scientific reports

OPEN

# Topic modeling-based prediction of software defects and root cause using BERTopic, and multioutput classifier

Devi Priya Gottumukkala[1]✉, Prasad Reddy P.V.G.D[2] & S. Krishna Rao[3]

**The occurrence of software defects remains a major obstacle in software engineering, resulting in costly debugging and maintenance efforts. This study introduces a new angle for software defect prediction (SDP), utilizing advanced natural language processing (NLP) and machine learning (ML) techniques. In this work, the proposed methodology, BERT-MOC, combines the power of BERTopic, a transformer-based topic modeling technique, with a multioutput classifier to predict software defects and the root cause (reason) of defects. BERTopic is used to extract the root cause of the defect from textual descriptions of software defects, creating a meaningful representation of the software artifacts. These topic representations are then combined with the defect log data set.A multi-output classifier is trained on the combined dataset to predict multiple outputs, i.e., defect/not defect and defect root cause, simultaneously. As an estimator, Logistic Regression, Decision Tree Classifier, Kneighbor Classifier, Random Forest Classifier, and Ensemble Method-Voting are included in the MultiOutput Classifier. The proposed model is evaluated by the metrics hamming loss, accuracy, F1-score, precision, recall, and Jaccard similarity. The multi-output classifier with ensemble method voting as an estimator achieved the highest performance with 97% accuracy and F1-score to predict the root cause of the defect and 94% accuracy and F1-score to predict defect or not.**

**Keywords** Software defect prediction (SDP), BERTopic, Multioutput classifier

Software development is a highly dynamic and complex activity in which detecting defects affects the behavior of software systems. This is a core element of software quality assurance and improvement since it helps in predicting defects in software[1]. Predicting and preventing software errors before they occur is very important. The conventional principles for SDP depend on metrics like complexity, churn in the code, and historical defect statistics[2]. While these metrics have been effective in determining the health of software projects, many focus on a direct layer that will not be able to understand deeper and more relationships within the codebase. As a result, researchers have been to explore more advanced techniques like NLP for analyzing textual information present in software artifacts such as comments, commit messages, and documentation[3]. One such innovation is the use of topic modeling, which, as a fragment of NLP, will be an advantageous method regarding defect log data analysis[4].

## Background

Data preprocessing, text data vectorization, and feature selection are essential steps when working with software defect logs for analysis or prediction. Since defect logs often contain text descriptions, text cleaning, tokenization, stop word removal, and stemming or lemmatization preprocess the text[5]. Processing natural language data involves several basic tasks, such as the vectorizing text data process, in which textual words are converted to numbers, which can be used for numerous NLP requirements like text classification, sentiment analysis, information retrieval, etc. Any ML algorithm that takes only numerical inputs for which vectorization is needed enables the algorithm to work on text data[6].

To this end, ML, deep learning, and transformer methods have become prevalent for defect prediction due to their adept capability in producing a less accurate baseline of defects as well as offering improvements in both precision and recall through proactive modeling[7–13]. In the last few years, improvements in NLP have created

[1]Department of CS&SE, TDR-HUB, Andhra University, Visakhapatnam, India. [2]Department of CS&SE, Andhra University, Visakhapatnam, India. [3]Department of CSE, Sir C.R.Reddy College of Engineering, ELURU, A.P, India. ✉email: mantena2377@gmail.com

new opportunities to improve software quality assurance methodologies. NLP plays a significant role in SDP by helping developers and quality assurance to automatically classify bug reports based on their content and severity and to identify the root causes of defects by analyzing the descriptions of bug reports[14,15]. This can help developers pinpoint the source of the problem and take corrective action.

Defect logs are textual documents where the record of errors and bugs encountered in the software during development and post-release gets recorded, which shoots unstructured text data. The descriptions can be categorized into topics related to potential root causes using topic modeling. Recent progress and innovations have emerged in the field of topic modeling. Latent Dirichlet Allocation (LDA) remains among the most commonly adopted and versatile topic extraction techniques[16]. Non-Negative Matrix Factorization (NMF) is a model for topic extraction that provides interpretable results and is particularly well-suited for tasks where non-negativity constraints are essential and when you want a parts-based representation of topics. BERTopic, a variant of the transformer-based BERT model, has emerged as a focus of research lately for its ability to extract coherent and semantically meaningful topics from text data[17].

### Problem statement

SDP has come a long way, most current models still focus only on identifying whether a software module has a defect. They often don't go further to explain why those defects occur. Typically, these models depend on structured metrics from source code and overlook the valuable insights that can be drawn from textual information like bug reports or defect logs. Because they miss this semantic layer, they struggle to help with more detailed diagnostic tasks, such as tracing the root cause of a defect. Also, many traditional classifiers handle each prediction in isolation, which makes them less effective when it comes to predicting related outcomes—like whether a defect exists and what caused it—at the same time. To overcome these limitations, this proposed work introduces a new SDP approach that integrates BERTopic for extracting key themes from text with a multi-output classifier that can predict both the defect and its underlying cause using defect log summaries.

### Research objectives

In this study, the main goal is to make SDP more useful—not just by detecting whether a defect exists, but also by figuring out why it occurred. Instead of relying only on code metrics, the approach taps into the meaningful information found in textual defect logs. To make this work, we use a method called BERTopic, which helps uncover topics in the text, and pair it with a multi-output classifier that can predict more than one outcome at once.

Here's what the study focuses on:

- First, cleaning and converting the defect logs into a form that can be analyzed semantically.
- Then, applying BERTopic to pull out topic patterns that relate to why defects occur.
- Next, building a classifier that does two things at once:

1. It tells us if a defect exists (yes or no), and.
2. It gives a possible reason behind that defect (from several possible causes).

- Finally, we test how well this setup works using different classifiers and performance measures like accuracy, F1-score, Jaccard index, and Hamming loss.

### Significance of the study

The study introduces a novel semantically enriched defect-prediction framework that attempts to fill two major gaps in the existing research: the lack of root cause analysis of defects and the underutilization of unstructured textual data. Using a transformer-based topic modeling technique called BERTopic, the framework makes use of the semantic richness in defect summaries to extract meaningful topics that essentially represent the root causes of defects. Another important feature is that a multi-output classifier is used to predict both defect presence and their associated root causes; this yields a richer and more actionable set of prediction outputs. Being able to make such a dual prediction is very useful in software maintenance settings, where speeding up debugging by means of not only identifying a defect but also knowing from where it most probably originated improves software quality. The framework also proposes a hybrid evaluation method that combines metrics for binary and multiclass performances, thereby proving its robustness and flexibility across different types of classifiers. Hence, intelligent defect management systems can be advanced with the proposed model, as well as automating aspects of quality assurance pipelines.

### Structure of the paper

The organization of the paper is described below:

Section 1 : Related Works: The existing body of research relevant to the study.
Section 2 : Proposed Methodology: Describes the suggested method in detail.
Section 3 : Experimental Results and Analysis: Presents and analyzes the experimental findings.
Section 4 : Conclusion and Future Work: Outlines the contributions and suggestions for future work.

## Literature review (Critical analysis & Comparison)

This literature review critically evaluates the evolution and current trends in SDP, comparing existing approaches with our proposed method to underscore advancements and highlight limitations addressed by this research. Data of various kinds was described: metric-based, textual, and version control-derived. The traditional techniques, using static code metrics and classical ML algorithms, were discussed, and, in the recent context, ensemble-based models were cited to improve prediction accuracy. NLP was introduced as a means of extracting semantic information from unstructured software artifacts. Topic modeling approaches such as BERTopic were further discussed as means for clustering defect-related information toward supporting the identification of root causes. The review ended with the topic of multi-output learning, which can make predictions on more than one defect attribute at a time, yielding broader and more actionable information for defect management.

## Types of datasets used in SDP

Several different types of datasets have been collected and used for SDP research, offering a wide variety of knowledge about the nature and incidence of software defects. Examples of such datasets include the defect log (which includes textual descriptions of reported issues), software measurement indicators (such as code complexity and quality quantified in various ways applicable to source code files), historical defect data that exposes patterns in previous defects, and mini-ancestry data from version control systems, including lines changed for each file. With these models in place, quality assurance efforts should be directed to highly prioritized modules containing possible defects at an earlier stage of the software development process, leading to more reliability and maintainability surrendered by Table 1 shows the different types of SDP datasets.

Despite textual defect logs' practical significance, mainstream SDP research rarely takes them into account. This deficiency sacrifices the practical utility and actionable insight coming from predictive models, owing to the fact that real-time defect management very heavily hinges on textual information, as provided in defect logs, bug reports, commit messages, and developer comments. Recognizing this gap, our study explicitly incorporates semantic textual information from defect logs using advanced topic-modeling methods. Using transformer-based semantic embeddings derived from textual defect data, this approach considerably enhances the practical applicability and interpretability of SDP in a way that is tightly coupled to real-time software development and quality-assurance practices.

## Traditional SDP approaches

Traditional methods mainly used static code metrics to assess the defect-proneness of software packages. These approaches usually applied classical ML algorithms such as decision tree (DT), logistic regression (LR), naïve Bayes, and support vector machines[26] and were often validated on benchmark datasets from NASA MDP, PROMISE, or Eclipse projects. Over time, these methods evolved to provide better learning capabilities for complex feature interactions from structured and semi-structured inputs through CNNs and RNNs[27]. Further, hybrid models were introduced to combine traditional classifiers with metaheuristic-based feature selection or ensembles to improve performance at the expense of addressing issues like overfitting or redundancy[28]. While the traditional and extended approaches gave strong baselines and optimized prediction accuracy for defects, nearly all remained binary classifiers. Our proposed method consists of performing semantic topic modeling (using BERTopic) to yield richer textual contexts and then combining those scenarios practically with multi-output classifiers. In doing so, it circumvents constraints in the past by including semantic information explicitly and by allowing multiple defect attributes to be predicted at the same time.

## NLP-based approaches in SDP (Textual bug data & deep Learning)

As the various software projects generate an enormous volume of unstructured text data in the form of bug reports, commit messages, defect summaries, etc., NLP has emerged out as a tool to extract insight from this information. In early works, agglomerated techniques such as TF-IDF and bag-of-words were exploited to carry out vectorization of the text and thereby enabling classifications of the attributes such as bug types and severity[29]. But such representations entirely neglect the context and generally fail to harbor semantic relationships among terms. To this, more recent researches incorporated word embeddings such as Word2Vec that allowed computer models to understand semantic similarity between terms, thus increasing their performance[30]. Then, after these studies, deep learning approaches such as CNN and RNN were applied to capture hierarchical or sequential patterns in bug reports[31]. More recently, transformer-based architectures, particularly BERT, have gained much acclaim in predicting bug severity and classifying issues[32]. Because these models create contextual embeddings that understand not only bug description content but also intent, they far outperform traditional methods in both accuracy and interpretability. In other words, NLP-driven approaches have hence evolved into key techniques for automated defect analysis, supporting triage, prioritization, and root cause identification.

Despite these advances, prior NLP-driven studies seldom integrated topic modeling insights systematically into predictive modeling frameworks. This integration gap severely limited the utility of defect prediction

| S.No | Year | Author(s) | Data set | Data set contains | References |
|---|---|---|---|---|---|
| 1 | 2019 | Iqbal, Ahmed, et al. | NASA (CM1, KC1, KC3, PC1, PC2 etc.) | software metrics and defect labels | [18] |
| 2 | 2023 | Khleel, et al. | PROMISE(KC1, KC3, PC1 etc.) | software metrics and defect labels | [19] |
| 3 | 2020 | Ferenc, Rudolf, et al. | Eclipse Bug Data | bug report and its resolution status | [20] |
| 4 | 2022 | Bala, Yahaya Zakariyau, et al. | AEEEM | metrics related to code changes and bug reports | [21] |
| 5 | 2011 | Wu, Rongxin, et al. | Relink | Contains code, bug reports, code metrics and changes | [22,23] |
| 6 | 2013<br>2023 | Ahmed, et al.<br>Dolga, Rares, et al. | Mozilla Bugzilla | Defect report attributes | [24,25] |

**Table 1**. Various SDP datasets.

outcomes in practical, actionable scenarios. Our study explicitly addresses this shortcoming by employing transformer-based topic modeling (BERTopic) as a critical semantic preprocessing step. The resulting topics and embeddings feed directly into our multi-output classifiers, uniquely enabling the simultaneous prediction of defect occurrence and root-cause attributes.

## Topic-modeling techniques in SDP

Topic-modeling techniques have proven beneficial in extracting meaningful patterns from defect-related textual data, thus permitting bug-report classification and root-cause analysis. Earlier methods including LDA were used to identify recurring defect-prone topics, whereas more recent techniques like BERTopic induce better semantic coherence with transformer-based embeddings. Such methods enrich the understanding of defect trends and, correspondingly, strengthen decisions made with regard to software maintenance.

Muhammad Laiq et al., 2023[33], proposed an LDA—a topic modeling method to help professionals know about preventive measures to prevent generating the same erroneous bug reports. First, they manually analyzed invalid bug report descriptions to identify related themes and employed expert-based validation for them. They identified similar characteristics of incorrect bug reports as important and used them to create a resistance strategy.

Tse-Hsun Chen et al., 2012[34] studied an approach to probabilistic topic discovery to estimate the topics and the impact of abstract design flaws on code quality. I proposed several metrics on these subjects to aid in the explanation of the entities' defect-proneness (i.e., quality). Their suggested metrics are important in that they consider each topic's defect history. Case analyses on numerous iterations of Mozilla Firefox, Eclipse, and Mylyn demonstrate that (i) a variety of topics are significantly more inclined to experiencing defects, (ii) these topics lean towards staying that way over time, and (iii) these topics offer better insight into code quality compared to current structural and historical metrics.

Xi Liu. et al., 2021[35], developed a novel intelligent defect classification model for radar software based on the LDA topic model. The suggested method combines a modified LDA model with radar software requirements, a text segmentation algorithm for defect detection utilizing a radar domain dictionary, and a top acquisition and classification method for radar software defects. To test its efficacy and applicability, the suggested method is applied to the typical flaws in radar software. The application findings show that the proposed approach's prediction accuracy and recall rate are up to 15–20% better than those of the previous defect classification methodologies.

Heng Li et al., 2018[36] utilized concepts of a code snippet that were automatically calculated, as well as the correlation between those topics and the possibility of a code segment being captured. They carried out an examination of six open-source projects and discovered a handful of "log-intensive" themes that are apt to be captured more regularly than the remaining topics. Each couple of the observed systems allots 12–62% of mutual topics, and the probability of logging such typical topics has a statistically substantial correlation of 0.35 to 0.62 across all the analyzed systems.

Wenjuan Bu et al., 2023[37] Automated tagging and tag updating using topic clustering and extracted subject words from software description texts with a customized BERTopic model The improvement of the data-space imbalance proposed by c-TF-IDF to balance the dataset is described for software classification in application function dimensions using the BERT-BiLSTM model. Investigative confirmation of two datasets reveals that the customized BERTopic model subclassifies 21 segments and 19 types in the given dataset. A software narrative text dataset is created and updated automatically. The grouping study achieves accuracy, recall rate, and F1 value as 0.92, 0.91, and 0.92, respectively.

In contrast, our methodology distinctly integrates BERTopic-derived semantic clusters directly into multi-output classifiers, enabling prediction not only of defect presence but also of detailed defect characteristics, including potential root causes. This explicit integration significantly advances beyond earlier topic-modeling-driven SDP studies by bridging semantic understanding directly to predictive actionable outcomes.

## Multi-Output learning in software engineering (Predicting multiple defect Attributes)

Defect prediction models, the classical ones, give weight to binary classifications. The models focus on identifying a particular module as defective or not. However, many real-world defects are multidimensional, they possess different types, causes, and priority levels. To deal with this, many recent researchers are studying the multi-output and multi-label learning framework. Pachouly et al.[38] have shown that, most of the time, defects may belong to many classes, for example, a bug, documentation, enhancement, etc. A multi-label classification method using classifier chains and deep learning was so proposed which showed improved performance in different classification metrics. In a similar way, Madaraboina et al.[39] focussed on applying multi-target learning techniques, including the Binary Relevance and Label Powerset approaches, to derive the best possible outcome for jointly predicting bug priority and resolution time by exploiting label dependency information. Grattan et al.[40] stress, through a systematic review, that these informative predictions are increasingly needed and should be modeled in ways that allow for the prediction of secondary characteristics such as severity, effort, or cause, apart from the binary output, thus supporting these investigations. Adding to the aforementioned efforts, Panda and Nagwani[41] model severity and priority predictions separately, though they emphasize that these are strongly correlated and could possibly be modeled in a multi-output fashion. Hence, these studies pave the way towards models that could simultaneously predict multiple defect attributes to give more actionable insights.

## Comparative summary

The Table 2 compares existing SDP research with the proposed approach across key research dimensions. It highlights specific gaps addressed by our methodology.

| Aspect of SDP Research | Existing Approaches | Proposed Approach |
|---|---|---|
| Types of Datasets | Mostly metric-based and historical data, limited textual datasets usage. | Extensive integration of textual defect logs and semantic data. |
| Traditional Approaches | Classical ML algorithms (e.g., DT, LR, SVM), CNN, RNN, primarily binary classifiers. | Integration of semantic embeddings with multi-output classifiers for enhanced predictive accuracy. |
| NLP-based Approaches | Use embeddings like Word2Vec, CNNs, RNNs, transformer models. | Comprehensive integration of transformer-based semantic embeddings (BERTopic) into predictive modelling. |
| Topic-modeling Techniques | Primarily descriptive use of LDA, limited predictive integration. | Direct integration of BERTopic-derived semantic insights into multi-output classifiers. |
| Multi-Output Learning | Very rare Multi-Output Learning. | Robust multi-attribute prediction using semantic embeddings, addressing practical defect management needs. |

**Table 2.** Comparative summary: existing vs. Proposed SDP Research.

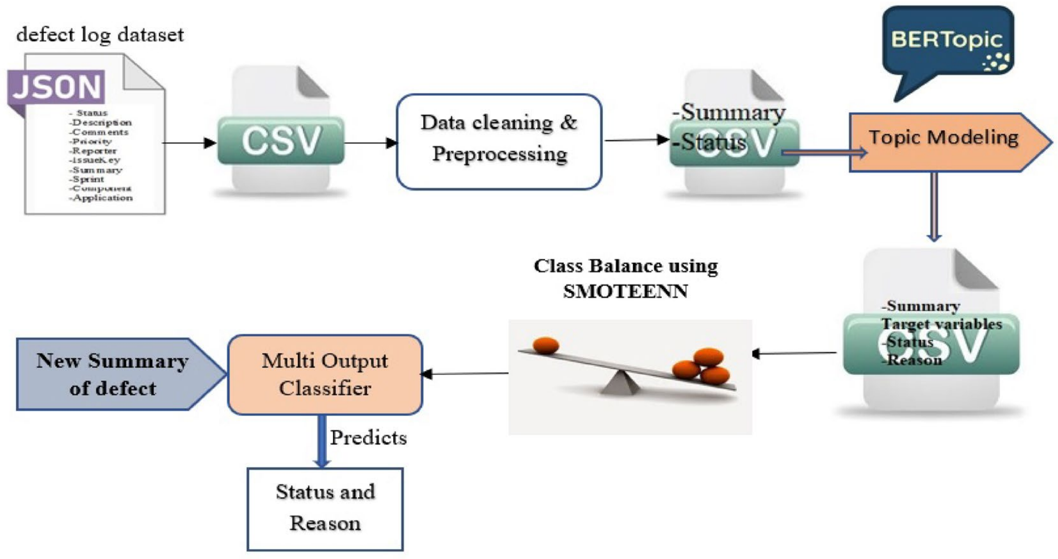


**Fig. 1.** The structure of the suggested BERT-MOC.

## Proposed method

Advanced topic modeling and classification techniques are utilized to predict software defects and identify their underlying root causes. A unique topic modeling method, BERTopic[42], is employed to extract root causes from software development defect report data. With BERTopic, which captures the semantic context from the defect log data, software defect patterns and their nature can be better understood. Created dynamic and highly interpretable topics from the semantics of the data. Employed a multioutput classifier that simultaneously predicts the presence of defects and the root cause of the defect.The model architecture of the proposed solution is illustrated in Fig. 1.

The proposed Algorithm 1, Multi-Output SDP, embraces the utilization of data preprocessing, topic modeling, and classification functionalities to predict software defects and the root cause behind them. First, the algorithm parses and preprocesses the defect log data, which is in JSON format, and gets it ready for the distinct analysis. Converting defect log data from JSON to CSV is essential for efficient data analysis and processing. CSV files are widely supported by data analysis and ML tools, are easier to understand and manipulate due to their tabular structure, and are generally smaller in size, leading to more efficient storage and faster operations. This format also facilitates easier data exchange and integration with various systems.

After that, the result of the data cleaning is prepared, normalized, and tokenized, following the stop words eliminated to enhance data conduciveness. The algorithm uses BERTopic for its identity to perform topic modeling based on the summaries of the reported defects and reveals the set of key topics, which might give an understanding of the nature of the defect reason found. Addressing the class distribution disparity, the algorithm employs the SMOTEENN (SMOTE and ENN), which helps in achieving better balancing of the dataset. Also employed is vectorization and singular value decomposition, commonly referred to as SVD, which converts the textual data into numerical form that is suitable for the classifier.

Other classifiers used include LR, DT Classifier, Kneighbor Classifier, Random Forest Classifier, and Ensemble Method-Voting, all of which are trained in MultiOutputClassifier for the examination of defect reasons and statuses. At last, the trained model is tested using some performance measures to achieve good results in predicting and analyzing the defects. This comprehensive approach enhances defect management processes by offering predictive insights into software issues and their potential causes.

```
Input: defect log JSON file (Summary, Status)
Output:  predicted reasons and status(valid, Invalid), performance metrics

Step 1: Read and Convert JSON Data
        file ← read_json(defect_log_data)
        data ←json_to_csv(file)
Step 2: Data Cleaning and Preprocessing
          for each Summary in data
               Summary ← clean_text (Summary)
             data ← drop_ unnecessary_columns(data)
         for each Summary in data
               Summary ← lower_case (Summary)
               words ← tokenize (Summary)
               words ← remove_stop _words(words)
               words ← stemmer(words)
               document <- join(words)
Step 3:  Topic Modelling
           // Use BERTopic for topic modelling
           topics, prob ← Bertopic (data. Summary)
          // Assign topic names
          Topic_names ← {-1:"As per design", 0:"Test Data issue", 1:"Duplicates", 2:"Test Environment"}
          for each topic in topics:
              topic ← Topic_names[topic]
         // Merge topic data to original data
          data. Reason ← topics
Step 4: Handle Class Imbalance
          balanced_data ← SMOTEENN(data)
Step 5: Vectorization and Singular Value Decomposition (SVD)
        vectorizer_data ← TfidfVectorizer(balanced_data)
        svd_data ← TruncatedSVD (vectorizer_data)
Step 6: Train Classifiers and Evaluate
        classifiers ← RandomForestClassifier, LogisticRegression, DecisionTreeClassifier, Kneighbor Classifier,
Ensemble method-Voting
        for each classifier in classifiers
            model ← classifier ()
        multi_output_classifier ← MultiOutputClassifier(model)
        predictions ← multi_output_classifier. predict (svd_data)
Step 7: Make Predictions on New Data
        // Preprocess the new data
        new_summary ← "example summary text"
        summary ← clean_text(new_summary)
        summary ← pre-process(summary)
        summary ← TfidfVectorizer(summary)
        summary ← TruncatedSVD (summary)
       // Make predictions using the trained model
       predictions ← multi_output_classifier. predict(summary)
End Algorithm
```

**Algorithm 1**. The proposed Multi Output SDP algorithm.

## Textual data preprocessing

To ensure high-quality semantic modeling of defect summaries, we applied a structured textual preprocessing pipeline. Each preprocessing step contributes to transforming raw, unstructured text into a normalized, semantically meaningful format suitable for vectorization and classification. Let the raw defect dataset be:

$$D = \{d_1, d_2, \ldots, d_n\} \tag{1}$$

where $d_i \in \mathbb{S}$ represents the i-th textual defect summary, and $\mathbb{S}$ denotes the space of all raw textual summaries.

**Step 1** Lowercasing and Bracket Content Removal.

Natural language text is case-sensitive and contains noisy elements like metadata or code embedded in brackets. Lowercasing standardizes all tokens, reducing the feature space. Removing square-bracketed content eliminates irrelevant annotations often present in bug-tracking systems. brackets() removes substrings matching \[.*?\].

$$d_i^{(1)} = f_1\left(d_i\right) = lower\left(d_i\right) \setminus\ brackets\left(d_i\right) \tag{2}$$

**Step 2** Punctuation and Alphanumeric Token Removal.

Punctuation and tokens with embedded digits (e.g., variable names, error codes) are unlikely to contribute semantically to defect understanding and may increase noise during tokenization. $\setminus$: set difference operator, meaning "remove these words".

$$d_i^{(2)} = f_2\left(d_i^{(1)}\right) = d_i^{(1)} \setminus\ \{w \mid\ w \in\ d_i^{(1)}, w \in\ \mathcal{P} \cup\ \mathcal{A}\} \tag{3}$$

**Step 3** Duplicate Removal.

Redundant textual records bias the learning process by over-representing certain patterns. Removing duplicates improves data quality and ensures diversity of learning examples.

$$d_i^{(3)} = d_i^{(2)} \setminus\ \{d_i \mid\ \exists\, j < i : d_j = d_i\} \tag{4}$$

**Step 4** Tokenization.

Tokenization segments text into discrete lexical units (words) that form the base elements for further semantic processing. Transformation function is:

$$T_i = \tau\ \left(d_i^{(3)}\right) = \{w_1, w_2, ....w_k\} \tag{5}$$

Where $\tau$ is word-level tokenizer (e.g., NLTK's word_tokenize) and $T_i$ is the token list for the -th summary.

**Step 5** Stopword Removal.

Stopwords (e.g., "the", "is", "and") are high-frequency function words with low semantic contribution. Removing them reduces noise and dimensionality. Transformation function is:

$$T_i'\ = f_3\left(T_i\right) = \{w_j \in\ T_i \mid\ w_j \notin\ \mathcal{S}\} \tag{6}$$

Where $\mathcal{S}$ = standard English stopword list. $f_3$ filters out uninformative tokens.

**Step 6** Stemming.

Stemming reduces words to their root forms, consolidating inflected or derived forms. This reduces feature sparsity and enhances semantic grouping. Transformation function is: Where $\sigma$ = stemming function (PorterStemmer).

$$T_i'{}'\ = \sigma\ \left(T_i'\right) = \{\sigma\left(\,w_j\right) \mid\ w_j \in\ T_i'\ \} \tag{7}$$

**Step 7** Document Reconstruction.

After all transformations, the stemmed tokens are recombined into normalized strings suitable for further processing (e.g., vectorization, topic modeling).

$$D_i^{final} = Join\left(T_i'{}'\ \right) = w_1\ +\ w_2 +\ ....+\ w_k \tag{8}$$

Where $T_i'{}'\ = \{w_1\ +\ w_2 +\ ....+\ w_k\}$be the set of stemmed tokens for the -th document. $Join\left(T_i'{}'\ \right)$ is the function that joins the tokens with whitespace to reconstruct the document. $D_i^{final}$ is the final preprocessed text used for vectorization or embedding of -th document.

The final preprocessed dataset is defined as:

$$D^{final} = \{D_1^{final}, D_2^{final}, ..., D_m^{final}\},\ \ with\, m \leq\ n \tag{9}$$

## Topic modelling : BERTopic

BERTopic (short for BERT topic modeling) is a topic modeling technique that uses state-of-the-art sentence embeddings from Google Research's transformer-based model to identify topics in text data. In this study, it is employed to extract the root causes of software defects from a defect log dataset. Key components of BERTopic include:

### *BERT embeddings*

The text data is converted into BERT embeddings, which constitute the first step in BERTopic. Encoding contextual and semantic information in sentences, SBERT "Sentence-BERT" embeddings are essential in NLP applications including information retrieval, document categorization, question answering, and recommendation systems[43]. "all-MiniLM-L6-v2" is a sentence-transformer, pre-trained model. It's a variant of the MiniLM model in SBERT. It converts sentences and paragraphs into a high-dimensional, dense vector space, supporting tasks such as clustering or semantic search. The default embedding models are "all-MiniLM-L6-v2" for the languageEnglish" and "paraphrase-multilingual-MiniLM-L12-v2" for the e language "multilingu.

*Dimensionality Reduction(UMAP)*
The nature of BERT embeddings is that they are high-dimensional; therefore, number of input variables reduction approaches like UMAP (Uniform Manifold Approximation and Projection) will be employed to compress the dimensions for topic resolution in BERTopic. After embedding the text into high-dimensional BERT vectors, these very high-dimensional vectors are reduced down to a lower-dimensional space for computational simplicity and better visualization. UMAP: A method for lowering dimensionality that preserves more of the global structure. It ensures documents close to each other in the original space should also be close in embedding spaces, even after reduction[44,45].

The mathematical underpinnings of UMAP involve aspects of topology, graph theory, and optimization. Here is a summary of the major mathematical components of UMAP:

Sparsified Correlation Matrix, which can be used to construct a KNN graph as follows: Compute the pairwise distances between data points in your dataset X. Let $d\left(a,b\right)$ denote the space between two data points a and b in the n-dimensional corresponding space.

$$d\left(a,b\right)=\sqrt{\sum\nolimits_{i=1}^{n}\left(b_i-a_i\right)^2} \tag{10}$$

In Eq. (10), $a_i,\ b_i$ are the Euclidean vectors in n-space. Lets choose $N\left(a_i\right)$ be the set of k nearest neighbors of the data point $a_i$. For that, construct a Fuzzy Simplicial Set: Calculate the degree of membership (a weight) between pairs of data points using a kernel function. Let $P_{ij}$ represent the degree of membership (weight) between data points $x_i$ and $y_i$. It can be calculated using a kernel function:

$$P_{ij}=exp(-\frac{d(x_i,x_j)-\rho\ _i}{\sigma\ _i}) \tag{11}$$

In Eq. (11), for each $x_i$ define $\rho\ _i$ and $\sigma\ _i$, where $\rho\, i$ is the minimum distance from $x_i$ and set $\sigma\ _i$ to be the value such that

$$\sum\nolimits_{j=1}^{k}\exp\left(\frac{-\left(d(\ x_i,x_j)-\rho\ _i\right)}{\sigma\ _i}\right)=\log_2\left(k\right) \tag{12}$$

Initialize Low-Dimensional Embedding and Optimization: Initialize Y for the data points randomly. Let Y be a matrix whose each row represents the low-dimensional embedding of a data point. Calculate the distances between data points in a new space Y. For implementation, $d_Y(y_i,y_j)$ represents distance between low dimensional embeddings $y_i$ and $y_j$ in Y. For each data point in Y, first calculate a membership strength value of its neighbors in Y.

$$q_{ij}=\frac{1}{\left(1+a(y_i-y_j\right)^{2b}} \tag{13}$$

In Eq. (13), $q_{ij}$ signifies the membership strength between low dimensional embeddings $y_i$ and $y_j$ in Y. Calculate the cross-entropy loss between fuzzy simplicial sets in high and low-dimensions. C is the cross-entropy loss

$$CE(X,Y)\ =\sum\nolimits_{i=1}^{n}\sum\nolimits_{j=1}^{m}\left[p_{ij}\left(X\right)\log\left(\frac{p_{ij}\left(X\right)}{q_{ij}\left(Y\right)}\right)+\left(1-p_{ij}(X\ )\right)log\left(\frac{1-p_{ij}\left(X\right)}{1-q_{ij}\left(Y\right)}\right)\right] \tag{14}$$

It is computed based on the dissimilarity between the fuzzy simplicial sets at a complex high space from that in the latent space. Compute the gradient of the loss relative to the locations of data points in Y. The variable $\delta\, yc$ is denoted as gradient of the loss representation the position of data points in Y. Using gradient descent with learning rate r update the position of data points in Y. $Y=Y-r.\delta\, yc$

**Clustering: HDBSCAN**: DBSCAN is a non-parametric clustering algorithm that groups data points based on density. It is used for grouping data points into clusters based on their spatial density[46]. ε (epsilon or the highest pairwise distance within the same cluster) and *MinPts* (the least number of data points needed to define a dense area) are the two user-defined parameters.

A data point P is considered a main point if there are leastwise *MinPts* data points among a distance of ε (eps) from P. In other words, the neighborhood of P contains enough points to form a dense region. Data points that are not core but are in the ε distance of a primary point are considered "density-reachable." These are the border points. Two data points P and Q are said to be "directly density-connected" if they are both core points and are within ε distance of each other. Common distance functions used in DBSCAN include:

**Euclidean distance** (The most familiar equation used to find the distance between numerical data) Also, the Euclidean distance between two points P and Q in a D-dimensional space can be represented as.

$$Dist(P,Q)=\sqrt{\sum\nolimits_{i=1}^{\mathrm{D}}(\mathrm{P}_i-\mathrm{Q}_i)^2} \tag{15}$$

In Eq. (15), $P_i$ and $Q_i$ are the values of the i-th feature of points P and Q, respectively.

**Manhattan distance** Also called "city block" or "taxicab" distance is the distance from point A to B is measured by summing the absolute variances among the coordinates along each dimension.

$$Dist\,(P,Q)=\sum\nolimits_{i=1}^{D}|P_i-Q_i| \tag{16}$$

**Cosine similarity** Used to be fish for the magnitude of the vector, especially in text or any other related high-dimension data. It calculates the cosine value of an angle between two vectors and goes from – 1 (the inverse) to 1 (the same direction).

$$Cosine\ Similarity(P,Q)=\frac{\sum_{i=1}^{D}P_i.Q_i}{\sqrt{\sum_{i=1}^{D}{P_i}^2}.\sqrt{\sum_{i=1}^{D}{Q_i}^2}} \tag{17}$$

In DBSCAN, a cluster is defined as a data series that is mutually density-connected. DBSCAN grows the cluster recursively by appending all points that are directly density-connected to it, starting from a vital point. "Noise points," or outliers, are points with neither core points nor density-reachable points. There is no cluster to which these points belong.

**Cluster formation process** In order to start, DBSCAN selects an arbitrary data point. It creates a fresh cluster and joins any points that are directly connected to density if the point is determined to be a core point. This process is continued until no additional essential points are added. Once each point of data is either classified as noise or assigned to a cluster, the algorithm proceeds to the next undiscovered data point, and so on.

*Topic representation*

Each cluster is considered a topic once documents are clustered. Each cluster is changed to a single document rather than being represented by multiple documents. To make topics more interpretable, BERTopic extracts the most frequent or representative words within each cluster document using the c-TF-IDF score $(W_{x,c})$. These words are used to label and describe the topics. The top words give a human-readable insight into the content of the topic.

$$W_{x,c}=tf_{x,c}\times\ \log\left(1+\frac{A}{f_x}\right) \tag{18}$$

In Eq. (18), $tf_{x,c}$ = frequency of word x in class c, $f_x$ = frequency of word x across all classes, $A$ = average number of words per class.

The four common topics about the summary of defect logs are displayed in Table 3. The quantity of postings related to any individual topic is indicated in the count column. The topics' names, as produced by BERTopic, are listed in the Name column. The words that best represent each of the related topics are listed in the Representation column.

Every topic differentiated by the model consists of a set of words along with the score tied to it. These scores indicate appropriateness of each word to the topic. Word scores are used to indicate the degree of emphasis that a specific word will have in the area of concern. Higher scores mean words that form a better approximation to the semantic centrality of a given topic. Looking at the word score, it is easier to determine the topic of the specific column as the top words and their score give a brief overview of the subject. It assists in the process of giving appropriate names to the topics. Figure 2 shows the word scores for our four topics.

| Topic | Count | Name | Representation |
|---|---|---|---|
| 1 | 3097 | app_cert_cabin_page | [app, cert, cabin, page, book, display, qa, screen, rt, user, app, vv, voyag, detail, crew, load, tablet, incorrect, show, error] |
| 2 | 188 | page_scroll_desktop_breakpoint | [page, scroll, desktop, breakpoint, requir, per, design, ui, filter, chrome, display, back, hd, medium, open, firefox, voyag, select, home, dwpdevchoos] |
| 3 | 58 | case_test_close_email | [case, test, close, email, chat, creat, exal, popul, qa, ignor, owner, servic, pleas, bug, send, lead, queue, prodsailoraccount, web, survey] |
| 4 | 41 | guid_modul_editori_port | [guid, modul, editori, port, app, bimini, publish, compon, updat, card, style, discov, font, red, hero, revis, cenot, underlin, rich, quot] |

**Table 3.** Four topics generated by BERTopic.

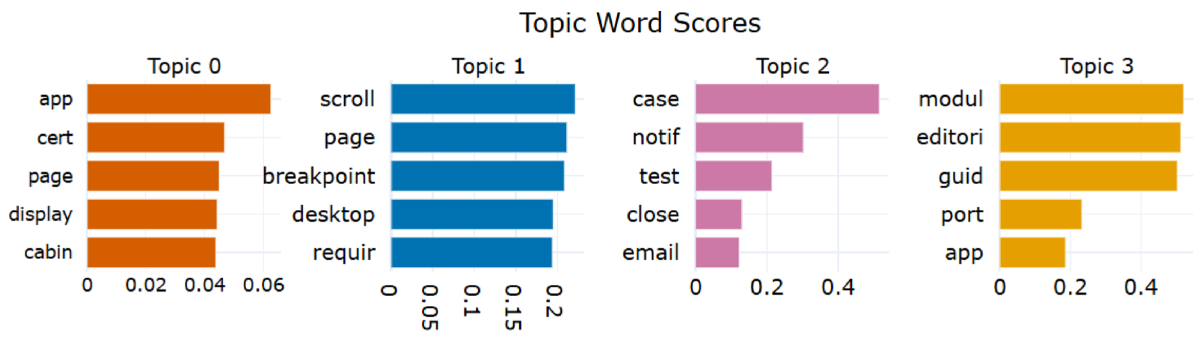


**Fig. 2**. Word scores for four topics.

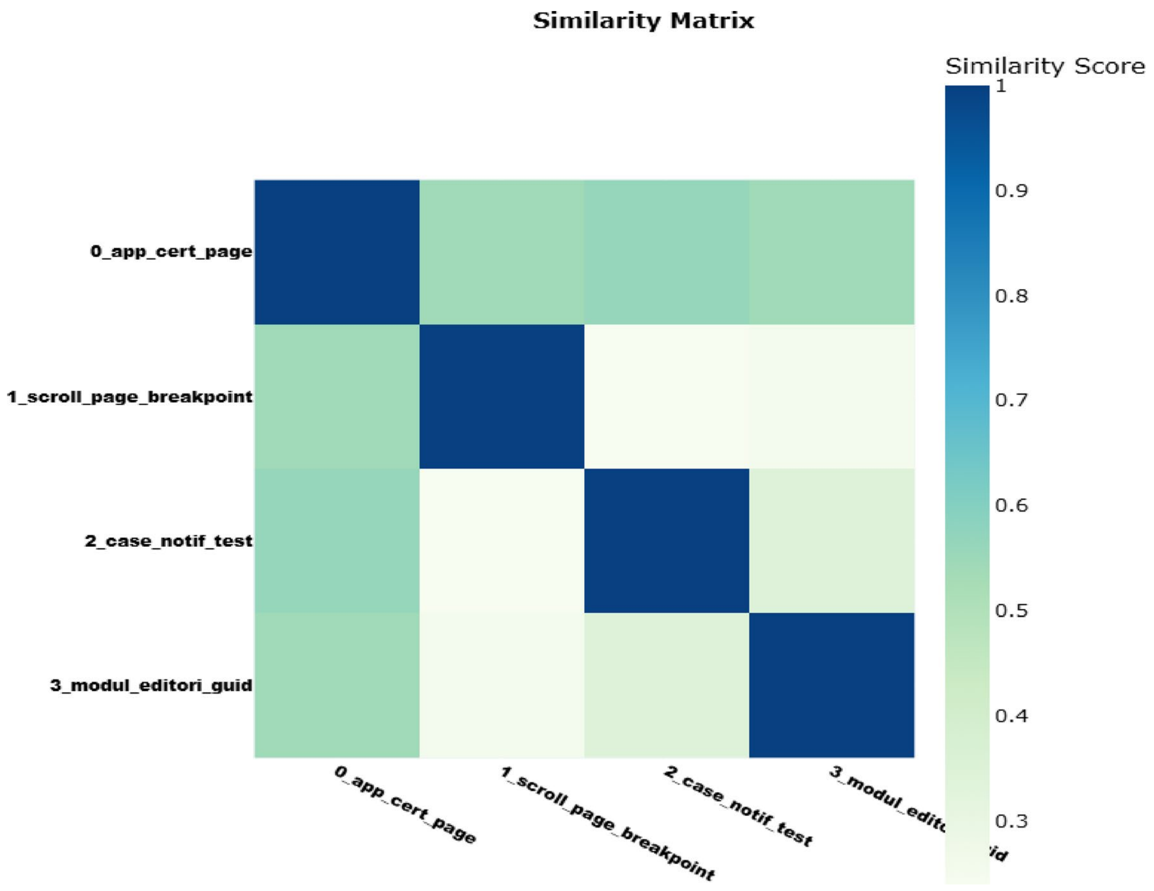


**Fig. 3**. Intertopic distance map of four topics.

A two-dimensional distance map and similarity matrix of four topics are shown in Figs. 3 and 4, providing information about the topics' intertopic distances and relationship organization. The similarity matrix is crucial for understanding how closely related different topics are, as shown in the below Fig. 3.

In the posttopic extraction stage using BERTopic, each topic is described by a set of high-weight keywords that describe the semantic content of the grouped defect log entries. It improves human interpretability and aligns the results for practical defect analysis. Each topic is manually labeled through the analysis of its most representative keywords. Topic 0 with keywords like "app," "cert," "incorrect," and "error" was considered 'Test Data issue', as the keywords suggest some problems with data display. Words such as "scroll," "desktop," "breakpoint," and "design" are found in Topic 1 and are therefore labeled 'As per design', referring to UI behavior aligned to expected design. Topic 2 has the words "case," "close," "email," and "ignor," which point to duplicate or premature closure of test cases and hence are labeled 'Duplicates'. In contrast, Topic 3 sorts among words like "guid," "modul," "publish," and "editori," stand for errors in a content management or publishing environment and are labeled 'Test Environment'. Each of these keyword-based labels reflects a separate root cause category relevant to the analysis of software defects.

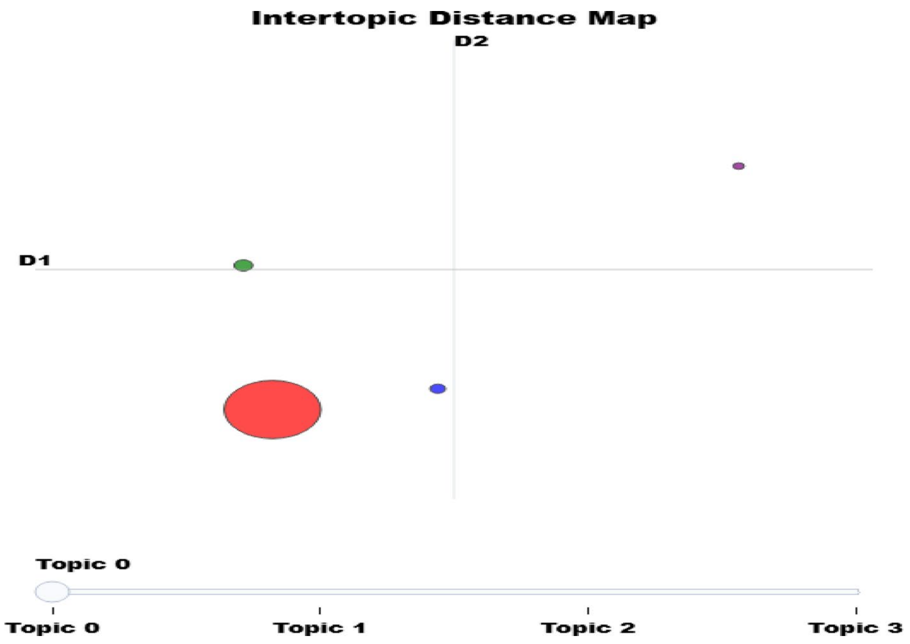


**Fig. 4**. Similarity matrix of four topics.

## Class balancing technique: SMOTEENN

SMOTEENN combines SMOTE with ENN, which is a composite resampling method. SMOTE[47] creates artificial data points for the underrepresented class, while ENN removes ambiguous or misclassified samples of the majority class, rendering the dataset more balanced and cleaner. This algorithm works with an unbalanced data set with minority and majority classes. There are two stages: first, extend the minority class samples by applying SMOTE. Next, the clean technique based on ENN is proposed to further improve the classification system by eliminating noisy and potential misclassification samples in the dataset[48].

*SMOTE (Synthetic minority Over-sampling Technique)*

SMOTE creates simulated points by interpolating across original under-sampled class instances and their neighbors. Let $\chi_{min} = \{x_1, x_2, ...., x_n\} \subset \mathbb{R}^d$ be the set of minority class data points. $k$ be the number of nearest neighbors. $\lambda \sim \mathcal{U}(0,1)$ represent a random variable following a uniform distribution. Then, for each $x_i \in \chi_{min}$, a synthetic sample $\tilde{x}$ is generated by the formula (19).

$$\tilde{x} = x_i + \lambda .(x_{NN} - x_i) \tag{19}$$

Where $x_{NN}$ is a randomly chosen neighbor from the k-nearest neighbors of $x_i$. This process creates new points along the line segments between $x_i$ and its neighbors, increasing minority representation. Let's take $D$ is an original imbalanced dataset. After applying SMOTE, a balanced dataset $D_{SMOTE}$ with synthetic samples is (20):

$$D_{SMOTE} = D \cup \tilde{x} \tag{20}$$

*ENN (Edited nearest Neighbors)*

ENN performs instance selection by removing high-frequency samples that are wrongly predicted by their nearest neighbors. To examine whether a data instance $x_i$ is noisy or inconsistent with its neighbors, predict its label based on the labels of its nearest neighbors and compare that prediction with its actual label. If predicted label is not same as actual label then then remove $x_i$. This operation is performed on both original and synthetic samples in $D_{SMOTE}$, although in practice, it's especially useful in cleaning the majority class and borderline synthetic minority samples. For each instance $x_i \in D_{SMOTE}$, find its k-nearest neighbors $\mathcal{N}_k(x_i)$. Let the predicted label $\hat{y_i}$( 21) be the majority class (mode) among its neighbors.

$$\hat{y_i} = mode\left(\{ y_j \mid x_j \in \mathcal{N}_k(x_i)\}\right) \tag{21}$$

If $\hat{y_i} \neq y_i$, then remove $x_i$ from $D_{SMOTE}$. After applying the ENN $D_{SMOTEEN}$ is the balanced dataset.

## Multi-output classifier

A multi-output classifier, also known as a multi-output model, is designed to handle problems with multiple target variables, where each target variable belongs to a different class or category. A multi-output classifier predicts multiple target variables simultaneously. Multi-output models learn a function (22):

$$f : \mathbb{R}^d \rightarrow \mathbb{Y}_1 \times \mathbb{Y}_2 \times \ldots \times \mathbb{Y}_k \tag{22}$$

Where $\mathbb{R}^d$ is the feature space. $\mathbb{Y}_1$: Output space for defect status (Valid, Invalid), $\mathbb{Y}_2$ : Output space for root cause categories (Test Data issue, Design issue, Duplicates, Test Environment issue). The four elements in Fig. 5 make up a proposed multi-output classifier pipeline:

*Tfidf Vectorizer*
The Term Frequency-Inverse Document Frequency (TF-IDF) is used in this stage to transform text data into quantitative feature vectors. This approach is implemented to preprocess text data without providing the original text data to a ML model, and it is typically used for text classification jobs.

*Truncated SVD*
It is a dimensionality reduction strategy. In our experiment, the dimensionality of the TF-IDF vectors was reduced to 3000 components. Dimensionality reduction can help reduce computational complexity and the risk of overfitting.

*Estimator (lr_multi)*
"estimator" refers to the base classifier that is used for each target variable in multi-output classification. The MultiOutputClassifier is a meta-estimator that takes a single classifier (the "estimator") and fits it separately to each target variable in your multi-output problem. This allows the application of a single classification algorithm to all target variables simultaneously. A base classifier, or estimator, is a basic ML model that is used to autonomously forecast every target variable in the multi-output classifiers. Multi-output classification entails the simultaneous prediction of multiple target variables, which can be effectively addressed by employing a different base classifier for each of them. The selection of the base classifier may change according to the particular nature and needs of the dataset. Each base classifier independently receives training on the same set of features but with dissimilar target variables. This method makes possible customized forecasts for each output, building upon the strengths of the particular base classifier selected. When several predictions are combined from various base classifiers, this multi-output classifier can handle complex predictive tasks involving many target variables that could be interdependent, thus improving overall predictability performance as well as making it more reliable.

*Rationale behind the Multi-Output classifier and base estimators*
Each defect instance can have a multiple target attributes in real-world software defect scenarios, such as defect status (valid, invalid) and root cause categories (e.g., test data issue, design flaw, duplicate, test environment issue). Asiatic binary and multiclass classifiers do not suffice to model such multi-dimensional outputs. The multi-output approach to classification is therefore implemented. The meta-estimator of MultiOutputClassifier involves breaking the multi-output problem down into independent subproblems. Each subproblem is solved by a common or special base estimator, chosen according to its ability to treat that particular output's characteristics. The following base classifiers were considered:

**Logistic regression (LR)** Efficient for linearly separable outputs and provides probabilistic interpretations.

**Decision tree (DT)** Captures non-linear relationships and performs well on imbalanced or categorical data.

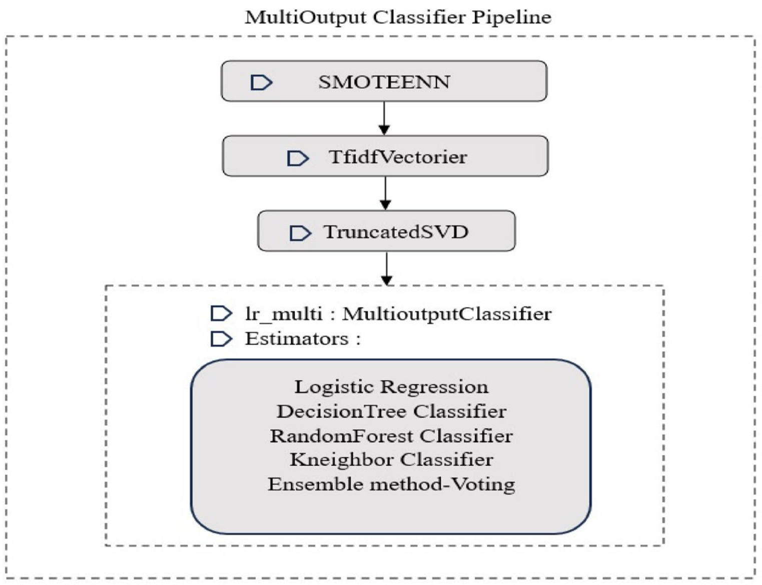


**Fig. 5**. Multi-output classifier pipeline.

**K-Nearest neighbors (KNN)** Effective for localized decision boundaries, useful when defect types have high similarity.

**Random forest (RF)** Ensemble of decision trees that offers robustness against noise and high variance.

**Voting classifier** Combines multiple classifiers to improve generalization by aggregating diverse prediction behaviors.

This architecture provided options to customize the learning strategy of each output dimension while exploiting the shared-feature representation. The pipeline included TF-IDF vectorization to generate meaningful features out of raw text, while Truncated SVD aimed at reducing dimensionality and making the computations much more efficient. This reduced feature space helped in combating overfitting and aided in training each base classifier effectively. By exploiting the predictive power of several base estimators within a multi-output setting, our model is capable of providing detailed and multi-faceted SDP—allowing developers to identify not just the issues but also the likely root cause.

## Experimental results and discussion

This part includes the dataset description, evaluation metrics to evaluate the impact of the suggested technique, and real experimentation.

### Dataset

The dataset is gathered from the proprietary internal defect tracking system for an enterprise application. It consists of 6,483 defect records, each having structured and unstructured information related to software defect reporting and analysis. Being proprietary, it is not made public, yet it describes a real-world scenario for software maintenance and debugging.The dataset comprises 11 fields. Key statistics of the dataset are outlined in Table 4.

- Summary and Description provide both concise and lengthy textual explanations of the defect.The summary contains only two to three sentences that basically convey the idea of the defect.
- Status is a binary classification indicating whether the issue is defective or clean.
- Other metadata such as priority, reporter, assignee, component, application, sprint, comments, and issue keys enable more profound analysis.

### Performance metrics

In the case of predicting the presence or absence of defects in a module, which is a binary classification task, the standard confusion matrix serves as an appropriate tool for evaluating model performance.When addressing root cause prediction, which involves multi-class classification across four distinct categories, several performance metrics can be derived from the confusion matrix (CM) to assess how accurately the model distinguishes between the underlying causes of software defects. It provides an overview of prediction results and allows us to compute various performance measures. The CM for a four-class classification issue is a 4×4 matrix, with the element at the $i^{th}$ row and $j^{th}$ column denoting the total number of samples of class $i$ that were wrongly predicted to be class $j$ is displayed in Fig. 6.

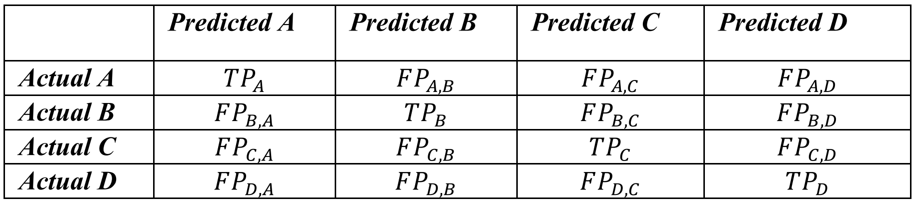
Where:
$TP_X$ is True Positives for class X
$FP_{X,Y}$ is False Positives for class X predicted as Y.
**Accuracy**:
The total instances ratio of correctly predicted instances is what accuracy is all about.

$$Accuracy = \frac{TP_A + TP_B + TP_C + TP_D}{\sum_i^j ConfusionMatrix_{i,j}} \tag{23}$$

**F1-Score**:

| Statistic | Value/Description |
|---|---|
| Total Records | 6,483 |
| Data Type | Tabular with both structured and unstructured fields |
| Text Fields | \`Summary\`, \`Description\`, \`Comments\` |
| Average Summary Length | ~ 18–25 words |
| Average Description Length | ~ 100–150 words (long procedural explanations and reproduction steps) |
| Comment Field | Nested JSON, contains multi-author, multi-timestamp developer discussions |
| Binary Class Label | \`Status\`: Invalid Defect (0) vs. Valid defect (1) |
| Class Imbalance | Invalid defects: 40%, Valid defects: 60% |
| Other Metadata Fields | \`Priority\`, \`Sprint\`, \`Application\`, \`Reporter\`, \`Assignee\`, \`IssueKey\` |
| Format | Originally JSON, preprocessed and exported to CSV |

**Table 4.** Key statistics of the dataset.



| | ***Predicted A*** | ***Predicted B*** | ***Predicted C*** | ***Predicted D*** |
|---|---|---|---|---|
| ***Actual A*** | $TP_A$ | $FP_{A,B}$ | $FP_{A,C}$ | $FP_{A,D}$ |
| ***Actual B*** | $FP_{B,A}$ | $TP_B$ | $FP_{B,C}$ | $FP_{B,D}$ |
| ***Actual C*** | $FP_{C,A}$ | $FP_{C,B}$ | $TP_C$ | $FP_{C,D}$ |
| ***Actual D*** | $FP_{D,A}$ | $FP_{D,B}$ | $FP_{D,C}$ | $TP_D$ |

**Fig. 6**. Confusion Matrix for 4 Classes.

| | **Metric_type** | | | |
|---|---|---|---|---|
| **MOC-Base classifier** | **Accuracy** | **F1_score** | **Jaccard_score** | **Hamming_loss** |
| Logisticregression | 0.91 | 0.91 | 0.84 | 0.08 |
| Decision tree | 0.81 | 0.81 | 0.69 | 0.18 |
| RandomForestClassifier | 0.93 | 0.93 | 0.87 | 0.06 |
| KNeighborsClassifier | 0.81 | 0.79 | 0.68 | 0.18 |
| Ensemble method-voting | 0.94 | 0.94 | 0.9 | 0.05 |

**Table 5**. Performance evaluation of target variable: Status.

The F1-score for class is the harmonic mean of precision and recall.

$$F1 - Score_X = \frac{2 \times \ Precision_X \times \ Recall_X}{Precision_X + Recall_X} \tag{24}$$

**Jaccard Index**:

The Jaccard Index for class is the relative size of the intersection to the size of the union of the predicted and actual labels for the class.

$$\text{Jaccard}_X = \frac{TP_X}{TP_X + \sum_{i \neq X} FP_{i,X} + \sum_{j \neq X} FP_{X,j}} \tag{25}$$

**Hamming Loss**:

The percentage of inaccurate predictions—that is, guesses that don't match the true labels—among all of the predictions is known as the Hamming loss. The total of the off-diagonal components, or misclassifications, can be used to compute inaccurate predictions in the context of the CM. For a multi-class classification issue with classes, given a confusion matrix C:

$C_{ij}$ signifies the number of instances of class $i$ that were predicted as class $j$.

The Hamming loss $H$ can be calculated as:

$$H = \frac{\sum_{i=1}^{L} \sum_{j=1, j \neq i}^{L} C_{ij}}{\sum_{i=1}^{L} \sum_{j=1}^{L} C_{ij}} \tag{26}$$

In Eq. (14), $\sum_{i=1}^{L} \sum_{j=1, j \neq i}^{L} C_{ij}$ is the sum of off-diagonal elements (misclassifications) and $\sum_{i=1}^{L} \sum_{j=1}^{L} C_{ij}$ is the total number of predictions.

## Result analysis

Our dataset contains two target variables, one with two classes and the other with four classes. The MultiOutputClassifier approach is implemented using base classifiers, ensuring that both target variables are handled appropriately and evaluated with suitable metrics, leveraging the strengths of the classifier to tackle multiple outputs simultaneously. The performance of the proposed model is evaluated using accuracy, F1-score, Jaccard score, and Hamming loss, selected based on their suitability for both binary and multi-class classification tasks.

Using various base classifiers in a multi-output classification framework, the performance evaluation of the target variable "status" yields unique metrics for every model. Table 5 summarizes the performance metrics for predicting the target variable Status. The ensemble voting method outperforms all individual classifiers, achieving the highest accuracy (0.94), F1-score (0.94), Jaccard score (0.90), and the lowest Hamming loss (0.05), indicating superior predictive capability and minimal error. Random Forest follows closely with strong overall performance, while Logistic Regression also performs well with balanced scores. In contrast, Decision Tree and K-Nearest Neighbors yield moderate results, showing relatively lower accuracy and higher Hamming loss.

Figure 7 shows the visual comparison of classifiers, while Fig. 8 highlights the different metrics scores across models for status.

A multi-output classification framework's performance evaluation of the target variable "reason" utilizing different base classifiers indicates significant variations in each classifier's efficacy. Table 6 summarizes the performance metrics for predicting the target variable Reason. Figure 9 shows the visual comparison of classifiers, while Fig. 10 highlights the accuracy, F1-score, Jaccard score, and Hamming loss across models. The RF classifier delivers the highest accuracy and F1-score (0.97), indicating exceptional overall performance. Logistic Regression and K-Nearest Neighbors also show strong results, with accuracies of 0.94 and 0.95 respectively, and low Hamming losses (0.05 and 0.02). The Decision Tree classifier performs moderately with an accuracy of 0.90 and F1-score of 0.90. While the ensemble voting method shows a high F1-score of 0.97, it records the lowest accuracy (0.61) and the highest Hamming loss (0.22), suggesting inconsistent predictive behavior despite strong class-wise performance.

The comparison provides the findings of the Random Forest Classifier and Ensemble Method using voting on both "status" and "reason" targets, revealing that the two algorithms have high accuracy, high F1 scores for both "status" and "reason" targets, and low Hamming loss for "status" and "reason.". Logistic regression likewise shows good performance, especially for multiclass "reason.". Neither the DT nor the K-neighbors classifier worked as well, particularly for the "reason" target, which seemed to be more intricate. When comparing the different approaches for multi-output classification, it can be stated that ensemble methods have the highest performance in all the aims.

## Implications of predicted defect root causes

Determining whether there is actually a defect or not is a valuable feature on its own, the ability to predict the actual root cause of a defect will provide even more valuable information to the development stakeholders. Root cause predictions help development teams triage and assign issues more efficiently, thereby decreasing the time spent diagnosing and transferring issues between teams. Classifying a defect as stemming from configuration rather than code logic can ensure that the ticket gets routed straight to the DevOps or infrastructure teams rather than mixed up with debugging for the developers. In such fast-paced development set-ups as working in agile sprints or CI/CD pipelines, such visibility into triaging will serve to lessen defect turnaround time and enhance team productivity. Organizations may look out for systemic problem areas across projects through aggregation of root cause predictions and thereby effect preventive measures at the architectural or process levels.

## Practical applications of the model

Proposed multi-output classification model finds widespread use in software engineering environments of today, especially along agile development pipelines, continuous integration workflows, and large-scale maintenance operations. By concurrently predicting whether a software report indicates a genuine bug or not and by also identifying its potential root cause, the model aids automated triage and intelligent assignment of issues. This reduction of manual workload for QA teams not only speeds up diagnosis but routing of issues as well; so, it fast-tracks the defect resolution process as a whole. In an organization where thousands of defect logs are managed across hundreds of applications, this model would facilitate the prioritization of high-impact issues, recognition of repeating failure patterns, whereas the secondary optimization of the workload would be achieved by routing issues to specialists most suitable for them. In turn, and as time goes by, the aggregation of predictions could feed

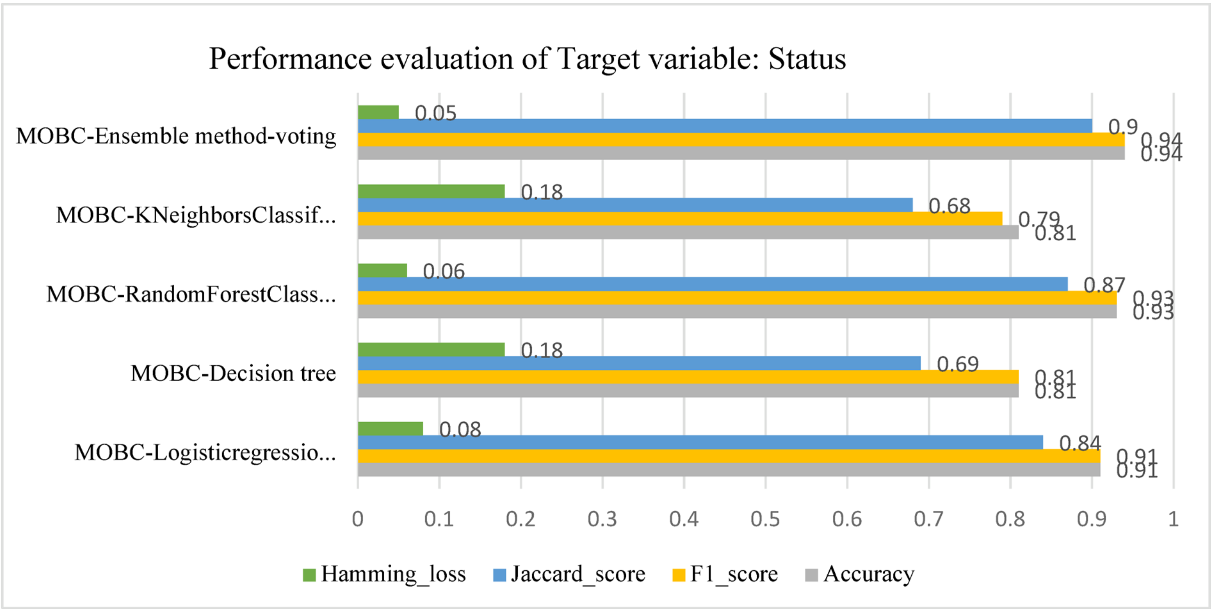


**Fig. 7**. Performance evaluation of Target variable: Status.

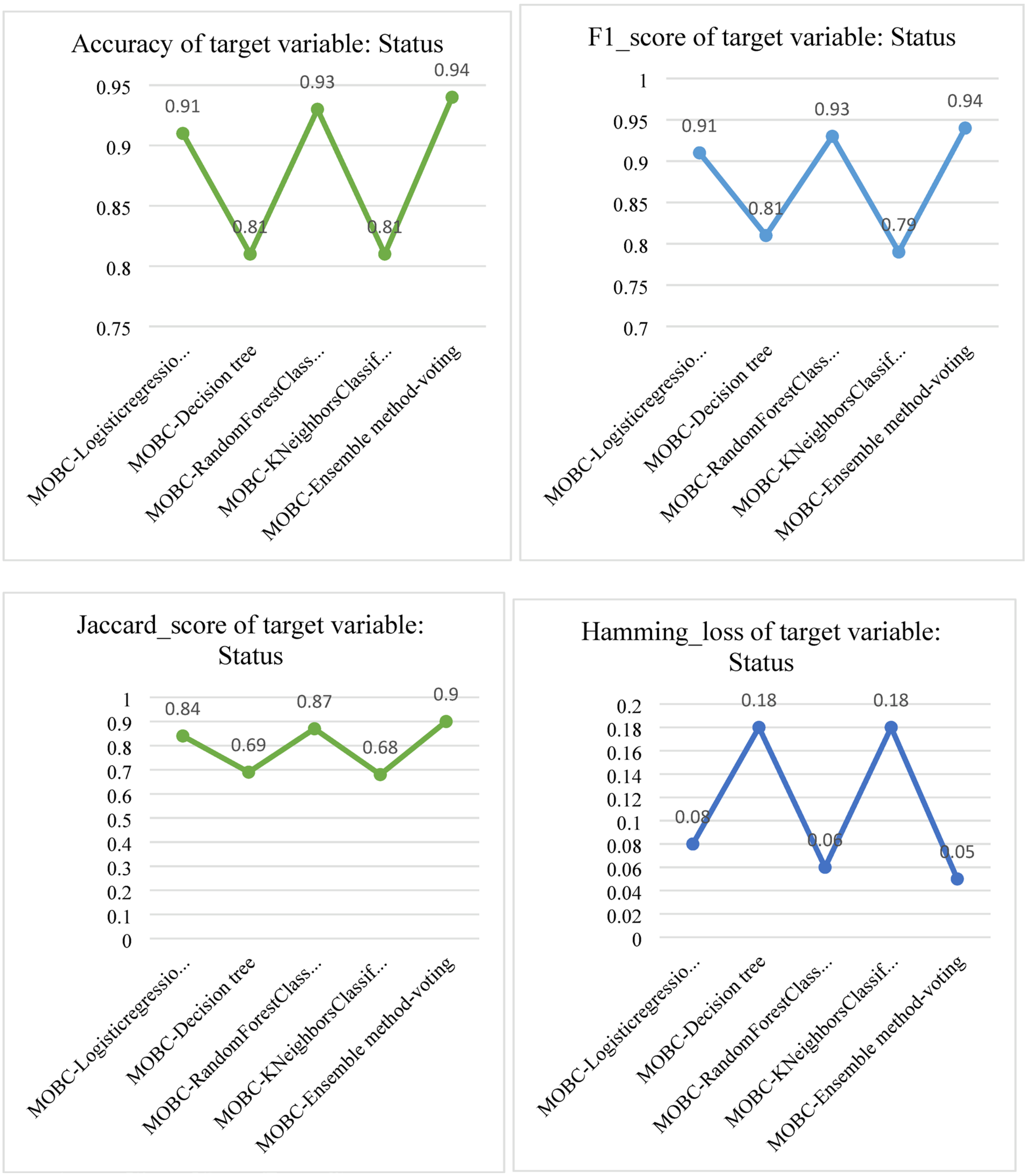


**Fig. 8.** Accuracy, F1_score, Jaccard_score and Hamming_loss of Target variable: Status.

| **Base classifier** | **Metric_type** | | | |
|---|---|---|---|---|
| | **Accuracy** | **F1_score** | **Jaccard_score** | **Hamming_loss** |
| Logisticregression | 0.94 | 0.94 | 0.89 | 0.05 |
| Decision tree | 0.9 | 0.9 | 0.82 | 0.09 |
| RandomForestClassifier | 0.97 | 0.97 | 0.95 | 0.02 |
| KNeighborsClassifier | 0.77 | 0.75 | 0.61 | 0.22 |
| Ensemble method-voting | 0.97 | 0.97 | 0.95 | 0.02 |

**Table 6.** Performance evaluation of target variable: Reason.

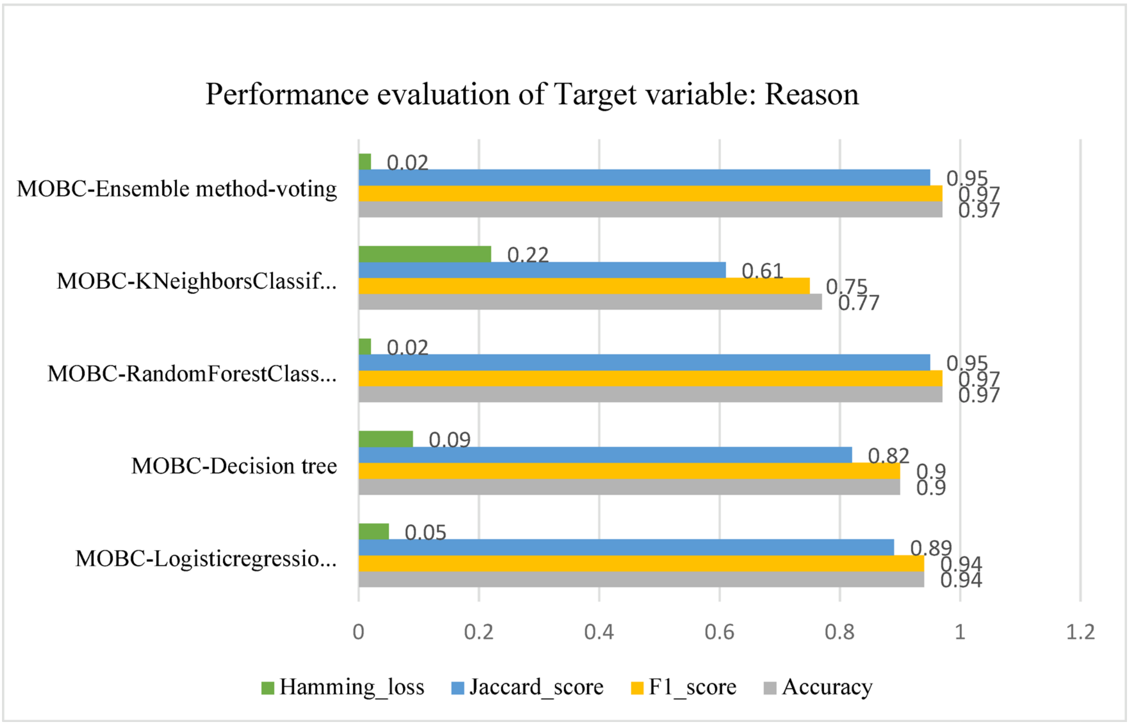


**Fig. 9**. Performance evaluation of Target variable: Reason.

software-quality audits, process improvements, and preventive measures, by exhibiting root cause trends. Thus, the model not only strengthens the operational capability of defect handling but also fosters long-term software quality and strategic decision-making.

In addition, the framework is extensible: adding additional output layers makes the model predict other important defect attributes, such as priority and severity. Priority prediction would enable informed scheduling, thereby enabling teams to attend first to the most urgent issues, while severity prediction would provide an objective quantification of the potential effect on system performance. These, in turn, will fund a holistic triage strategy that would decrease turnaround time and enhance the overall standard and stability of the software systems in question. When aggregated over time, such predictions will be able to pinpoint defect trends and systemic weaknesses that would require proactive quality assurance intervention.

## Comparative analysis with prior research

Three primary dataset types have been used in SDP research over the past ten years: (1) code metrics datasets, (2) defect log (bug report) datasets, and (3) process/other datasets (such as version-control or combined sources). Approximately 70–80% of defect prediction papers used static code metrics, while only about 20–30% used process metrics. On the other hand, very few recent studies use defect logs or bug report text as their main source of data. Traditional SDP heavily depends on structured code or process metrics. Rich textual descriptions found in defect log datasets—such as bug reports and issue tracker entries offer a deeper understanding of the signs, origins, and possible seriousness of defects. Since testers and developers frequently write these logs during real-world use and maintenance, they are extremely pertinent for severity classification, priority estimation, and root cause analysis. Therefore, adding defect log datasets makes it possible to create more context-aware, detailed, and useful defect prediction models that do more than just mark modules as flawed.

## Conclusion and future scope

By combining BERTopic, a transformer-based topic modeling technique, with a Multi-Output Classifier, this study offers a novel method for finding the software defects by concurrently predicting the existence of a defect and its underlying root cause. Our approach makes use of defect log datasets, which allows for a more contextually aware and semantically rich understanding of software faults than traditional models that only use static code or process metrics and provide binary classification. Based on evaluation metrics like accuracy, F1-score, Jaccard index, and Hamming loss, the experimental results show that ensemble-based classifiers—especially the voting method—perform better than individual base learners. This demonstrates how well multi-output learning and sophisticated NLP work in practical software defect analysis.

Future enhancements could incorporate additional output targets such as defect priority, severity, or estimated resolution time, enabling holistic support for triaging and workload optimization. Incorporating SHAP or LIME (Local Interpretable Model-Agnostic Explanations) would offer transparency into model decisions, increasing trust and usability for software teams. To ensure model robustness across domains, the framework could be evaluated on cross-project datasets using domain adaptation or transfer learning strategies to generalize to

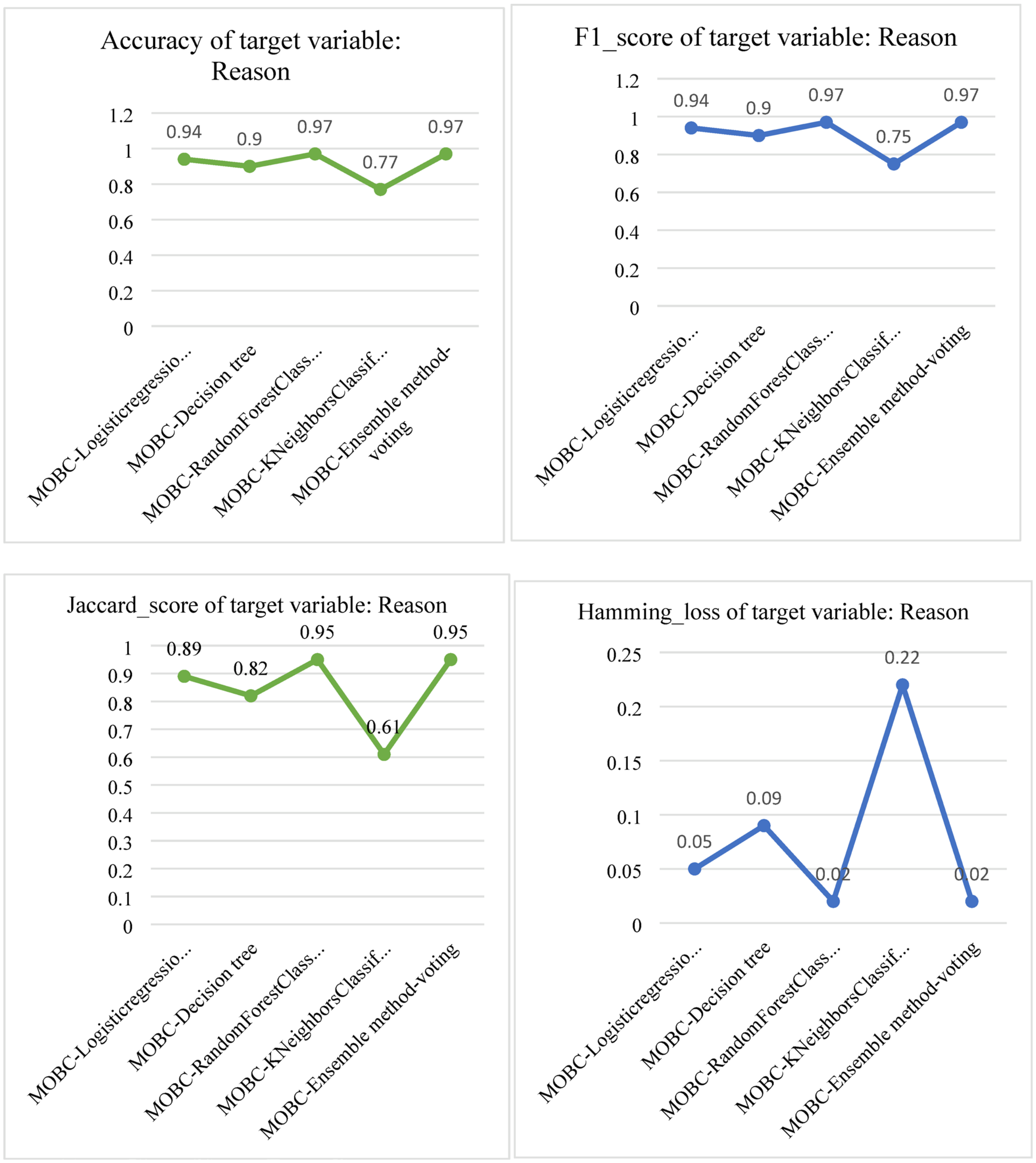


**Fig. 10**. Accuracy, F1_score, Jaccard_score and Hamming_loss of Target variable: Reason.

unfamiliar software systems. A promising application of this model lies in real-time integration with DevOps pipelines to provide automated feedback during pull request validations, enabling preventive quality assurance during development itself. Given the computational complexity of NLP-based models, future work can explore model pruning, knowledge distillation, or quantization to enable deployment on low-resource environments or edge devices. Longitudinal analysis of root cause predictions could help organizations discover recurring defect patterns and systemic architectural weaknesses, feeding into broader software process improvement cycles. The suggested method lays a solid basis for creating multifaceted, intelligent defect prediction tools that meet the needs of modern software engineering.

## Data availability

Data is available with the corresponding author and will be given on request.

## Code availability
The code is available from the corresponding author and can be given on request.

## Materials availability
Materials used in this research are available with corresponding author and given on request.

Received: 8 January 2025; Accepted: 10 July 2025
Published online: 14 July 2025

## References
1. Li, Z., Niu, J. & Jing, X. Y. Software defect prediction: future directions and challenges. *Automated Softw. Eng.* **31**(1), 19 (2024).
2. Hemández-Molinos, M.J., Sánchez-García, Á.J. and Barrientos-Martínez, R.E., 2021, October. Classification algorithms for software defect prediction: a systematic literature review. In 2021 9th International Conference in Software Engineering Research and Innovation (CONISOFT) (pp. 189-196). IEEE.
3. Thota, M. K., Francis, H., Shajin, P. & Rajesh Survey on software defect prediction techniques. *Int. J. Appl. Sci. Eng.* **17** (4), 331–344 (2020).
4. Khurana, D. et al. Natural language processing: state of the art, current trends and challenges. *Multimed Tools Appl.* **82**, 3713–3744. https://doi.org/10.1007/s11042-022-13428-4 (2023).
5. Hickman, L., Thapa, S., Tay, L., Cao, M. & Srinivasan, P. Text preprocessing for text mining in organizational research: review and recommendations. *Organizational Res. Methods*. **25** (1), 114–146 (2022). https://doi.org/10.1177/1094428120971683
6. Krzeszewska, U., Poniszewska-Marańda, A., & Ochelska-Mierzejewska, J. Systematic Comparison of Vectorization Methods in Classification Context. *Appl. Sci.* **12**(10), 5119. https://doi.org/10.3390/app12105119 (2022).
7. Sharma, T., Jatain, A., Bhaskar, S. & Pabreja, K. Ensemble machine learning paradigms in software defect prediction. *Procedia Comput. Sci.* **218**, 199–209 (2023). https://doi.org/10.1016/j.procs.2023.01.002
8. Giray, G., Bennin, K. E., Köksal, Ö., Babur, Ö. & Tekinerdogan, B. On the use of deep learning in software defect prediction. *J. Syst. Softw.* **195**, 111537 (2023). https://doi.org/10.1016/j.jss.2022.111537
9. Khan, M. A. et al. Software defect prediction using artificial neural networks: A systematic literature review. *Sci. Program.* **2022(1)**, 2117339. https://doi.org/10.1155/2022/2117339 (2022).
10. Khleel, N. A. A. & Nehéz, K. A new approach to software defect prediction based on convolutional neural network and bidirectional long short-term memory. *Prod. Syst. Inf. Eng.* **10**(3), 1–18 (2022).
11. Munir, H. S., Ren, S., Mustafa, M., Siddique, C. N. & Qayyum, S. Attention based GRU-LSTM for software defect prediction. *Plos one* **16**(3), e0247444 (2021).
12. Zhou, X. and Lu, L., 2020, December. Defect prediction via LSTM based on sequence and tree structure. In 2020 IEEE 20th International Conference on Software Quality, Reliability and Security (QRS) (pp. 366-373). IEEE.
13. =Zhang, Q. and Wu, B., 2020, June. Software defect prediction via transformer. In 2020 IEEE 4th Information Technology, Networking, Electronic and Automation Control Conference (ITNEC) (Vol. 1, pp. 874-879). IEEE.
14. Gomes, L., da Silva Torres, R. & Côrtes, M. L. BERT-and TF-IDF-based feature extraction for long-lived bug prediction in FLOSS: a comparative study. *Inf. Softw. Technol.* **160**, 107217 (2023). https://doi.org/10.1016/j.infsof.2023.107217
15. Alsaedi, S. A., Noaman, A. Y., Gad-Elrab, A. A. & Eassa, F. E. Nature-based prediction model of bug reports based on ensemble machine learning model. *IEEE Access.* **11**, 63916–63931. https://doi.org/10.1109/ACCESS.2023.3288156 (2023).
16. Zimmermann, J., Champagne, L. E., Dickens, J. M. & Hazen, B. T. Approaches to improve preprocessing for latent dirichlet allocation topic modeling. *Decis. Support Syst.* **185**, 114310 (2024). https://doi.org/10.1016/j.dss.2024.114310
17. Egger, R. & Yu, J. A topic modeling comparison between lda, nmf, top2vec, and bertopic to demystify twitter posts. *Front. sociol.* **7** 886498 (2022).
18. Siddiqui, T. & Mustaqeem, M. Performance evaluation of software defect prediction with NASA dataset using machine learning techniques. *Int. J. Inform. Technol.* **15** (8), 4131–4139 (2023). https://doi.org/10.1007/s41870-023-01528-9
19. Khleel, N. A. A. & Nehéz, K. Software defect prediction using a bidirectional LSTM network combined with oversampling techniques. *Cluster Comput.* **27**(3), 3615–3638 (2024).
20. Ferenc, R., Tóth, Z., Ladányi, G., Siket, I. & Gyimóthy, T. A public unified bug dataset for Java and its assessment regarding metrics and bug prediction. *Software Qual. J.* **28**, 1447–1506 (2020).
21. Bala, Y. Z., Samat, P. A., Sharif, K. Y. & Manshor, N. Improving cross-project software defect prediction method through transformation and feature selection approach. *IEEE Access* **11**, 2318–2326 (2022).
22. Bala, Y. Z., Samat, P. A., Sharif, K. Y. & Manshor, N. Cross-project software defect prediction through multiple learning. *Bull. Electr. Eng. Inf.* **13** (3), 2027–2035 (2024). https://doi.org/10.11591/eei.v13i3.5258
23. Zhu, H.N. and Rubio-González, C., 2023, May. On the reproducibility of software defect datasets. In 2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE) (pp. 2324-2335). IEEE.
24. Hai, T. et al. Cloud-based bug tracking software defects analysis using deep learning. *J. Cloud Comput.* **11** (1), 32 (2022). https://doi.org/10.1186/s13677-022-00311-8
25. Dolga, R., Zmigrod, R., Silva, R., Alamir, S. and Shah, S., 2023, December. Log Summarisation for Defect Evolution Analysis. In Proceedings of the 1st International Workshop on Software Defect Datasets (pp. 11-16).
26. Azam, M., Nouman, M. & Gill, A. R. Comparative Analysis of Machine Learning Technique to Improve Software Defect Prediction: Comparative Analysis of Machine Learning Technique to Improve Software Defect Prediction. KIET Journal of Computing and Information Sciences, 5(2), (2022).
27. Farid, A. B., Fathy, E. M., Eldin, A. S. & Abd-Elmegid, L. A. Software defect prediction using hybrid model (CBIL) of convolutional neural network (CNN) and bidirectional long short-term memory (Bi-LSTM). *PeerJ Comput. Sci.* **7**, e739 (2021). https://doi.org/10.7717/peerj-cs.739
28. Malhotra, R., Chawla, S. & Sharma, A. Software defect prediction using hybrid techniques: a systematic literature review. *Soft. Comput.* **27** (12), 8255–8288 (2023). https://doi.org/10.1007/s00500-022-07738-w
29. Agrawal, R. & Goyal, R. Developing bug severity prediction models using word2vec. *Int. J. Cogn. Comput. Eng.* **2**, 104–115 (2021). https://doi.org/10.1016/j.ijcce.2021.08.001
30. Dao, A. H. & Yang, C. Z. Severity prediction for bug reports using multi-aspect features: a deep learning approach. *Mathematics* **9** (14), 1644. https://doi.org/10.3390/math9141644 (2021).
31. Ali, A., Xia, Y., Umer, Q. & Osman, M. BERT based severity prediction of bug reports for the maintenance of mobile applications. *J. Syst. Softw.* **208**, 111898 (2024). https://doi.org/10.1016/j.jss.2023.111898
32. Siddiq, M.L. and Santos, J.C., 2022, May. Bert-based github issue report classification. In Proceedings of the 1st international workshop on natural language-based software engineering (pp. 33-36).
33. Laiq, M., bin Ali, N., Börstler, J. & Engström, E. A data-driven approach for understanding invalid bug reports: An industrial case study. *Inf. Soft. Technol.* **164**, 107305. (2023).

34. Chen, T.H., Thomas, S.W., Nagappan, M. and Hassan, A.E., 2012, June. Explaining software defects using topic models. In 2012 9th IEEE working conference on mining software repositories (MSR) (pp. 189-198). IEEE.
35. Liu, X., Yin, Y., Li, H., Chen, J., Liu, C., Wang, S. and Yin, R., 2021. Intelligent radar software defect classification approach based on the latent Dirichlet allocation topic model. EURASIP Journal on Advances in Signal Processing, 2021(1), p.44.
36. Li, H., Chen, T. H., Shang, W. & Hassan, A. E. Studying software logging using topic models. *Empir. Softw. Eng.* **23**(5), 2655–2694 (2018).
37. Wenjuan Bu, H., Shu, F., Kang, Q., Zhao, Y. & Hu and Software subclassification based on BERTopic-BERT-BiLSTM model. *Electron. 2023.* **12**, 3798 (2023). https://doi.org/10.3390/electronics12183798
38. Pachouly, J., Ahirrao, S., Kotecha, K., Kulkarni, A. & Alfarhood, S. Multilabel classification for defect prediction in software engineering. *Sci. Rep.* **15** (1), 8739 (2025). https://doi.org/10.1038/s41598-025-93242-8
39. Madaraboina, S. N., Sharma, S., Singh, S. & Kumar, V. Efficient multi-target classification for bug priority and resolution time prediction. Multimedia Tools and Applications, pp.1-30 (2024).
40. Grattan, N., da Costa, D. A. & Stanger, N., The need for more informative defect prediction: A systematic literature review. *Inf Soft. Technol.* **171**, 107456 (2024).
41. Panda, R. R. & Nagwani, N. K. Software bug priority prediction technique based on intuitionistic fuzzy representation and class imbalance learning. *Knowl. Inf. Syst.* **66** (3), 2135–2164 (2024). https://doi.org/10.1007/s10115-023-02000-7
42. Gokcimen, T. and Das, B., 2024, April. Topic modelling using bertopic for robust spam detection. In 2024 12th International Symposium on Digital Forensics and Security (ISDFS) (pp. 1-5). IEEE.
43. Reimers, N. & Gurevych, I., 2019. Sentence-bert: Sentence embeddings using siamese bert-networks. arXiv preprint arXiv:1908.10084.
44. McInnes, L., Healy, J. and Melville, J., 2018. Uniform manifold approximation and projection for dimension reduction. arXiv. Preprint] doi, 10.
45. Trozzi, F., Wang, X. & Tao, P. UMAP as a dimensionality reduction tool for molecular dynamics simulations of biomacromolecules: A comparison study. *J. Phys. Chem. B.* **125** (19), 5022–5034. https://doi.org/10.1021/acs.jpcb.1c02081 (2021).
46. Stewart, G. & Al-Khassaweneh, M. An implementation of the HDBSCAN* clustering algorithm. Appl. Sci. 12(5), 2405 (2022).
47. Dipa, W.A. and Sunindyo, W.D., 2021, November. Software defect prediction using smote and artificial neural network. In 2021 International Conference on Data and Software Engineering (ICoDSE) (pp. 1-4). IEEE.
48. Husain, G. et al. SMOTE vs. SMOTEENN: A study on the performance of resampling algorithms for addressing class imbalance in regression models. *Algorithms* **18** (1), 37 (2025).

## Author contributions
All authors contributed to the study's conception and design. Material preparation, data collection, and analysis were performed By Devi Priya Gottumukkala, Prasad Reddy P V G D , and S. Krishna Rao. The first draft of the manuscript was written by Devi Priya Gottumukkala all authors commented on previous versions of the manuscript. All authors read and approved the final manuscript.

## Declarations

## Consent for publication
The authors give consent for their publication.

## Competing interests
The authors declare no competing interests.

## Additional information
**Correspondence** and requests for materials should be addressed to D.P.G.