Computers, Materials & Continua

Tech Science Press

Doi:10.32604/cmc.2025.057792

ARTICLE

Amalgamation of Classical and Large Language Models for Duplicate Bug Detection: A Comparative Study

Sai Venkata Akhil Ammu[1], **Sukhjit Singh Sehra**[1,*], **Sumeet Kaur Sehra**[2] **and Jaiteg Singh**[3]

[1]Department of Physics and Computer Science, Wilfrid Laurier University, Waterloo, N2L 3C5, Canada
[2]Appilied Computer Science and Information Technology, Conestoga College, Waterloo, N2J 2W2, Canada
[3]Chitkara University Institute of Engineering and Technology, Chitkara University, Punjab, 140401, India

*Corresponding Author: Sukhjit Singh Sehra. Email: ssehra@wlu.ca

Received: 27 August 2024; Accepted: 13 January 2025; Published: 26 March 2025

ABSTRACT: Duplicate bug reporting is a critical problem in the software repositories' mining area. Duplicate bug reports can lead to redundant efforts, wasted resources, and delayed software releases. Thus, their accurate identification is essential for streamlining the bug triage process mining area. Several researchers have explored classical information retrieval, natural language processing, text and data mining, and machine learning approaches. The emergence of large language models (LLMs) (ChatGPT and Huggingface) has presented a new line of models for semantic textual similarity (STS). Although LLMs have shown remarkable advancements, there remains a need for longitudinal studies to determine whether performance improvements are due to the scale of the models or the unique embeddings they produce compared to classical encoding models. This study systematically investigates this issue by comparing classical word embedding techniques against LLM-based embeddings for duplicate bug detection. In this study, we have proposed an amalgamation of models to detect duplicate bug reports using textual and non-textual information about bug reports. The empirical evaluation has been performed on the open-source datasets and evaluated based on established metrics using the mean reciprocal rank (MRR), mean average precision (MAP), and recall rate. The experimental results have shown that combined LLMs can outperform (recall-rate@k = 68%–74%) other individual models for duplicate bug detection. These findings highlight the effectiveness of amalgamating multiple techniques in improving the duplicate bug report detection accuracy.

KEYWORDS: Duplicate bug detection; large language models; information retrieval

1 Introduction

Duplicate bug reporting is a significant challenge in software development, leading to wasted resources and delayed resolution times. Numerous bug reports are generated in the software industry, and manual processing is time-consuming and error-prone, resulting in development delays [1,2]. Software bug reports are essential for developing and maintaining software [3,4]. Bug reports in the bug repository are classified to facilitate bug triage and assignment of bug-fixing. Identifying and handling duplicate bug reports can be time-consuming for developers, QA personnel, and triggers [1]. Identifying duplicate bug reports is vital for reducing triaging efforts. The "component" attribute in bug reports identifies the bug's root cause and the responsible developers [1]. However, it can be challenging due to poorly written and ambiguous text. Extensively researched automated techniques have been developed to tackle this issue [3,5–7]. Despite their potential, these approaches encounter challenges that affect their reliability and scalability in practical applications.

Natural language processing (NLP) techniques have demonstrated significant potential in tackling various language-related challenges [5]; however, they often lack interpretability, and the robustness required for industry-level applications. Numerous methods have been implemented to detect duplicate bug reports [4,8–12]. An increase of 200% in the count of detected bug reports has been reported by [13]. Numerous methods have been suggested for automating the identification of duplicate bug reports, including NLP [14], machine learning [3,15,16], information retrieval (IR) [17], topic analysis [6,15], deep learning (DL) [13,18–22], and hybrid models [11,18].

Recent advancements in DL demonstrate that substantial data are essential to achieve high precision with DL models [23]. In contrast, Neysiani et al. [24] reported that machine learning models had better validation performance (up to 40% higher) than IR-based approaches for duplicate bug report detection on an Android dataset. Jiang et al. [6] found that DL methods alone did not outperform IR-based methods for ranking duplicate bug reports. To address the limitations of both approaches, some researchers have proposed combining IR and DL techniques to compute textual similarity comprehensively. Jiang et al. [6] proposed a method combining IR and DL techniques, improving the mean average precision (MAP) metric by up to 28.97% compared to classic IR-based methods. Fang et al. [25] introduced RepresentThemAll, a pre-trained approach that learns a universal representation of bug reports and can handle multiple downstream tasks, including detecting duplicate bug reports. However, these approaches often fail to capture nuanced semantic similarities or integrate textual and non-textual features effectively, leading to suboptimal results. The advent of large language models (LLMs) introduces new possibilities for enhancing duplicate bug detection. LLMs excel in capturing semantic relationships and understanding complex textual data, addressing some limitations of traditional IR and DL approaches [4]. However, their integration with classical models for effective bug report analysis remains underexplored.

This study aims to streamline the identification and management of duplicate bug reports, which can otherwise increase developer workload and delay the resolution of critical issues.

To mitigate these consequences, this study aims to evaluate the effectiveness of contemporary IR models, semantic textual similarity (STS) models, and LLMs (ChatGPT and Huggingface) in identifying duplicate bug reports. This study combines sparse and dense vector representations to derive amalgamated models. The amalgamated approach combines the strengths of classical and LLM-based methods for improved duplicate bug detection. Classical methods like term frequency-inverse document frequency (TF-IDF) excel at keyword-based matching, while LLMs provide deep semantic analysis of complex text. The model balances precision and recall by integrating both, leveraging classical efficiency and LLMs' contextual understanding. A heuristic ranking system harmonizes outputs, enhancing accuracy and reducing false results. Traditional text similarity models (global vectors for word representation (GloVe), TF-IDF, and Word2Vec) are combined with sentence-BERT (SBERT) and LLMs to create amalgamated similarity scores. Using this score, the bug triaging team can see duplicate (most similar) bug reports. The proposed models consider bug reports' textual and non-textual features.

A composite score is calculated using similarity scores from individual approaches to identify the top k duplicate bug recommendations. Three datasets (Eclipse, Apache, and KDE) have been used for empirical evaluation. Table 1 describes the datasets in detail. Three established performance metrics evaluate the effectiveness of the proposed approach: mean reciprocal rank (MRR), MAP, and recall rate@k. This study aims to investigate and provide contributions to the following research questions.

- RQ1: Which factors influence the ability to detect duplicates in bug reports, and what are the most effective methods for identification?
- RQ2: How can the reliability of similarity scores be enhanced when using semantic search and LLMs for duplicate detection, considering their limited context awareness?

- RQ3: How effective is amalgamating different models to rank duplicate bug reports and improve duplicate bug report detection?
- RQ4: What are the proposed models' statistical significance and effect size in the context of duplicate bug report detection?

Table 1: Dataset description [26]

Project	Apache	Eclipse	KDE
Distinct id	2416	31,811	26,114
# of reports	44,049	503,935	365,893
# of products	35	232	584
# of components	350	1486	2054
Min report opendate	2000-08-26	2001-02-07	1999-01-21
Max report opendate	2017-02-10	2017-02-07	2017-02-13
Max number of rediscoveries	19	128	405
Non-discovered reports (%)	86	83	70

The remainder of the study has been organized as follows. The next section provides a detailed overview of related work. The third section elaborates on the methodology and provides information on the evaluation metrics. The fourth section explains the dataset and steps followed in pre-processing. The findings from the proposed model have been discussed in the fifth section. Potential threats to validity are addressed in the sixth section, and the final section concludes the paper while outlining directions for future work.

2 Related Work

Bocu et al. [27] reviewed artificial intelligence (AI)-driven bug-triaging methods to improve automatic bug management and address conceptual and practical challenges and potential research directions. Extensive research has been conducted to detect duplicate bug reports automatically [28–32]. LLMs are revolutionizing natural language comprehension, marking a significant step towards artificial general intelligence [33,34]. Transformer language models, like the top-rated ones, have revolutionized NLP since their introduction in 2018 [35]. However, due to hardware constraints, these models often rely on few-shot or zero-shot learning [35]. LLMs and vision-language models have recently been used to identify affordances and constraints, generating trajectories for manipulation tasks [36]. LangNav [37] has used LLMs for navigation by synthesizing data with GPT-4 [38] and fine-tuning with Llama2 [39] as the core framework.

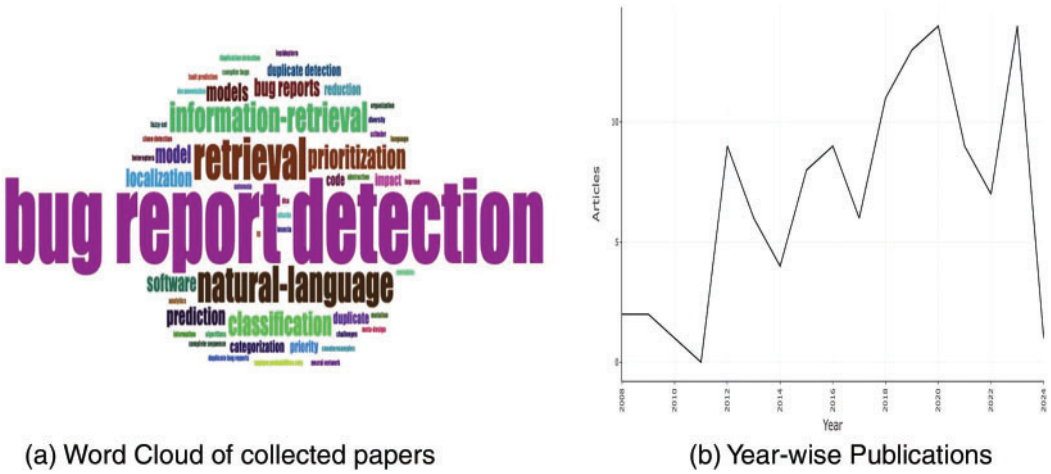
Literature analysis has revealed various bug detection methods, including IR techniques (TF-IDF weighting), cosine similarity), NLP (text embedding (Word2Vec, GloVe), topic modelling, named entity recognition (NER) and natural language independence models, hybrid approaches, clustering and classification methods (k-means clustering, hierarchical clustering, and support vector machines (SVM)), DL techniques (long short-term memory (LSTM), deep neural networks (DNNs), and convolutional neural networks (CNNs)). The traditional TF-IDF approach represents a bug report as a vector to calculate the similarity of textual features [40,41]. Duplicate detection has been implemented using an n-grams-based approach [42]. The study by Jalbert and Weimer [11] combined the duplicate reports' textual and non-textual features. Further, Wang et al. combined the execution traces and textual information [8]. Word embedding is a prevalent approach to represent document vocabulary using word lists and software dictionaries to capture each issue report's implicit context [15]. Further, Latent Dirichlet Allocation (LDA) has demonstrated excellent potential for duplicate bug report identification [7]. Zou et al. [1] have suggested that LDA with

the n-gram algorithm is more efficient than the state-of-the-art methods. A DL technique for duplicate bug reports has been proposed by Budhiraja et al. [18].

Classical models such as Word2Vec [42] and GLoVe [43] pioneered word embeddings, capturing semantic relationships through vector representations. These early methods were foundational but limited to word-level embeddings. Since the introduction of transformer-based architectures such as BERT [44] and RoBERTa [45], the field has expanded to include embeddings for longer text sequences such as sentences and documents. Fine-tuned sentence encoders SBERT [46] further refined this approach, providing robust contextual embeddings for various NLP tasks. The emergence of LLMs, including GPT [38] and LLaMA [39], has dramatically increased the size and complexity of embedding models. These models produce high-quality embeddings and excel across numerous NLP benchmark classical models. Despite their success, it's uncertain whether the improved performance is solely due to their larger scale or fundamentally distinct embeddings.

In a recent study, Zhang et al. [19] proposed CUPID, combining the traditional duplicate bug detection approach REP with the advanced LLM (ChatGPT) under the zero-shot setting to extract essential information from bug reports. While LLMs like PaLM and ADA perform better on word analogy tasks, they often yield results like classical models such as SBERT [46]. This indicates that smaller, more resource-efficient models can still be highly effective. Common challenges in identifying duplicate bug reports include vocabulary mismatches, low report quality, managing structured and unstructured data, and scalability issues in large software projects. The current study aims to test and refine our understanding of how classical embeddings and LLM-based embeddings differ and are similar. It has also presented a recent trend in amalgamating the contextual, statistical, and semantic models for identifying duplicate bug reports.

The study accessed the Web of Science to collect articles through a search for "Duplicate" AND "Bug" OR "Detect*". The open-source tool JabRef was used to collect, screen, select, and prepare the corpus. The literature was manually reviewed to identify the articles that satisfied the required criteria. Eliminating duplicate articles or those out of focus resulted in 116 articles in the final literature dataset. Fig. 1a displays the word cloud from the literature abstracts, exported to a CSV file using JabRef. Fig. 1b shows the publication trend for bug detection. Fig. 1c presents the top ten authors in duplicate bug detection from 2008-2024, and Fig. 1d displays the top ten publishing venues based on dataset occurrences.



(a) Word Cloud of collected papers (b) Year-wise Publications

Figure 1: (Continued)

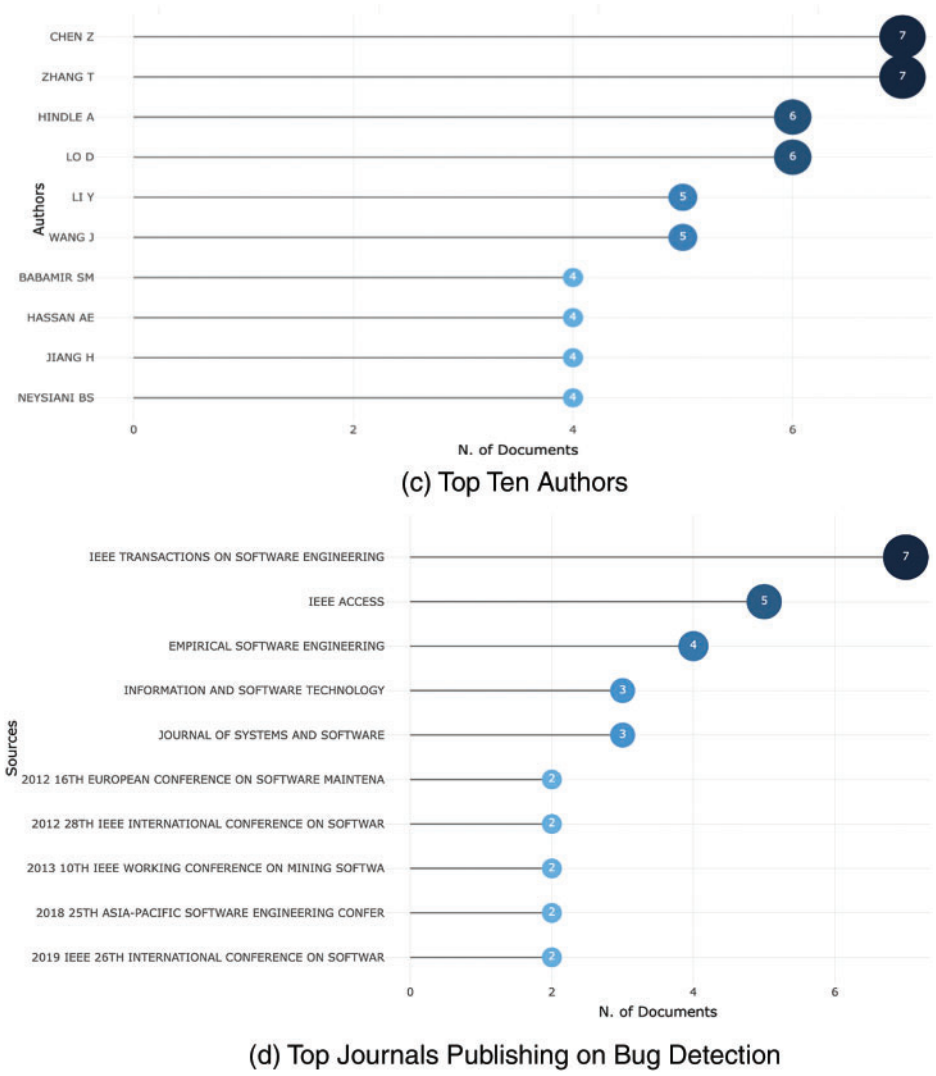


Figure 1: Analysis of literature

3 Methodology

After thorough data preparation, semantic search employs query meaning to retrieve relevant documents from a collection effectively. The existing bug reports act as a "database," and the new bug report as a "query." Detecting duplicate bugs involves searching the database for reports that match the query's semantic content by embedding the query and bug reports to find the closest matches. This method uses vector representations of text for mathematical comparison, comparing embedding vectors for semantic matches rather than keywords. This study compares traditional approaches with the most recent LLMs and amalgamated approaches for duplicate bug detection. The flowchart in Fig. 2 shows the study approach.

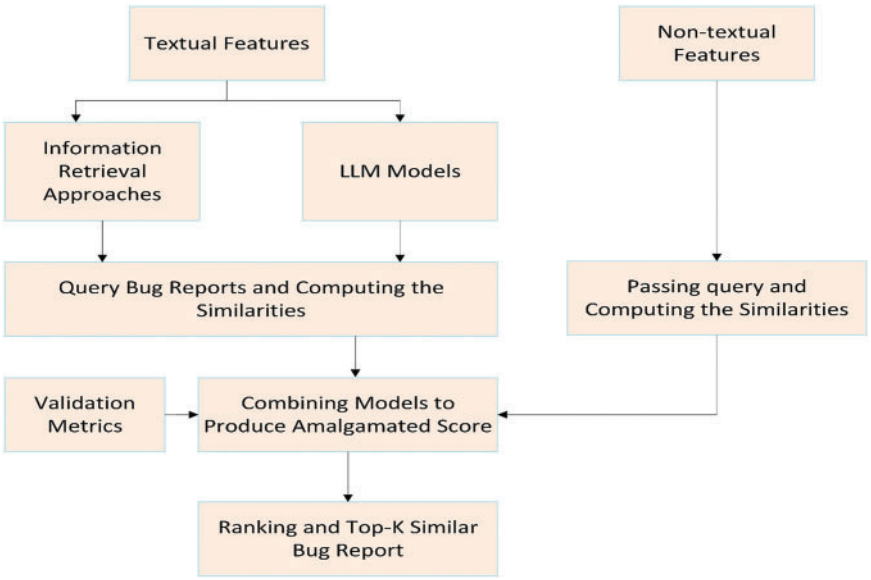


Figure 2: Flow of the duplicate bug detection system

3.1 Term Frequency-Inverse Document Frequency

TF-IDF is a method for determining a term's significance within a document collection [47,48]. It evaluates a term's significance based on its frequency within a document relative to its frequency across the entire collection. The TF measures how often a term appears in a document, while the IDF measures how unique a term is across the entire collection (Eq. (1)) [1].

$$IDF = \frac{\log(1 + n_d)}{\left(1 + df_{i,j}\right) + 1} \tag{1}$$

It is also a pre-trained two-layer neural network. There are two types of Word2Vec models: continuous bag-of-words (CBOW) and skip-gram. Both models demonstrate how a word interacts with surrounding words differently. To find a target word's probability, the softmax function is applied to a predicted word vector $\hat{r}$ and a target word vector w_t, as given in Eq. (2) [1].

$$P\left(w_t \middle| \hat{r}\right) = \frac{exp(w_t, \hat{r})}{\sum_{w \in W} \exp(w', r')} \tag{2}$$

The Gensim implementation of the Word2Vec (skip-gram) model was used. It was pre-trained on the Google News corpus, which consists of 3 billion running words. The model includes 3 million 300-dimensional English word vectors. Here, W represents all target word vectors set and $exp(w_t, \hat{r})$ calculates the target word w_t,s compatibility with the context $\hat{r}$.

3.2 Global Vectors for Word Representation

An unsupervised learning algorithm, GloVe combines global matrix factorization with local context window methods [49]. The Google News pretrained model was used to reduce the error between the word

embedding vectors and the co-occurrence probability log's dot product. It is represented in Eq. (3), where w and $\tilde{w}$ are word vectors.

$$F\left(w_i, w_i, \tilde{w}\right) = \frac{P_{ik}}{P_{jk}} \tag{3}$$

where i, j, and k are three words, and P_{ik}/P_{jk} depends upon them.

3.3 Large Language Models

A language model is a mathematical model that assigns probabilities to strings from a language's vocabulary. It uses algorithms to analyze text, learn a language, and predict words in a sentence. Advanced language models are highly effective for NLP tasks, leveraging transformer architectures for rich word embeddings. Among the top models, SBERT [46] is known for generating sentence-level embeddings by fine-tuning BERT with a Siamese network structure, making it ideal for semantic similarity, clustering, and retrieval tasks. Voyage-large-2-instruct ranks at the top of the Massive Text Embedding Benchmark (MTEB) [50] for its instruction-tuned capabilities, outperforming models from OpenAI and Cohere in critical tasks such as retrieval, classification, and reranking. GPT-4, developed by OpenAI, excels in STS due to its ability to handle complex sentence structures and long-range dependencies, making it versatile for various NLP tasks beyond similarity measures. RoBERTa, an optimized version of BERT, improves performance in many STS benchmarks due to its robust training approach [45]. DistilBERT [51] is a lighter version of BERT, suitable for real-time applications where resource constraints are a concern.

These models, available on platforms like Hugging Face [52], provide powerful tools for high-quality text embeddings and accurate semantic similarity measurements. For this manuscript, the LLM models, OpenAI, SBERT, and Voyage AI, were chosen based on the literature and as the top-performing models of the MTEB for the STS task [50].

3.4 Proposed Amalgamated Model

Established models like NextBug [53] often rely on a single approach, limiting their effectiveness in complex bug detection. This study addresses this by integrating LLMs with classical machine learning techniques, combining their strengths to improve prediction accuracy and robustness. The flow of the entire system is illustrated in Fig. 2 and implementation details have been provided in the Git repository[1] for reproducibility. The similarity scores vector (S_1, S_2, S_3, S_4, etc., from individual models) for the k most similar bug reports is obtained from each approach. A heuristic ranking method combines these and creates a universally ranked set of results. The ranking method adjusts the weights of each item in the similarity scores vector based on its position, following Eq. (4).

$$R_i = \frac{1}{Position_i} \tag{4}$$

After obtaining the ranks for each bug report from each selected model, the combined score is generated by summing these ranks (Eq. (5)) as discussed in [1]. The ranks are zero for non-contributing models. This results in a vector with elements less than or equal to nk, where k represents the number of duplicate bug reports returned from each model, and n is the number of combined models.

$$S = \left(R_1 + R_2 + R_3 + R_4 + \ldots\right) * PC \tag{5}$$

In Eq. (5), S represents each returned bug report's combined score (rank), with $R_1, R_2, R_3, R_4, \cdots$ denoting the ranks obtained from different models, respectively, as defined in Eq. (4). PC is the product & component

score, acting as a filter. For example, if two bug reports pertain to the same product and component, their similarity depends on the document similarity score. However, if they belong to different products and components, they are unlikely to be similar, even if their document similarity score is high, resulting in a score of zero.

For example, a bug report, "Null pointer exception when attempting to load configuration file," undergoes pre-processing, including lowercasing, tokenization, and lemmatization. Feature embeddings are then generated using methods like TF-IDF, which weights terms by domain relevance, Word2Vec for semantic relationships, and SBERT for sentence-level encoding. Similarity scores are computed for a query report (e.g., "Error when loading config file") with TF-IDF, SBERT, and a weighted combination yielding scores of 0.76, 0.84, and 0.82, respectively. The amalgamation step aggregates these scores using a heuristic ranking method, incorporating product and component filters to ensure relevance, assigning a score of 0 to mismatched components.

3.5 Evaluation Metrics

The established metrics [1], including recall-rate@k (RR@k), MAP, and MRR, have been used to assess the models, which measure how effectively the system retrieves relevant duplicate reports. These metrics are commonly used in recommendation systems to address software engineering tasks [6].

1) *Recall-rate@k:* RR@k is a metric used to evaluate the effectiveness of a recommendation system by measuring how many of the top k recommended items are relevant. For a given query q, RR@k is defined as follows:

$$RR\,(q) = 1, if\, S\,(q) \cap R\,(q) \neq 0, = 0, otherwise \tag{6}$$

where, $S(q)$ represents the ground truth set of relevant bug reports for the query q, and $R(q)$ represents the set of top-k recommendations the system returns. If there is at least one relevant bug report in the top-k recommendations, the RR@k is 1; otherwise, it is 0.

2) *Mean Average Precision:* MAP evaluates the precision of a recommendation system across all queries and provides an overall measure of the system's accuracy by considering the order of the recommended items. It is defined as the mean of the average precision (*Avg P*) values for all queries, as shown below:

$$MAP = \sum_{q=1}^{|Q|} \frac{AvgP(q)}{|Q|} \tag{7}$$

where, Q represents the total number of queries in the test set. A single query's average precision (*Avg P*) is calculated by averaging the precision at each rank position where a relevant item is retrieved. The average precision for a query q is calculated as:

$$Avg\,P\,(q) = \frac{P_{nk=1}\,(P\,(k)\,.\,rel(k))}{number\,of\,relevant\,documents} \tag{8}$$

where $P(k)$ is the precision at rank k, $rel(k)$ is a binary indicator function that is 1 if the item at rank k is relevant and 0 otherwise, and n is the total number of items in the recommendation list.

3) *Mean Reciprocal Rank (MRR):* MRR focuses on the first relevant item's ranking position in the recommendation list and is calculated as the average of the reciprocal ranks of the first relevant item for all queries, as defined below:

$$MRR = \frac{1}{|Q|} \sum_{i=1}^{|Q|} \frac{1}{rank_i} \tag{9}$$

Q represents the total number of queries, and $rank_i$ is the rank position of the first relevant item for the i-th query. The reciprocal rank for a query q, denoted as $ReciprocalRank(q)$, is calculated as:

$$Reciprocal\ Rank\ (q) = \frac{1}{index_q} \tag{10}$$

Here, $index_q$ represents the first relevant bug report position for the query q.

4 Dataset and Pre-Processing

4.1 Dataset

Sadat et al. [26] curated a collection of bug reports for research purposes, which has been widely utilized in numerous studies [1]. The repository contains three defect datasets in CSV format from Bugzilla, covering Apache, Eclipse, and KDE open-source software projects. The datasets include 914,000 defect reports from 1999 to 2017, highlighting the intricate relationships between duplicate defects. Table 1 presents descriptive statistics for the datasets. The textual information refers to the bug's user description, which is represented by the "Short desc" field. Non-textual features, in contrast, include attributes that are not based on free-text input. For example, the "Product" field specifies the software product affected by the defect (e.g., Apache, Eclipse), while the "Component" field identifies the specific module or feature of the software where the bug was found (e.g., "UI," "Database"). Other non-textual features include "Bug severity" (e.g., "Critical," "Minor"), "Priority" (e.g., "High," "Low"), "Bug status" (e.g., "New," "Resolved"), "Version" (e.g., "v1.0," "v2.3"), "Current status" (e.g., "Open," "Closed"), and "Duplicate list" (which identifies the list of related bug reports that are considered duplicates). These non-textual features are essential for understanding the context of each bug report and are used for tasks such as filtering or categorizing defects. For instance, the "Product" and "Component" fields are commonly used as filters in duplicate detection algorithms, while the "Duplicate list" feature is used as the ground truth for evaluating the performance of duplicate detection techniques.

4.2 Pre-processing

The corpus's textual features were prepared using pre-processing and term filtering. Subsequently, the identified sentences, words, and characters were converted into tokens to create a corpus. This process involved converting everything into lowercase, normalizing words, removing punctuation characters, and performing lemmatization.

5 Results

We used a high-performance computing machine with 64 GB of RAM and a 1 TB hard disk to evaluate the models. Our approach compares new bug reports to a database of resolved reports to find similar cases. The algorithms were developed using Python, primarily relying on standard library packages and LangChain to use LLM models (from OpenAI and Huggingface). We tested values of k = 1, 10, 20, 30, 40, and 50 to measure improvements in accuracy and efficiency when identifying duplicate bug reports. This study rigorously tested the proposed models against a suite of established approaches for recommending duplicate bug reports. The models were applied to a publicly accessible bug report dataset comprising three distinct subsets. Each dataset was partitioned into training and testing sets. The test set consists of bug reports labelled with ground truths about duplicates; specifically, bug reports in the test dataset either contain a non-empty list of duplicates or are marked as 'NA' (indicating no duplicates).

Using data from open-source software (OSS) bugs in the real world adds practical relevance to our study, allowing us to test models under realistic conditions that software developers might encounter. In the OSS dataset, one column contained the duplicate bug list for validating evaluation parameters.

The Apache, Eclipse, and KDE project test datasets contained 2518, 34,316, and 30,377 bug reports, respectively. We divided each project dataset into 80% for training and 20% for testing. The training dataset converted the existing textual information into the vector representation for the models. The test data detected duplicate bug reports from the training dataset, which were considered resolved.

5.1 Empirical Analysis

The proposed ensemble model has been empirically analyzed on OSS data sets. The models analyze the text in the training dataset to build a vocabulary for detecting duplicate test bug reports.

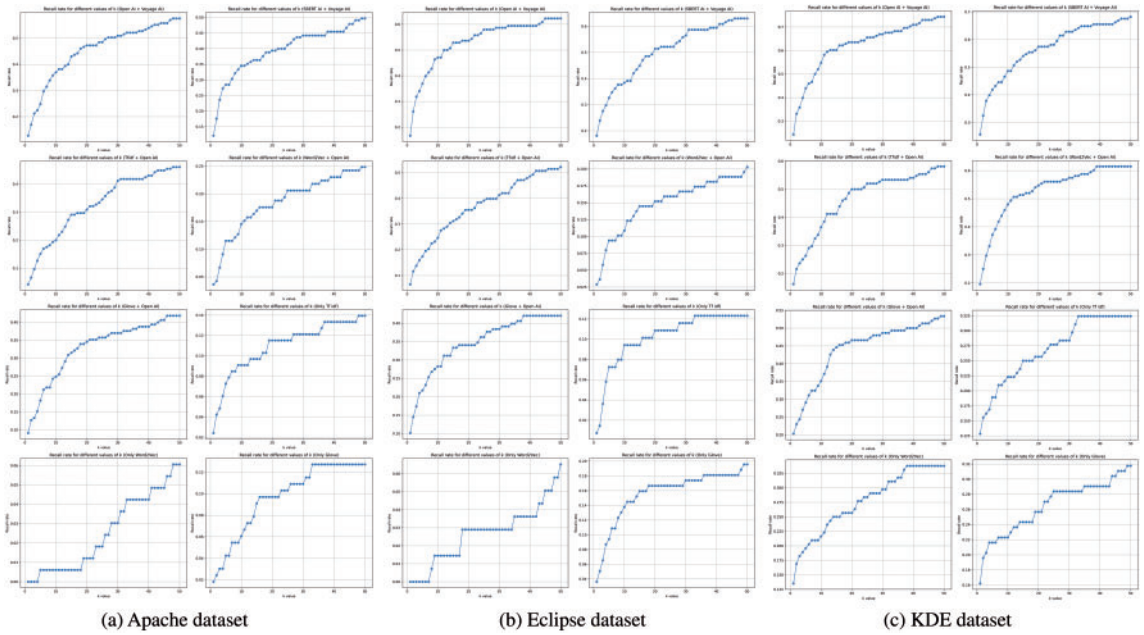
1) *Apache dataset:* The Apache dataset is the smallest of the three datasets, containing 44,049 bug reports generated for 35 products and 350 components.
2) *Eclipse dataset*: The Eclipse dataset contains 503,935 bug reports and 31,811 distinct IDs, including 232 products and 1486 component bug reports. Due to its size, a random sample of 10% of the full dataset was taken.
3) *KDE dataset:* This dataset comprises 365,893 bug reports for 584 products, of which 2054 were used. A random sample representing 10% of the data was selected due to its large size.

For empirical validation, we compared the performance of different amalgamated approaches with individual established approaches in detecting duplicate bug reports. Various amalgamated models can be created from selected models, but this section focuses on amalgamating only the top-performing models.

Fig. 3a–c illustrates the RR@k for amalgamated and established individual approaches, with varying values of the parameter k for Apache, Eclipse, and KDE datasets, respectively. The recall rate analysis across the Apache, Eclipse, and KDE datasets shows that OpenAI + Voyage AI and SBERT AI + Voyage AI consistently achieve the highest recall rates, stabilizing around 0.74.

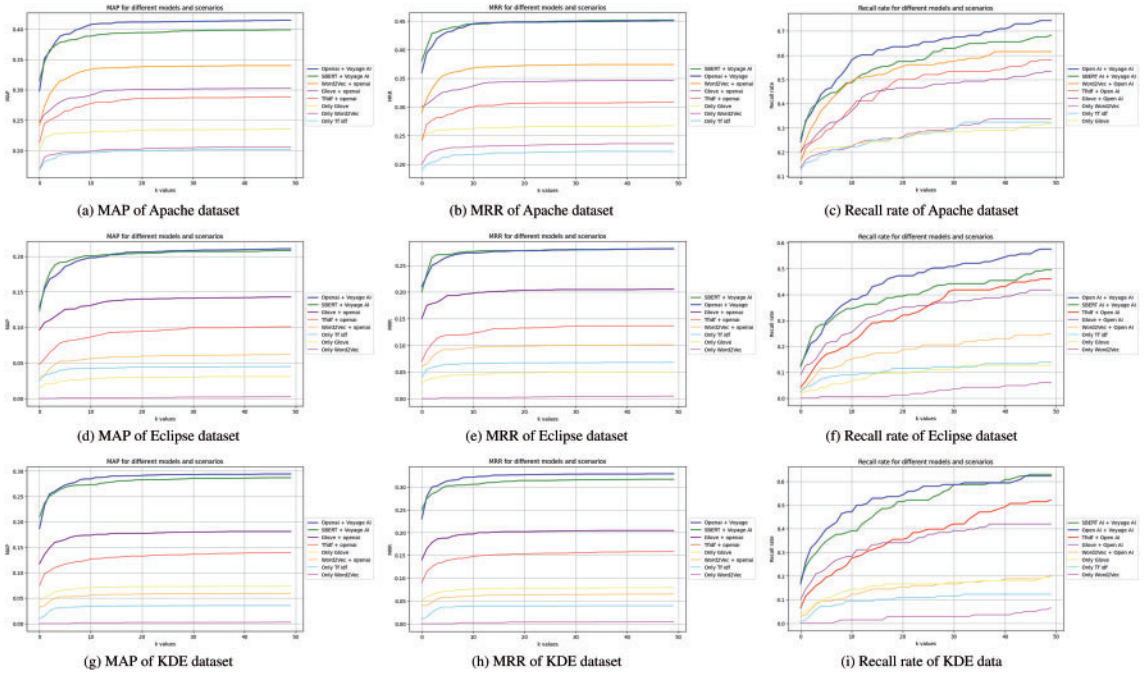
The amalgamation of TF-IDF+ OpenAI and Glove + OpenAI showed moderate performance, 0.54 and 0.58, respectively. Traditional models, like TF-IDF, Word2Vec, and Glove, exhibit the lowest recall rates. Recall rates improve rapidly at lower k values and stabilize at higher k values. Advanced models with Voyage AI significantly enhance recall performance across all datasets. Further, in Fig. 4, the evaluation metrics comparison has been presented for all the datasets. The analysis has revealed that the amalgamated score is more valuable than those obtained from individual approaches.

The proposed method was also compared with several state-of-the-art approaches for duplicate bug report detection. TF-IDF + SVM struggles with ambiguity and non-textual data, while BERT performs well with text but has limitations in context and requires high computational resources. Word2Vec + Random Forests is effective for structured data but faces challenges with domain-specific terminology. Hybrid approaches combining deep learning and traditional methods improve performance but lack scalability and integration of non-textual information. In contrast, the proposed amalgamated model (OpenAI + Voyage AI, SBERT + Voyage AI) outperforms these methods in RR@k, MAP, and MRR, achieving higher recall, better accuracy, and more consistent ranking.



(a) Apache dataset (b) Eclipse dataset (c) KDE dataset

Figure 3: Recall rate for amalgamated and established individual approaches, with varying values of the parameter k for datasets



(a) MAP of Apache dataset (b) MRR of Apache dataset (c) Recall rate of Apache dataset

(d) MAP of Eclipse dataset (e) MRR of Eclipse dataset (f) Recall rate of Eclipse dataset

(g) MAP of KDE dataset (h) MRR of KDE dataset (i) Recall rate of KDE data

Figure 4: Comparative evaluation results for the models for MAP, MRR, and recall rate for different datasets

5.2 Ablation Study

In the ablation study, we evaluated the contributions of classical models (e.g., TF-IDF, GloVe, Word2Vec) and LLMs, such as SBERT and OpenAI to the amalgamated model for duplicate bug detection. By systematically removing different components, we observed several key findings. First, removing classical

models, such as TF-IDF and GloVe, resulted in a moderate drop in performance, indicating that these models play an important role in handling simpler, structured bug reports. On the other hand, removing LLMs, such as SBERT and OpenAI, led to a significant degradation in performance, highlighting their critical role in capturing complex semantic relationships within bug reports. Additionally, a layer-wise ablation of LLMs demonstrated that the deeper layers are essential for maintaining high performance. Among the tested combinations, the amalgamation of OpenAI (Voyage AI) and SBERT consistently delivered the best results, achieving the highest scores in both precision and recall across all datasets. The study confirms that combining classical and LLM-based approaches results in a more robust and effective system for detecting duplicate bug reports.

The analysis of various models using the Apache, Eclipse, and KDE datasets consistently shows that the combinations of OpenAI + Voyage AI and SBERT + Voyage AI achieve the highest scores across all metrics. These combinations demonstrate superior performance in quickly retrieving relevant results with higher precision, ranking, and recall values than other combinations. Integrating Voyage AI with OpenAI or SBERT significantly enhances retrieval performance across different scenarios.

5.3 Discussion

1) *RQ1:* Several factors influence the ability to detect duplicates in bug reports, and various methods have been proposed to address this challenge. The variation in vocabulary makes it difficult to detect duplicate bug reports. Different terms used by users create gaps between similar issues. Poorly written or ambiguous description of the bug report also hinders the accurate identification of duplicates. Additionally, excluding non-textual information and a bug that spans multiple components or products can impact result quality. Traditional vector space models have been widely used for duplicate detection, often combined with heuristics and domain-specific adaptations. Machine learning models like support vector machines, random forests, and neural networks have shown promising results, especially when combining textual and structured features. Recent studies have explored DL techniques such as CNNs, attention mechanisms, and transformer models (e.g., BERT) to capture semantic similarities between bug reports [55]. Pre-trained language models fine-tuned on bug report data using transfer or few-shot learning have shown potential for effective duplicate detection. These models can generate more accurate and contextually relevant responses, addressing the limitations of earlier methods. For example, LLMs have been effectively used to mitigate misinformation by recanting and retrieving accurate statements, achieving high similarity recall rates. Despite their advantages, LLMs are not without challenges. Issues such as model hallucination, where the model generates inaccurate or misinterpreted data, and concerns around user privacy and data optimization must be addressed to ensure reliable and ethical use. Moreover, integrating LLMs with traditional IR systems can lead to a more balanced and user-centric approach. To mitigate hallucination, integrating rule-based validation, using classical IR systems for cross-verification, and fine-tuning LLMs on domain-specific data can improve accuracy. Hybrid models combining LLMs with traditional IR approaches balance precision and contextual understanding, enhancing reliability. For example, methodologies to optimize the retrieval process, select optimal models, and effectively scale and orchestrate LLMs can improve result accuracy and cost efficiency. Still, their effectiveness can vary across different domains and applications. For this study, the amalgamation of models took advantage of their strengths and improved overall performance. Further, LLM models and their fusion (OpenAI + Voyage AI and SBERT + Voyage AI) have outperformed the classical IR approaches.
2) *RQ2:* LLMs struggle with semantically related and randomly chosen pairs [56]. This search presented the ensemble approach of the classical models and LLMs to enhance the reliability of similarity scores in

semantic search for duplicate detection, especially given their limited context awareness. There are other approaches mentioned in the literature that can be used for improvement, which we will experiment with in future work. One of the methods is to leverage the rewriting capabilities of LLMs to augment data, thereby improving the robustness of models in recognizing duplicates. For instance, the technique employs contextual rewriting and entity replacement strategies to enhance the quality of training data. This can significantly improve the performance of few-shot named entity recognition (NER) tasks and, by extension, duplicate detection tasks [57]. Additionally, integrating knowledge graphs (KGs) with LLMs can address the issue of hallucinations and enhance the reasoning process. LLMs can combine explicit and implicit knowledge by iteratively exploring KGs and retrieving task-relevant subgraphs, leading to more reliable similarity assessments [58]. Further, fine-tuning, or continual pre-training of LLMs on software engineering data can help them better understand domain-specific terminology and context. Incorporating structured information (e.g., product, component, stack traces) alongside unstructured text can provide additional context and improve similarity assessments. The current study successfully integrates non-textual components that previous researchers could not incorporate [27]. By combining these strategies, the reliability of similarity scores in semantic search and LLMs for duplicate detection can be significantly enhanced, addressing the challenges posed by limited context awareness.

3) *RQ3:* Amalgamating or combining different models has shown promising results in improving the performance of duplicate bug report detection. For duplicate detection tasks framed as ranking or retrieval problems, ensembling different ranking models or combining their similarity scores can improve the overall ranking quality. Polisetty et al. [59] identified that although DL models perform better than traditional machine learning models, they only partially fulfill the criteria set by practitioners. Qian et al. [60] suggested integrating NLP models to enhance bug report descriptions and runtime information accuracy. Jiang et al. [13] found that their combined IR and deep learning method outperformed individual approaches in ranking potential duplicate bug reports. Although amalgamating models can improve performance, it is essential to carefully design the ensemble strategy, considering factors such as model diversity, computational complexity, and potential trade-offs between accuracy and efficiency. Additionally, effective techniques for model selection, weighting, and fusion are crucial for realizing the full potential of ensemble approaches.

Designing the ensemble strategy requires attention to several critical factors. Model diversity is essential, and this is achieved by selecting models with different architectures and training data to capture various aspects of the data. Balancing trade-offs between improved accuracy and increased computational cost is necessary. This is done by testing different combinations and weights of the models and employing strategies such as the heuristic ranking method that combines these and creates a universally ranked set of results. Potential limitations and challenges associated with combining multiple models include computational efficiency. Measures such as efficient coding practices and hardware acceleration (e.g., using GPUs) are employed to optimize computational efficiency. Trade-offs between computational cost and accuracy are inevitable, and accepting a slight reduction in accuracy may be necessary for significant gains in efficiency. Techniques like early stopping during model training or using simpler models in the ensemble can help achieve this balance. By carefully considering these factors and employing appropriate techniques, the challenges associated with combining multiple models can be effectively managed to enhance the performance of duplicate bug report detection systems.

4) *RQ4:* (Statistical significance and effect size): We used the Wilcoxon signed rank test to calculate the p-value and measured Cliff's Delta and Spearman correlation to assess the results of the proposed model. Cliff's Delta measure is interpreted in Table 2. After the Shapiro-Wilk test identified non-normal distribution, the non-parametric Spearman correlation test was used to assess relationships between

different method results. Its value varies between −1 and +1, with 0 implying no correlation. Non-parametric tests were chosen because the data did not follow a normal distribution, as indicated by the Shapiro-Wilk test. Non-parametric tests like the Wilcoxon signed rank test and Spearman correlation are more appropriate for non-Gaussian data because they do not assume a specific distribution, making them suitable for our dataset.

Table 2: Interpretation of cliff's delta scores [54]

Effect size	Cliff's delta (δ)
Negligible	$-1.00 \leq \delta < 0.147$
Small	$0.146 \leq \delta < 0.330$
Medium	$0.330 \leq \delta < 0.474$
Large	$0.474 \leq \delta \leq 1.00$

Table 3 presents the *p*-value, Cliff's Delta measure and Spearman's correlation coefficient of an amalgamated (OpenAI + Voyage AI) model with TF-IDF (for example) for the given datasets. The TF-IDF model and amalgamated approach, often considered a benchmark model in previous studies, have been compared. Table 3 shows a positive correlation with a medium to large effect size, indicating improvement through model amalgamation. A positive Spearman correlation implies that as the performance of one model improves, the performance of the other model also improves, indicating consistency between the models' results. Cliff's Delta measure, indicating a large effect size, suggests that the magnitude of improvement due to the amalgamation of models is substantial. This means the amalgamated model performs significantly better than the benchmark model, highlighting the effectiveness of combining models.

Table 3: *p*-value of Wilcoxon signed-rank test, Cliff's Delta and Spearman's correlation coefficient comparing the metrics of amalgamated (OpenAI + Voyage AI) model with TF-IDF for Apache dataset

Metrics	Spearman's r	Cliff's delta	*p*-value
Recall	0.98	0.9712	4.77618E–17
MRR	0.97	1	5.4159E–18
MAP	0.98	1	5.57032E–18

Further, the error analysis of the proposed amalgamated model for duplicate bug detection identified several key failure modes and areas for improvement. The model struggled with semantic misunderstandings, often misclassifying bug reports with subtle contextual differences as duplicates. Additionally, it faced challenges with ambiguous descriptions, indicating a reliance on textual features that could be enhanced. For instance, the report 'System crashes during data import' was erroneously identified as a duplicate of 'Application error while uploading files,' reflecting a semantic misunderstanding. Similarly, a vague description like 'App stops responding when opened' could erroneously match issues such as 'Crash on startup' or 'Hangs after login.' The model also underutilized non-textual attributes that are critical for differentiation and exhibited instances of overfitting to training data, leading to lower generalization performance. Furthermore, as dataset sizes increased, especially with Eclipse and KDE, performance slightly declined, suggesting scalability issues. This analysis highlights essential areas for refining the model's accuracy and robustness in practical applications.

Further, as dataset size increased (e.g., Eclipse and KDE), scalability issues became evident, particularly with LLM-based methods. While the proposed method achieved higher recall rates, future work should explore model optimization techniques like quantization and pruning to balance accuracy and efficiency. Additionally, leveraging lightweight architectures (e.g., DistilBERT) could enable real-time applications while maintaining acceptable performance levels.

6 Threats to Validity

6.1 Internal Validity

There may be threats to the effectiveness of compiling the bug reports dataset in finding a sufficiently large number of bug reports. These threats are confined to internal validity errors in current empirical experiments. The dataset repository contains bug reports up to 2017, but each report contains limited textual information. We used well-established NLP methods to prepare the corpus from large datasets, so we believe there are no significant threats to internal validity.

6.2 External Validity

Another limitation of this study may be the generalization of results. The similarity score was computed by following several steps, and each of these steps significantly affected the results. However, results are verified using open-source datasets to achieve enough generalization. To improve generalizability, future studies should extend the analysis to larger and more diverse datasets. Validation of additional datasets would help assess the robustness of the proposed amalgamated model in varied contexts.

7 Conclusion

This study tackles the critical challenge of duplicate bug reporting in software repositories by comparing classical word embedding techniques with contemporary LLM-based word embeddings. The findings show that amalgamating multiple models, incorporating both textual and non-textual information, significantly enhances the accuracy of duplicate bug detection. Several validation steps were undertaken to ensure that the fusion process improved performance and maintained the robustness and generalizability of the results without introducing significant biases: 1) *Empirical Validation*: The performance of different amalgamated approaches was compared with individual established approaches in detecting duplicate bug reports. The evaluation was performed on multiple datasets, including Apache, Eclipse, and KDE. The metrics used for comparison included RR@k, MAP, and MRR; 2) *Statistical Tests*: Statistical significance and effect size of the proposed models were evaluated using the Wilcoxon signed rank test, Cliff's Delta measure, and Spearman correlation. These tests provided insights into the improvement of amalgamated models over benchmark models, such as TF-IDF. Positive Spearman correlation values and large Cliff's Delta effect sizes indicated substantial improvements due to model amalgamation; 3) *Validation Metrics*: The use of established metrics such as RR@k, MAP, and MRR ensured that the performance improvements were measured accurately and consistently across different datasets and scenarios; 4) *Generalization*: Results were verified using open-source datasets to ensure that the findings were generalizable beyond the specific datasets used in the study. This helped in minimizing the risk of overfitting and ensured the applicability of the models in different contexts; 5) *Evaluation of Diverse Models*: The amalgamated models incorporated both traditional models (like TF-IDF, Word2Vec, GloVe) and advanced language models (like OpenAI, SBERT, and Voyage AI), ensuring diversity in the model architectures and training data. This diversity helped in capturing various aspects of the data and improving the overall performance of the duplicate bug report detection system. The empirical analysis of OSS datasets, including Apache, Eclipse, and KDE, revealed that combined scores from amalgamated models outperformed individual approaches. Specifically, OpenAI + Voyage AI and SBERT

+ Voyage AI consistently achieved the highest scores across all metrics, indicating superior performance in quickly and consistently retrieving relevant results. Strategies to improve the reliability of similarity scores include fine-tuning LLMs on domain-specific data and incorporating structured information. Future work can delve deeper into the factors influencing the detection of duplicate bug reports, exploring enhancements in semantic search and LLMs to address context awareness limitations. Additionally, investigating the proposed models' statistical significance and effect size will provide more insights into their practical applicability. We will include visualizations to demonstrate how different components contribute to the final predictions for specific bug reports. To improve generalizability and scalability, future research will explore domain-specific fine-tuning of LLMs, leveraging knowledge graphs for context, and adopting lightweight models for resource-constrained settings.

Acknowledgement: The authors are grateful to all the editors and anonymous reviewers for their comments and suggestions.

Funding Statement: The authors received no specific funding for this study.

Author Contributions: The authors confirm contribution to the paper as follows: study conception and design, data collection, original draft writing: Sukhjit Singh Sehra, Sai Venkata Akhil Ammu; data collection, cleaning, coding: Sumeet Kaur Sehra; analysis and interpretation of results: Sukhjit Singh Sehra, Jaiteg Singh; draft manuscript preparation: Sumeet Kaur Sehra, Sai Venkata Akhil Ammu. All authors reviewed the results and approved the final version of the manuscript.

Availability of Data and Materials: The data that support the findings of this study are openly available at https://zenodo.org/record/400614.XaNPt-ZKh8, accessed on 12 January 2025.

Ethics Approval: Not applicable.

Conflicts of Interest: The authors declare no conflicts of interest to report regarding the present study.

Abbreviations

LLMs	Large Language Models
NLP	Natural Language Processing
DL	Deep Learning
IR	Information Retrieval
STS	Semantic Textual Similarity
AI	Artificial Intelligence
MAP	Mean Average Precision
MRR	Mean Reciprocal Rank
GLOVE	Global Vectors for Word Representation
TF-IDF	Term Frequency-Inverse Document Frequency

References

1. Sehra SS, Abdou T, Başar A, Sehra SK. Amalgamated models for detecting duplicate bug reports. In: Goutte C, Zhu X, editors. Advances in artificial intelligence. Canadian AI. Lecture notes in computer science. Cham: Springer; 2020. Vol. 12109, p. 470–82. doi:10.1007/978-3-030-47358-7_49.
2. Wu X, Shan W, Zheng W, Chen Z, Ren T, Sun X. An intelligent duplicate bug report detection method based on technical term extraction. In: 2023 IEEE/ACM International Conference on Automation of Software Test; 2023 May; Melbourne, Australia. p. 1–12. doi:10.1109/AST58925.2023.00005.

3. Sun C, Lo D, Wang X, Jiang J, Khoo S-C. A discriminative model approach for accurate duplicate bug report retrieval. In: Proceedings of the 32nd ACM/IEEEInternational Conference on Software Engineering; 2010 May; Cape Town, South Africa. p. 45–54. doi:10.1145/1806799.1806811.
4. Zhang T, Han D, Vinayakarao V, Irsan IC, Xu B, Thung F, et al. Duplicate bug report detection: how far are we? ACM Trans Softw Eng Methodol. 2023 Jul;32(4):1–32. doi:10.1145/3603109.
5. Rakha MS, Shang W, Hassan AE. Studying the needed effort for identifying duplicate issues. Empir Softw Eng. 2016 Oct;21(5):1960–89. doi:10.1007/s10664-015-9404-6.
6. Jiang Y, Su X, Treude C, Shang C, Wang T. Does deep learning improve the performance of duplicate bug report detection? an empirical study? J Syst Softw.2023 Apr;198(3):111607. doi:10.1016/j.jss.2023.111607.
7. Sun C, Lo D, Khoo SC, Jiang J. Towards more accurate retrieval of duplicate bug reports. In: Proceedings of IEEE/ACM International Conference on Automated Software Engineering; 2011 Nov; Lawrence, KS, USA. p. 253–62. doi:10.1109/ASE.2011.6100061.
8. Klein N, Corley CS, Kraft NA. New features for duplicate bug detection. In: Proceedings of the 11th Working Conference on Mining Software Repositories; 2014; Hyderabad, India. p. 324–7. doi:10.1145/2597073.2597090.
9. Nguyen AT, Nguyen TT, Nguyen TN, Lo D, Sun C. Duplicate bug report detection with a combination of information retrieval and topic modeling. In: Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering; 2012 Sep; Essen, Germany. p. 70–9. doi:10.1145/2351676.2351687.
10. Wang X, Zhang L, Xie T, Anvik J, Sun J. An approach to detecting duplicate bug reports using natural language and execution information. In: Proceedings of the 30th International Conference on Software Engineering; 2008 May; Leipzig, Germany. p. 461–70. doi:10.1145/1368088.1368151.
11. Jalbert N, Weimer W. Automated duplicate detection for bug tracking systems. In: Proceedings of the 30th International Conference on Software Engineering; 2008 Jun; Anchorage, AK, USA. p. 52–61. doi:10.1109/DSN.2008.4630070.
12. Zou J, Xu L, Yang M, Zhang X, Zeng J, Hirokawa S. Automated duplicate bug report detection using multi-factor analysis. IEICE Trans Inf Syst. 2016 Jul;E99.D(7):1762–75. doi:10.1587/transinf.2016EDP7052.
13. Yang X, Lo D, Xia X, Bao L, Sun J. Combining word embedding with information retrieval to recommend similar bug reports. In: 2016 IEEE 27th International Symposium on Software Reliability Engineering; 2016 Oct; Ottawa, ON, Canada. p. 127–37. doi:10.1109/ISSRE.2016.33.
14. Jiang H, Nazar N, Zhang J, Zhang T, Ren Z. PRST: a pagerank-based summarization technique for summarizing bug reports with duplicates. Int J Softw Eng Knowl Eng. 2017;27(6):869–96. doi:10.1142/S0218194017500322.
15. Tian Y, Sun C, Lo D. Improved duplicate bug report identification. In: 2012 16th European Conference on Software Maintenance and Reengineering (CSMR'03); 2012; Szeged, Hungary. p. 385–90. doi:10.1109/CSMR.2012.48.
16. Alipour A, Hindle A, Stroulia E. A contextual approach towards more accurate duplicate bug report detection. In: 2013 10th Working Conference on Mining Software Repositories (MSR 2013); 2013 May; San Francisco, USA. p. 183–92. doi:10.1109/MSR.2013.6624026.
17. Lazar A, Ritchey S, Sharif B. Improving the accuracy of duplicate bug report detection using textual similarity measures. In: Proceedings of the 11th Working Conference on Mining Software Repositories; 2014; Hyderabad, India. p. 308–11. doi:10.1145/2597073.2597088.
18. Budhiraja A, Dutta K, Reddy R, Shrivastava M. DWEN: deep word embedding network for duplicate bug report detection in software repositories. In: Proceedings of the 40th International Conference on Software Engineering: Companion; 2018 May; Gothenburg, Sweden. p. 193–4.
19. Zhang T, Irsan IC, Thung F, Lo D. Cupid: leveraging ChatGPT for more accurate duplicate bug report detection. 2023. doi:10.48550/arXiv.2308.10022.
20. Kukkar A, Mohana R, Kumar Y, Nayyar A, Bilal M, Kwak K-S. Duplicate bug report detection and classification system based on deep learning technique. IEEE Access. 2020 Oct;8:200749–63. doi:10.1109/ACCESS.2020.3033045.
21. Sivapurnima S, Manjula D. Adaptive deep learning model for software bug detection and classification. Comput Syst Sci Eng. 2023;45(2):1233–48. doi:10.32604/csse.2023.025991.
22. Gupta S, Gupta SK. A systematic study of duplicate bug report detection. Int J Adv Comput Sci Appl. 2021 Jan;12(1):578–89. doi:10.14569/issn.2156-5570.

23. He J, Xu L, Yan M, Xia X, Lei Y. Duplicate bug report detection using dual-channel convolutional neural networks. In: IEEE/ACM 28th International Conference on Program Comprehension; 2020 Oct; Seoul, Republic of Korea. p. 117–27. doi:10.1145/3387904.3389263.
24. Neysiani BS, Babamir SM. Automatic duplicate bug report detection using information retrieval-based versus machine learning based approaches. In: 2020 6th International Conference on Web Research; 2020 Apr; Tehran, Iran. p. 288–93. doi:10.1109/ICWR49608.2020.9122288.
25. Fang S, Zhang T, Tan Y, Jiang H, Xia X, Sun X. RepresentThemAll: a universal learning representation of bug reports. In: 2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE); 2023 May; Melbourne, Australia. p. 602–14. doi:10.1109/ICSE48619.2023.00060.
26. Sadat M, Bener AB, Miranskyy AV. Rediscovery datasets: connecting duplicate reports. In: Proceedings of the 2017 IEEE/ACM 14th International Conference on Mining Software Repositories; 2017 May; Buenos Aires, Argentina. p. 527–30. doi:10.1109/MSR.2017.50.
27. Bocu R, Baicoianu A, Kerestely A. An extended survey concerning the significance of artificial intelligence and machine learning techniques for bug triage and management. IEEE Access. 2023;11(1):123924–37. doi:10.1109/ACCESS.2023.3329732.
28. Zheng W, Li Y, Wu X, Cheng J. Duplicate bug report detection using named entity recognition. Knowl-Based Syst. 2024;284(4):111258. doi:10.1016/j.knosys.2023.111258.
29. Jahan S, Rahman MM. Towards understanding the impacts of textual dissimilarity on duplicate bug report detection. In: 2023 IEEE International Conference on Software Analysis, Evolution and Reengineering; 2023 Mar; Taipa, Macao. p. 25–36. doi:10.1109/SANER56733.2023.00013.
30. Chauhan R, Sharma S, Goyal A. DENATURE: duplicate detection and type identification in open source bug repositories. Int J Syst Assur Eng Manag. Mar. 2023;14(Suppl 1):S275–92. doi:10.1007/s13198-023-01855-x.
31. Isotani H, Washizaki H, Fukazawa Y, Nomoto T, Ouji S, Saito S. Sentence embedding and fine-tuning to automatically identify duplicate bugs. Front Comput Sci. 2022;4:108318. doi:10.3389/fcomp.2022.1032452.
32. Zhou C, Li B, Sun X, Yu S. Leveraging multi-level embeddings for knowledge-aware bug report reformulation. J Syst Softw. 2023;198(3):111617. doi:10.1016/j.jss.2023.111617.
33. Kalyan KS. A survey of GPT-3 family large language models including ChatGPT and GPT-4. Nat Lang Process. 2024;6(6):100048. doi:10.1016/j.nlp.2023.100048.
34. Minaee S, Mikolov T, Nikzad N, Chenaghlu M, Socher R, Amatriain X, et al. Large language models: a survey. 2024 Feb. doi:10.48550/arXiv.2402.06196.
35. Zhao WX, Zhou K, Li J, Tang T, Wang X, Hou Y, et al. A survey of large language models. 2023 Nov. doi:10.48550/arXiv.2303.18223.
36. Huang J, Gu S, Hou L, Wu Y, Wang X, Yu H, et al. Large language models can self-improve. In: Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing; 2023 Dec; Singapore. p. 1051–68. doi:10.18653/v1/2023.emnlp-main.67.
37. Pan S, Luo L, Wang Y, Chen C, Wang J, Wu X. Unifying large language models and knowledge graphs: a roadmap. IEEE Trans Knowl Data Eng. 2024;36(7):1–20. doi:10.1109/TKDE.2024.3352100.
38. De Angelis L, Baglivo F, Arzilli G, Privitera GP, Ferragina P, Tozzi AE, et al. ChatGPT and the rise of large language models: the new AI-driven infodemic threat in public health. Front Public Health. 2023;11:1166120. doi:10.3389/fpubh.2023.1166120.
39. Touvron H, Lavril T, Izacard G, Martinet X, Lachaux M-A, Lacroix T, et al. LLaMA: open and efficient foundation language models. Tech Rep. 2023 Feb. doi:10.48550/arXiv.2302.13971.
40. Nagwani NK, Singh P. Bug mining model based on event-component similarity to discover similar and duplicate GUI bugs. In: 2009 IEEE International Advance Computing Conference; 2009 Mar; Patiala, India. p. 1388–92, 2009. doi:10.1109/IADCC.2009.4809219.
41. Sureka A, Jalote P. Detecting duplicate bug report using character n-gram-based features. In: 17th Asia Pacific Software Engineering Conference (APSEC 2010); 2010 Dec; Sydney, Australia. p. 366–74. doi:10.1109/APSEC.2010.49.
42. Mikolov T. Efficient estimation of word representations in vector space. arXiv:1301.3781. 2013;3781.

43. Brochier R, Guille A, Velcin J. Global vectors for node representations. In: World Wide Web Conference (WWW '19); 2019 May; San Francisco, CA, USA. p. 2587–93. doi:10.1145/3308558.3313595.
44. Devlin J, Chang M-W, Lee K, Toutanova K. Bert: pre-training of deep bidirectional transformers for language understanding. 2018 Oct. doi:10.48550/arXiv.1810.04805.
45. Liu Y, Ott M, Goyal N, Du J, Joshi M, Chen D, et al. RoBERTa: a robustly optimized BERT pretraining approach. 2019. doi:10.48550/arXiv.1907.11692.
46. Thakur N, Reimers N, Daxenberger J, Gurevych I. Augmented SBERT: data augmentation method for improving bi-encoders for pairwise sentence scoring tasks. In: Toutanova K, Rumshisky A, Zettlemoyer L, Hakkani-Tur D, Beltagy I, Bethard S, et al., editors. Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies; 2021 Jun; Association for Computational Linguistics. p. 296–310. doi:10.18653/v1/2021.naacl-main.28.
47. Fournier L, Dupoux E, Dunbar E. Analogies minus analogy test: measuring regularities in word embeddings. arXiv:2010.03446. 2020
48. Neysiani BS, Babamir SM. New methodology for contextual features usage in duplicate bug reports detection. In: 5th International Conference on Web Research; 2019 Apr; Tehran, Iran. p. 178–83. doi:10.1109/ICWR.2019.8765296.
49. Akilan T, Shah D, Patel N, Mehta R. Fast detection of duplicate bug reports using LDA-based topic modeling and classification. In: 2020 IEEE International Conference on Systems, Man, and Cybernetics (SMC); 2020 Oct; Toronto, ON, Canada. p. 1622–9. doi:10.1109/SMC42975.2020.9283289.
50. Face H. Massive text embedding benchmark (MTEB); 2024 [cited 2024 Nov 20]. Available from: https://huggingface.co/spaces/mteb.
51. Sanh V, Debut L, Chaumond J, Wolf T. Distilbert, a distilled version of BERT: smaller, faster, cheaper and lighter. 2019 Oct. doi:10.48550/arXiv.1910.01108.
52. Wolf T, Debut L, Sanh V, Chaumond J, Delangue C, Moi A, et al. Huggingface's transformers: state-of-the-art natural language processing. 2019 Oct. doi:10.48550/arXiv.1910.03771.
53. Rocha H, Valente MT, Marques-Neto H. Characterizing bug workflows in Mozilla Firefox. In: Thirtieth Brazilian Symposium on Software Engineering (SBES 2016); 2016 Sep; Brazil. p. 43–52. doi:10.1145/2973839.2973844.
54. Macbeth G, Razumiejczyk E, Ledesma RD. Cliff's delta calculator: a non-parametric effect size program for two groups of observations. Univ Psychol. 2011;10(2):545–55. doi:10.11144/Javeriana.upsy10-2.cdcp.
55. Messaoud MB, Miladi A, Jenhani I, Mkaouer MW, Ghadhab L. Duplicate bug report detection using an attention-based neural language model. IEEE Trans Rel. 2023;72(2):846–58. doi:10.1109/TR.2022.3193645.
56. Freestone M, Santu SKK. Word embeddings revisited: do LLMs offer something new?. 2024 Feb. doi:10.48550/arXiv.2402.11094.
57. Ye J, Xu N, Wang Y, Zhou J, Zhang Q, Gui T, et al. Data augmentation via large language models for few-shot named entity recognition. 2024 Feb. doi:10.48550/arXiv.2402.14568.
58. Li Y, Zhang R, Liu J. An enhanced prompt-based LLM reasoning scheme via knowledge graph-integrated collaboration. 2024 Feb. doi:10.48550/arXiv.2402.14568.
59. Polisetty S, Miranskyy A, Başar A. On usefulness of the deep-learning-based bug localization models to practitioners. In: Proceedings of the Fifteenth International Conference on Predictive Models and Data Analytics in Software Engineering (PROMISE'19); 2019 Sep; Recife Brazil. p. 16–25. doi:10.1145/3345629.3345632.
60. Qian C, Zhang M, Nie Y, Lu S, Cao H. A survey on bug deduplication and triage methods from multiple points of view. Appl Sci. 2023;13(15):8788. doi:10.3390/app13158788.