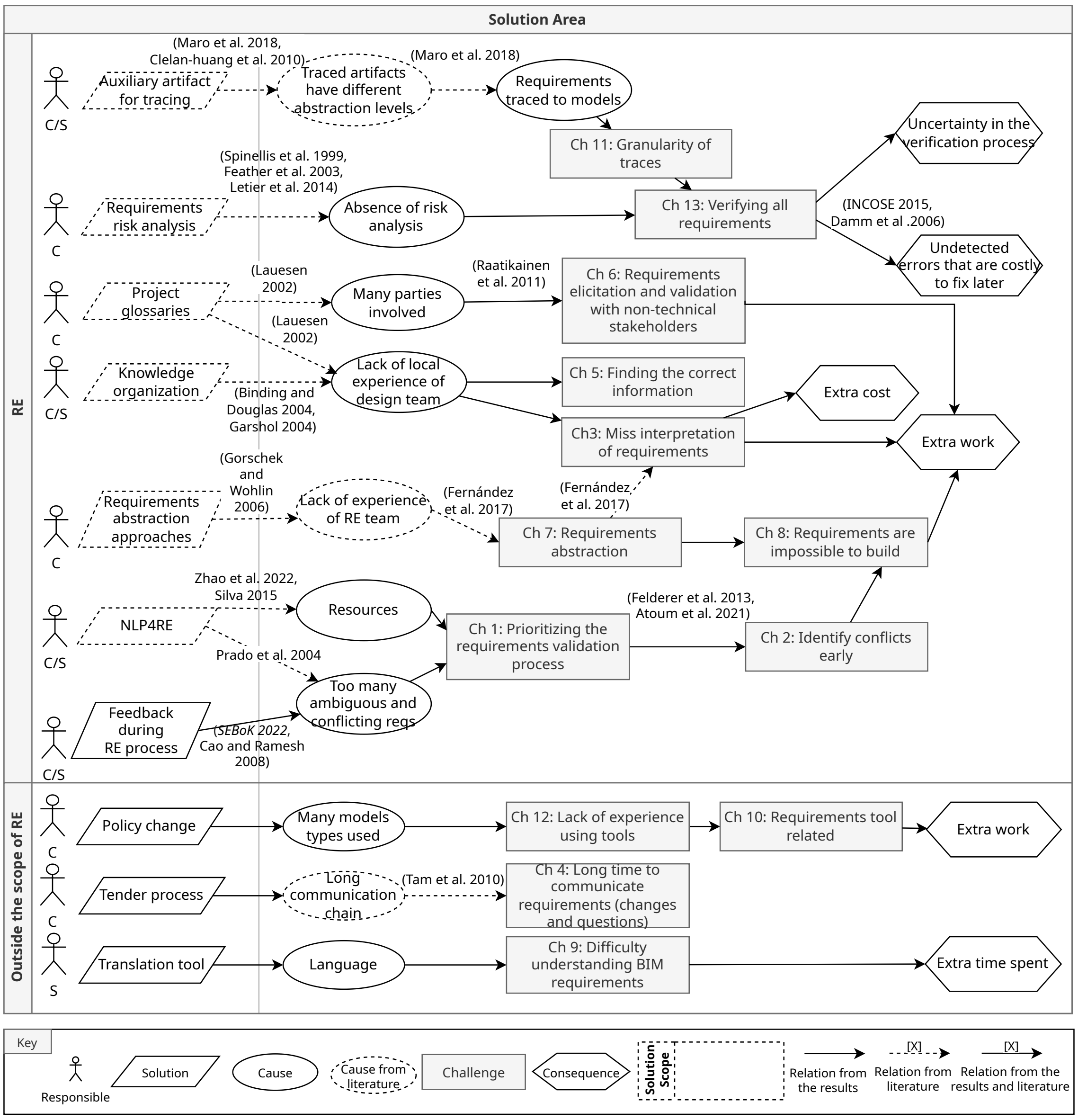

Solution Area
RE
Outside the scope of RE
C/S
Auxiliary artifact for tracing
(Maro et al. 2018, Clelan-huang et al. 2010)
Traced artifacts have different abstraction levels
(Maro et al. 2018)
Requirements traced to models
Ch 11: Granularity of traces
Uncertainty in the verification process
C
Requirements risk analysis
(Spinellis et al. 1999, Feather et al. 2003, Letier et al. 2014)
Absence of risk analysis
Ch 13: Verifying all requirements
(INCOSE 2015, Damm et al .2006)
Undetected errors that are costly to fix later
C
Project glossaries
(Lauesen 2002)
Many parties involved
(Raatikainen et al. 2011)
Ch 6: Requirements elicitation and validation with non-technical stakeholders
(Lauesen 2002)
C/S
Knowledge organization
(Binding and Douglas 2004, Garshol 2004)
Lack of local experience of design team
Ch 5: Finding the correct information
Extra cost
Ch3: Miss interpretation of requirements
Extra work
C
Requirements abstraction approaches
(Gorschek and Wohlin 2006)
Lack of experience of RE team
(Fernández et al. 2017)
(Fernández et al. 2017)
Ch 7: Requirements abstraction
Ch 8: Requirements are impossible to build
C/S
NLP4RE
Zhao et al. 2022, Silva 2015
Resources
Prado et al. 2004
Ch 1: Prioritizing the requirements validation process
(Felderer et al. 2013, Atoum et al. 2021)
Ch 2: Identify conflicts early
C/S
Feedback during RE process
(SEBoK 2022, Cao and Ramesh 2008)
Too many ambiguous and conflicting reqs
C
Policy change
Many models types used
Ch 12: Lack of experience using tools
Ch 10: Requirements tool related
Extra work
C
Tender process
Long communication chain
(Tam et al. 2010)
Ch 4: Long time to communicate requirements (changes and questions)
S
Translation tool
Language
Ch 9: Difficulty understanding BIM requirements
Extra time spent
Key
Responsible
Solution
Cause
Cause from literature
Challenge
Consequence
Solution Scope
Relation from the results
[X]
Relation from literature
[X]
Relation from the results and literature

Figure 9. Relationship between challenges, the abbreviation (ch) refers to challenge

The solution areas proposed in this section are not exhaustive for all possible solutions for the challenges and their causes. Rather, they are potential solutions where their effectiveness in addressing the challenges has been reported in literature but still needs to be verified in the particular context we have studied. Furthermore, we elaborate on where a solution could be implemented, at the client or the supplier side, by indicating the main *responsible party*. We present the solution areas in more detail next.

Auxiliary artifacts in tracing When tracing requirements to downstream artifacts in software (e.g., test cases) and system (e.g., design models) projects, an auxiliary artifact could be used to address the different abstraction levels of the traced artifacts. Thus, the auxiliary artifact should be traceable to artifacts with different abstraction levels. It can be an artifact that is produced during the project execution, e.g., a lower level requirement [57], or an external artifact that is not part of the software or system, e.g., a domain ontology containing domain concepts and has a hierarchical structure [58]. Using an auxiliary artifact solves the abstraction level mismatch between requirements and the downstream artifacts (e.g., test cases or design models), and consequently could address the granularity of traces challenge (ch11). Fine-grained trace links make it possible to trace requirements to their implementation; thus, verifying all requirements (ch13) in the design model can be more feasible. This solution should be implemented by the client (for the requirements) and by the supplier (for the delivered artifact).

Requirements risk analysis A risk analysis could be conducted on all requirements to assess the severity and consequences of failing to fulfill a requirement. A requirements-risk analysis is usually conducted in a similar way as those done for a project plan. In the requirements-risk analysis, three main concepts are identified: problematic requirements, the possible failures, and activities that could prevent or mitigate these risks [59]. The risk analysis results help engineers in decision-making by reducing uncertainty [60]. Such decisions are where resources should be spent [59] or which requirements should be prioritized for verification [61]. The client should be responsible for introducing requirements risk analysis as part of the RE process to ensure that the project adheres to the requirements that threaten the project success.

Project glossaries Project glossaries consist of a set of terms and their descriptions in a specific domain or a project. Project glossaries are used to mitigate misinterpretations of terms (ch3) by project members with different backgrounds [62]. Implementing project glossaries can address differences in experience of design teams align terminology between parties with different backgrounds involved during requirements elicitation (ch6).

Project glossaries address these issues by providing an explicit description of terms that can be misunderstood. Delisle and Olson investigate whether project-based terminology and definitions are actually as widely accepted as believed and conclude that more effort should be dedicated towards coordination of glossaries and dissemination of information about project management terms and definitions [63]. We speculate that a broader adoption of glossaries can be supported by automated construction of project glossaries [64, 65]. In our case, the client specifies the project requirements, so it is his responsibility to develop and maintain the project glossary.

Knowledge organization Knowledge organization is the process of indexing, classifying, and archiving documents and books to ease access to them [66]. McClory et al. stressed the importance of knowledge management and organisational learning and suggested triple-loop learning as an organisational structure [67].

Infrastructure projects contain many documents (e.g., rules and regulations) that need to be structured and organized to be useful for requirements extraction and analysis. Garshol [68] identified different types of knowledge organization systems, e.g., controlled vocabularies and taxonomies. The use of an appropriate knowledge organization system to structure domain knowledge [69] could allow engineers to find information about local projects more efficiently and effectively (ch5). This solution could be introduced either by the client or the supplier. The client owns the rules and regulations, and organizing them through automated classification makes access to information more efficient. The supplier could be obligated to adhere to international standards which apply to multiple projects.

Requirements abstraction methods First, one of the causes of the major challenges in requirements engineering is the existence of too abstract requirements, which could result in miss interpretation (ch3) [70]. Thus, a requirement should not be too abstract or unambiguous [71]. Second, according to SWEBOK, a requirement is defined as “a property that must be exhibited by something in order to solve some problem in the real world" and a good SRS should be an agreement between the client and supplier about "what the software product is to do", not how to do it [71]. Thus, a requirement should not restrict possible solutions or be impossible to be realized (ch8). We argue that requirement abstraction methods should be applied to ensure requirements are written on an appropriate abstraction level. Gorschek and Wohlin [72] have developed the Requirements Abstraction Model (RAM), a method to systematically specify requirements on multiple levels of abstraction (from product to component level). RAM has been shown to improve the requirements engineering process and the quality of the requirements specified in practice [73]. Liebel et al. suggested coordinating requirements on various abstraction to avoid communication and coordination breakpoints [30]. Requirements engineers on the client side should adopt one of these solutions as part of the RE process.

NLP4RE Natural language processing (NLP) is the use of (semi-)automated techniques to analyze and model human language [74], mainly through employing machine learning. This is particularly beneficial as the majority of requirements in software and system projects are specified in natural language [75, 76]. NLP approaches have shown their potential to support the requirements validation process as presented by Zhao et al. [77] in their review of the literature on NLP for requirements engineering (NLP4RE). The use of NLP4RE reduces the number of resources required for the requirements validation [77, 78], e.g., by performing automated model checking. Moreover, NLP4RE supports requirements engineers in identifying conflicts and ambiguities (ch2), which requires expertise [79]. Both the client and supplier could benefit from adopting this solution in the RE process.

Feedback during RE process One of the good practices in system engineering [48] and software engineering [80] during the requirements engineering process is adopting an iterative approach, where the client and supplier agree on the requirements to be developed incrementally. This approach increases the understanding of requirements and helps with their prioritization [81]. In Case Two, an iterative approach for RE is adopted where the client and the contractor (i.e., supplier) negotiate the requirements before they are stored in a requirements management tool (Figure 6). However, in Case One, the requirements were specified and communicated up-front, and changes were difficult to introduce due to contractual obligations. Adopting an iterative approach when specifying the requirements where the feedback of the supplier is considered reduces ambiguity and conflict in the specified requirements. Both the client and the supplier are responsible for implementing this solution.

Outside the scope of RE Other challenges (4,9,10,12) and solutions areas are outside the scope of RE but in the area of project management. A change in companies' policy (client), an improved tender process (client), and using better tools are required in order to tackle these challenges.

5.3. Impact on academia and industry

This study illustrates the persistence of challenges related to requirements. Since existing solutions may not be effective, a root-cause analysis should be conducted to identify the causes behind these challenges and find solutions to address them. Furthermore, the requirements process and identified challenges in infrastructure projects are similar to those in the software engineering domain. Thus, although they are two distinct domains, they share many similarities when it comes to requirement engineering. The software and system engineering domain researchers should learn from each other's experiences in investigating requirements engineering processes and addressing challenges related to them.

When researchers and practitioners are looking for a solution to specific challenges in one domain, they should explore existing solutions in the other. The challenges that we identified in this study and their mapping to potential solutions, as we presented in this section, illustrate the similarity between the system and software engineering domain. Both domains use natural language to specify requirements, and both have digital assets that need to be verified against the specified requirements. In the software domain, engineers produce digital artifacts such as design documents and source code, while in the system domain, engineers produce design models (digital twins of the system) before building the physical build. Moreover, adopting requirements communication techniques is highly important for the system domain in general and the infrastructure domain in particular, due to the sub-contracted work being the norm, which adds more complexity to the requirements communication process.

5.4. Threats to validity

We use the framework by Runesson et al. [46] to discuss the validity threats of our study. The framework lists four categories of case study validity: construct, internal, external, and reliability. We analyzed the validity threats of our study from the beginning of our study, and we continued to revise these threats to minimize them.

Construct validity We mitigated threats to design and execution by involving more than one researcher in designing and conducting the study. The involved researchers reviewed the research questions and the case study protocol. Feedback sessions were conducted to discuss the protocol and make improvements. For example, the first author wrote a set of interview questions per studied process. During a feedback session, the third author suggested that the questions should be adapted to each interviewee's role. On that premise, the interview questions were adapted to each role by the first author, and the other researchers reviewed them. The improvement of the interview questions for each interview was done by the researchers before the start of said interview.

Internal validity Although the involved researchers revised and improved the case study protocol, there might be a bias or errors in the collected data. To reduce the risk of bias in data collection, between two and three researchers were present during each interview. One researcher led the interview, and the other researchers observed the interview and asked follow-up questions if necessary. In addition, after an interview was completed, we transcribed the interview, a second researcher verified the transcript and finally sent it out to the interviewee for comments.

Another internal threat to validity is the risk of bias when coding the transcripts as part of the data analysis phase. We mitigate this threat by assigning the coding task to one researcher and the verification of those codes to another. Eventual disagreements were resolved in meetings.

External validity It could be argued that generalization would be difficult with two case studies and that more case studies may be required to increase the validity of the study and achieve generalization. However, generalization can be achieved by individual cases. It is a matter of the selected cases. As argued by Flyvbjerg, one of the five misunderstandings of a case study is the inability to generalize from a single case [82]. We focus on analytical generalization rather than statistical generalization by providing detailed case descriptions and discussing the implications of our findings. Moreover, we believe that our cases are a good representation of the population (infrastructure projects). They are two large infrastructure projects, with large-sized (45 000+ employees) companies involved. Case One is in its final stages, while Case Two is in its early stages.

Reliability To ensure that our study is repeatable, we present in Section 3 the protocol of our study. We explain the setup of our studied cases, list the cluster of the interview questions, present the steps in which we conducted the interviews, and the coding mechanism of the interview data.

6. Conclusion

We have conducted two case studies in three companies to explore the requirements validation, requirements communication, digital assets verification processes, and the challenges associated with these processes between the client and the supplier in infrastructure projects. We identified 13 challenges and proposed potential solutions from the literature to address these challenges. Many of the challenges faced, during the forward communication (tender documents) and backward communication (project deliverables), between the client and supplier can be addressed in the area of requirements engineering. Furthermore, the solution for similar challenges in the software domain can potentially address the challenges observed in the system domain.

The system requirements play a main role in the communication between the client and supplier in infrastructure projects, and their quality likely affects subsequent processes, e.g., verification and acceptance. The verification process is a difficult task in projects on this large scale, with thousands of requirements and models to validate. One particular difficulty is keeping track of the requirements during the verification process.

7. Future work

We have identified several future work directions based on the finding from this paper, that we outline below:

- Exploring methods to introduce early requirements risk analysis and how to estimate potential risks on early and often incomplete requirements and how to enable risk-based requirements reasoning [59].
- Exploring auxiliary artifacts and their usage scenario in requirements traceability. This involves exploring the relationship between the structural characteristics of auxiliary artifacts (e.g., a taxonomy) and the performance of machine learning models for requirements traceability.
- Investigating what types of knowledge organization systems are the most suitable for large-scale infrastructure projects – we believe that project glossaries should be considered as an efficient way of organizing knowledge and building common vocabulary between the parties. We also plan to explore the suitability of topic maps knowledge organizational structure for infrastructure projects [68].
- Investigating the suitability of requirements abstraction models from the software domain (e.g., RAM [73]) for the infrastructure projects.
- Exploring the use of NLP4RE techniques to detect conflicts and ambiguity in functional requirements from the infrastructure domain.

CRediT authorship contribution statement

Waleed Abdeen: conceptualization, methodology, formal analysis, investigation, and writing – original draft. Krzysztof Wnuk: conceptualization, methodology, investigation, supervision, and writing – review and editing. Michael Unterkalmsteiner: conceptualization, methodology, formal analysis and investigation, supervision, project administration, funding acquisition, and writing – review and editing. Alexandros Chirtoglou: conceptualization, formal analysis and writing – review and editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Funding

This research was performed within DCAT Project, funded by Trafikverket, Sweden.

References

[1] H. Hofmann and F. Lehner, "Requirements engineering as a success factor in software projects," *IEEE Software*, Vol. 18, No. 4, Jul. 2001, pp. 58–66.

[2] E. Bjarnason, K. Wnuk, and B. Regnell, "Requirements are slipping through the gaps – A case study on causes and effects of communication gaps in large-scale software development," in *IEEE 19th international requirements engineering conference*. IEEE, 2011, pp. 37–46.

[3] O. Liskin, "How artifacts support and impede requirements communication," in *Requirements Engineering: Foundation for Software Quality: 21st International Working Conference*. Springer, 2015, pp. 132–147.

[4] P. Brereton, "The software customer/supplier relationship," *Commun. ACM*, Vol. 47, No. 2, Feb. 2004, p. 77–81.

[5] C. Ebert, "Optimizing supplier management in global software engineering," in *International Conference on Global Software Engineering (ICGSE)*, 2007, pp. 177–185.

[6] T. Vullinghs, T. Gantner, S. Steinhauer, and T. Weber, "Experiences on lean techniques to manage software suppliers," in *Product Focused Software Process Improvement: Second International Conference, PROFES 2000, Oulu, Finland, June 20-22, 2000. Proceedings 2*. Springer, 2000, pp. 244–256.

[7] Y. Zheng, S. Liu, W. Huang, and J.J.Y. Jiang, "Inter-organizational cooperation in automotive new product development projects," *Industrial Management and Data Systems*, Vol. 120, No. 1, 2020, pp. 79–97.

[8] J.J. Jiang, G. Klein, J.C.A. Tsai, and Y. Li, "Managing multiple-supplier project teams in new software development," *International Journal of Project Management*, Vol. 36, No. 7, 2018, pp. 925–939. [Online]. https://www.sciencedirect.com/science/article/pii/S026378631830125X

[9] A. Butt, M. Naaranoja, and J. Savolainen, "Project change stakeholder communication," *International Journal of Project Management*, Vol. 34, No. 8, 2016, pp. 1579–1595.

[10] G. Wu, C. Liu, X. Zhao, and J. Zuo, "Investigating the relationship between communication-conflict interaction and project success among construction project teams," *International Journal of Project Management*, Vol. 35, No. 8, 2017, pp. 1466–1482.

[11] Y.F. Badir, B. Büchel, and C.L. Tucci, "A conceptual framework of the impact of NPD project team and leader empowerment on communication and performance: An alliance case context," *International Journal of Project Management*, Vol. 30, No. 8, 2012, pp. 914–926.

[12] H.Ç. Demirel, W. Leendertse, L. Volker, and M. Hertogh, "Flexibility in PPP contracts – Dealing with potential change in the pre-contract phase of a construction project," *Construction management and economics*, Vol. 35, No. 4, 2017, pp. 196–206.

[13] A.D. Songer and K.R. Molenaar, "Project characteristics for successful public-sector design-build," *Journal of construction engineering and management*, Vol. 123, No. 1, 1997, pp. 34–40.

[14] C.W. Ibbs, Y.H. Kwak, T. Ng, and A.M. Odabasi, "Project delivery systems and project change: Quantitative analysis," *Journal of construction engineering and management*, Vol. 129, No. 4, 2003, pp. 382–387.

[15] E. Kania, E. Radziszewska-Zielina, and G. Śladowski, "Communication and information flow in Polish construction projects," *Sustainability*, Vol. 12, No. 21, 2020, p. 9182.

[16] Z.Y. Zhao, Q.L. Lv, J. Zuo, and G. Zillante, "Prediction system for change management in construction project," *Journal of Construction Engineering and Management*, Vol. 136, No. 6, 2010, pp. 659–669.

[17] S. Malik, M. Taqi, J.M. Martins, M.N. Mata, J.M. Pereira et al., "Exploring the relationship between communication and success of construction projects: The mediating role of conflict," *Sustainability*, Vol. 13, No. 8, 2021, p. 4513.

[18] S.D. Vermillion and R.J. Malak, "An investigation on requirement and objective allocation strategies using a principal-agent model," *Systems Engineering*, Vol. 23, No. 1, 2020, pp. 100–117.

[19] R. Makkinga, R.d. Graaf, and H. Voordijk, "Successful verification of subcontracted work in the construction industry," *Systems Engineering*, Vol. 21, No. 2, 2018, pp. 131–140.

[20] A.M. Madni and M. Sievers, "Systems integration: Key perspectives, experiences, and challenges," *Systems Engineering*, Vol. 17, No. 1, 2014, pp. 37–51.

[21] J. Xu, H. Smyth, and V. Zerjav, "Towards the dynamics of trust in the relationship between project-based firms and suppliers," *International Journal of Project Management*, Vol. 39, No. 1, 2021, pp. 32–44.

[22] L.S. Henderson, "Encoding and decoding communication competencies in project management – An exploratory study," *International Journal of Project Management*, Vol. 22, No. 6, 2004, pp. 469–476.

[23] M. Loosemore and H. Muslmani, "Construction project management in the Persian Gulf: Inter-cultural communication," *International Journal of Project Management*, Vol. 17, No. 2, 1999, pp. 95–100.

[24] C. Semeraro, M. Lezoche, H. Panetto, and M. Dassisti, "Digital twin paradigm: A systematic literature review," *Computers in Industry*, Vol. 130, 2021, p. 103469.

[25] A. Toygar, "A new asset type: Digital assets," *Journal of International Technology and Information Management*, Vol. 22, No. 4, 2013, p. 9.

[26] P.E.D. Love, J. Zhou, J. Matthews, and H. Luo, "Systems information modelling: Enabling digital asset management," *Advances in Engineering Software*, Vol. 102, 2016, pp. 155–165.

[27] A.R. Hevner, S.T. March, J. Park, and S. Ram, "Design science in information systems research," *MIS quarterly*, 2004, pp. 75–105.

[28] R.J. Wieringa, *Design science methodology for information systems and software engineering*. Springer, 2014.

[29] E. Bjarnason, B. Gislason Bern, and L. Svedberg, "Inter-team communication in large-scale co-located software engineering: A case study," *Empirical Software Engineering*, Vol. 27, No. 2, 2022, p. 36.

[30] G. Liebel, M. Tichy, E. Knauss, O. Ljungkrantz, and G. Stieglbauer, "Organisation and communication problems in automotive requirements engineering," *Requirements Engineering*, Vol. 23, 2018, pp. 145–167.

[31] J. Iqbal, R. Ahmad, M.H.N.B.M. Nasir, and M.A. Noor, "A framework to address communication issues during requirements engineering process for software development outsourcing," *Journal of Internet Technology*, Vol. 19, No. 3, 2018, pp. 845–859.

[32] S. Fricker, T. Gorschek, C. Byman, and A. Schmidle, “Handshaking: Negotiate to provoke the right understanding of requirements,” *IEEE Software*, Vol. 99, 2009.

[33] J. Pernstål, T. Gorschek, R. Feldt, and D. Florén, “Requirements communication and balancing in large-scale software-intensive product development,” *Information and Software Technology*, Vol. 67, 2015, pp. 44–64.

[34] S. Maalem and N. Zarour, “Challenge of validation in requirements engineering,” *Journal of Innovation in Digital Ecosystems*, Vol. 3, No. 1, 2016, pp. 15–21.

[35] T.F. Lynghaug, S. Kokkula, and G. Muller, “Investigating systems engineering approaches in the norwegian construction industry: A multi-case study,” *INCOSE International Symposium*, Vol. 32, No. 1, 2022, pp. 792–808.

[36] R. de Graaf, R. Pater, and H. Voordijk, “Level of sub-contracting design responsibilities in design and construct civil engineering bridge projects,” *Frontiers in Engineering and Built Environment*, Vol. 3, No. 3, 2023, pp. 192–205, publisher: Emerald Publishing Limited.

[37] M. Raatikainen, T. Männistö, T. Tommila, and J. Valkonen, “Challenges of requirements engineering – A case study in nuclear energy domain,” in *IEEE 19th International Requirements Engineering Conference*. IEEE, 2011, pp. 253–258.

[38] D. Saxena and J. McDonagh, “Communication breakdowns during business process change projects – Insights from a sociotechnical case study,” *International Journal of Project Management*, Vol. 40, No. 3, 2022, pp. 181–191.

[39] E. Ochieng and A. Price, “Managing cross-cultural communication in multicultural construction project teams: The case of Kenya and UK,” *International Journal of Project Management*, Vol. 28, No. 5, 2010, pp. 449–460.

[40] T.U. Daim, A. Ha, S. Reutiman, B. Hughes, U. Pathak et al., “Exploring the communication breakdown in global virtual teams,” *International Journal of Project Management*, Vol. 30, No. 2, 2012, pp. 199–212.

[41] L.S. Henderson, R.W. Stackman, and R. Lindekilde, “The centrality of communication norm alignment, role clarity, and trust in global project teams,” *International Journal of Project Management*, Vol. 34, No. 8, 2016, pp. 1717–1730.

[42] A. Terry Bahill and S.J. Henderson, “Requirements development, verification, and validation exhibited in famous failures,” *Systems engineering*, Vol. 8, No. 1, 2005, pp. 1–14.

[43] E. Bjarnason, P. Runeson, M. Borg, M. Unterkalmsteiner, E. Engström et al., “Challenges and practices in aligning requirements with verification and validation: a case study of six companies,” *Empirical software engineering*, Vol. 19, No. 6, 2014, pp. 1809–1855.

[44] S. Hotomski, E.B. Charrada, and M. Glinz, “An exploratory study on handling requirements and acceptance test documentation in industry,” in *IEEE 24th International Requirements Engineering Conference (RE)*, 2016, pp. 116–125, ISSN: 2332-6441.

[45] A.R. Hevner, “A three cycle view of design science research,” *Scandinavian journal of information systems*, Vol. 19, No. 2, 2007, p. 4.

[46] P. Runeson and M. Höst, “Guidelines for conducting and reporting case study research in software engineering,” *Empirical software engineering*, Vol. 14, No. 2, 2009, p. 131.

[47] P. Grünbacher and N. Seyff, “Requirements Negotiation,” in *Engineering and Managing Software Requirements*, A. Aurum and C. Wohlin, Eds. Berlin, Heidelberg: Springer, 2005, pp. 143–162.

[48] SEBoK, “Stakeholder needs and requirements – SEBoK,” 2020, [Online; accessed 15-February-2021].

[49] G. Brataas, G.K. Hanssen, X. Qiu, and L.S. Græslie, “Requirements engineering in the market dialogue phase of public procurement: A case study of an innovation partnership for medical technology,” in *Requirements Engineering: Foundation for Software Quality*, V. Gervasi and A. Vogelsang, Eds. Cham: Springer International Publishing, 2022, pp. 159–174.

[50] C.E. Moe, M. Newman, and M.K. Sein, “The public procurement of information systems: dialectics in requirements specification,” *European Journal of Information Systems*, Vol. 26, No. 2, 2017, pp. 143–163.

[51] D. Damian, R. Helms, I. Kwan, S. Marczak, and B. Koelewijn, "The role of domain knowledge and cross-functional communication in socio-technical coordination," in *35th International Conference on Software Engineering (ICSE)*. IEEE, 2013, pp. 442–451.

[52] D.D. Walden and INCOSE, Eds., *INCOSE Systems Engineering Handbook: A Guide for System Life Cycle Processes and Activities*. New York: John Wiley and Sons, Incorporated, 2015.

[53] L.O. Damm, L. Lundberg, and C. Wohlin, "Faults-slip-through—a concept for measuring the efficiency of the test process," *Software Process: Improvement and Practice*, Vol. 11, No. 1, 2006, pp. 47–59.

[54] I. Atoum, M.K. Baklizi, I. Alsmadi, A.A. Otoom, T. Alhersh et al., "Challenges of software requirements quality assurance and validation: A systematic literature review," *IEEE Access*, Vol. 9, 2021, pp. 137 613–137 634.

[55] M. Felderer and A. Beer, "Using defect taxonomies for requirements validation in industrial projects," in *2013 21st IEEE International Requirements Engineering Conference (RE)*. IEEE, 2013, pp. 296–301.

[56] V.W.Y. Tam, L.Y. Shen, and J.S.Y. Kong, "Impacts of multi-layer chain subcontracting on project management performance," *International Journal of Project Management*, Vol. 29, No. 1, 2011, pp. 108–116.

[57] S. Maro, J.P. Steghöfer, and M. Staron, "Software traceability in the automotive domain: Challenges and solutions," *Journal of Systems and Software*, Vol. 141, 2018, pp. 85–110.

[58] J. Cleland-Huang, A. Czauderna, M. Gibiec, and J. Emenecker, "A machine learning approach for tracing regulatory codes to product specific requirements," in *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering – Volume 1*, ICSE '10. Association for Computing Machinery, 2010, pp. 155–164.

[59] M.S. Feather and S.L. Cornford, "Quantitative risk-based requirements reasoning," *Requirements Engineering*, Vol. 8, No. 4, 2003, pp. 248–265.

[60] E. Letier, D. Stefan, and E.T. Barr, "Uncertainty, risk, and information value in software requirements and architecture," in *Proceedings of the 36th International Conference on Software Engineering*, ICSE 2014. Association for Computing Machinery, 2014, pp. 883–894.

[61] D. Spinellis, S. Kokolakis, and S. Gritzalis, "Security requirements, risks and recommendations for small enterprise and home-office environments," *Information Management and Computer Security*, Vol. 7, No. 3, 1999, pp. 121–128, publisher: MCB UP Ltd.

[62] S. Lauesen, *Software Requirements: Styles and Techniques*. Pearson Education, 2002.

[63] C.L. Delisle and D. Olson, "Would the real project management language please stand up?" *International Journal of Project Management*, Vol. 22, No. 4, 2004, pp. 327–337.

[64] C. Arora, M. Sabetzadeh, L. Briand, and F. Zimmer, "Automated extraction and clustering of requirements glossary terms," *IEEE Transactions on Software Engineering*, Vol. 43, No. 10, 2017, pp. 918–945.

[65] C. Wang, X. Peng, M. Liu, Z. Xing, X. Bai et al., "A learning-based approach for automatic construction of domain glossary from source code and documentation," in *Proceedings of the 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ESEC/FSE 2019. Association for Computing Machinery, 2019, pp. 97–108.

[66] B. Hjørland, "What is knowledge organization (KO)?" *KO Knowledge Organization*, Vol. 35, No. 2-3, 2008, pp. 86–101.

[67] S. McClory, M. Read, and A. Labib, "Conceptualising the lessons-learned process in project management: Towards a triple-loop learning framework," *International Journal of Project Management*, Vol. 35, No. 7, 2017, pp. 1322–1335, social Responsibilities for the Management of Megaprojects.

[68] L.M. Garshol, "Metadata? Thesauri? Taxonomies? Topic maps! Making sense of it all," *Journal of Information Science*, Vol. 30, No. 4, 2004, pp. 378–391, publisher: SAGE Publications Ltd.

[69] C. Binding and D. Tudhope, "KOS at your service: Programmatic access to knowledge organisation systems," *Journal of Digital Information*, 2004.

[70] D.M. Fernández, S. Wagner, M. Kalinowski, M. Felderer, P. Mafra et al., “Naming the pain in requirements engineering: Contemporary problems, causes, and effects in practice,” *Empirical software engineering*, Vol. 22, 2017, pp. 2298–2338.

[71] *IEEE Recommended Practice for Software Requirements Specifications*, IEEE Std. 830-1998, 10 1998.

[72] T. Gorschek and C. Wohlin, “Requirements abstraction model,” *Requirements Engineering*, Vol. 11, No. 1, 2006, pp. 79–101.

[73] T. Gorschek, P. Garre, S.B.M. Larsson, and C. Wohlin, “Industry evaluation of the requirements abstraction model,” *Requirements Engineering*, Vol. 12, No. 3, 2007, pp. 163–190.

[74] J. Hirschberg and C.D. Manning, “Advances in natural language processing,” *Science*, Vol. 349, No. 6245, 2015, pp. 261–266, publisher: American Association for the Advancement of Science.

[75] M. Kassab, C. Neill, and P. Laplante, “State of practice in requirements engineering: Contemporary data,” *Innovations in Systems and Software Engineering*, Vol. 10, No. 4, 2014, pp. 235–241.

[76] S. Wagner, D.M. Fernández, M. Felderer, A. Vetrò, M. Kalinowski et al., “Status quo in requirements engineering: A theory and a global family of surveys,” *ACM Transactions on Software Engineering and Methodology (TOSEM)*, Vol. 28, No. 2, 2019, pp. 1–48.

[77] L. Zhao, W. Alhoshan, A. Ferrari, K.J. Letsholo, M.A. Ajagbe et al., “Natural language processing for requirements engineering: A systematic mapping study,” *ACM Computing Surveys*, Vol. 54, No. 3, 2022, pp. 1–41.

[78] A.R. da Silva, “SpecQua: Towards a framework for requirements specifications with increased quality,” in *Enterprise Information Systems*, Lecture Notes in Business Information Processing, J. Cordeiro, S. Hammoudi, L. Maciaszek, O. Camp, and J. Filipe, Eds. Springer International Publishing, 2015, pp. 265–281.

[79] F.A.C. Pinheiro, *Perspectives on Software Requirements*. Boston, MA: Springer US, 2004, Ch. Requirements Traceability, pp. 91–113.

[80] B. Ramesh, L. Cao, and R. Baskerville, “Agile requirements engineering practices and challenges: An empirical study,” *Information Systems Journal*, Vol. 20, No. 5, 2010, pp. 449–480.

[81] L. Cao and B. Ramesh, “Agile requirements engineering practices: An empirical study,” *IEEE Software*, Vol. 25, No. 1, 2008, pp. 60–67.

[82] B. Flyvbjerg, “Five misunderstandings about case-study research,” *Qualitative Inquiry*, Vol. 12, No. 2, 2006, pp. 219–245, publisher: SAGE Publications Inc.

Authors and affiliations

Waleed Abdeen
e-mail: waleed.abdeen@bth.se
ORCID: https://orcid.org/0000-0001-8142-9631
Software Engineering, BTH, Sweden

Krzysztof Wnuk
e-mail: krzysztof.wnuk@bth.se
ORCID: https://orcid.org/0000-0003-3567-9300
Software Engineering, BTH, Sweden

Michael Unterkalmsteiner
e-mail: michael.unterkalmsteiner@bth.se
ORCID: https://orcid.org/0000-0003-4118-0952
Software Engineering, BTH, Sweden

Alexandros Chirtoglou
e-mail: alexandros.chirtoglou@hochtief.de
HOCHTIEF ViCon GmbH, Germany

Aims and Scope

The purpose of **e-Informatica Software Engineering Journal** is to publish original and significant results in all areas of software engineering research.

Our focus is on empirical software engineering, as well as data science (in particular, artificial intelligence and machine learning methods) in software engineering.

e-Informatica Software Engineering Journal is published online. The online version is from the beginning published as a gratis, no authorship fees, open-access journal, which means that it is available at no charge to the public.

Topics of interest

- AI/ML/LLMs for software engineering
- Software engineering for AI
- Software requirements engineering and modeling
- Software architectures and design
- Software components and reuse
- Software testing, analysis and verification
- Agile software development methodologies and practices
- Model driven development
- Software quality
- Software measurement and metrics
- Reverse engineering and software maintenance
- Empirical and experimental studies in software engineering (incl. replications)
- Evidence-based software engineering
- Systematic reviews and mapping studies (see SEGRESS guidelines)
- Statistical analyses and meta-analyses of experiments
- Robust statistical methods
- Reproducible research in software engineering
- Object-oriented software development
- Aspect-oriented software development
- Software tools, containers, frameworks and development environments
- Formal methods in software engineering.
- Internet software systems development
- Dependability of software systems
- Human-computer interaction
- AI and knowledge based software engineering
- Data science in software engineering
- Prediction models in software engineering
- Mining software repositories
- Search-based software engineering
- Multiobjective evolutionary algorithms
- Tools for software researchers or practitioners
- Project management
- Software products and process improvement and measurement programs
- Process maturity models

Funding acknowledgements: Authors are requested to identify who provided financial support for the conduct of the research and/or preparation of the article and to briefly describe the role of the sponsor(s), if any, in study design; in the collection, analysis, and interpretation of data; in the writing of the paper. If the funding source(s) had no such involvement, then this should be stated as well.

Detailed paper requirements can be found on the Paper Requirements and Recommendations web page.

The submissions will be accepted for publication on the basis of positive reviews done by the international Editorial Board and reviewers.

English is the only accepted language of publication.

To submit an article, please use the Manuscript Central journal submission web page.

Subsequent issues of the journal will appear continuously according to the reviewed and accepted submissions.

The journal is included in the ISI Web of Science, Scopus, DBLP, Google Scholar, DOAJ, BazTech etc.

http://www.e-informatyka.pl/

e-Informatica

ISSN 1897-7979