# ChatBR: Automated Assessment and Improvement of Bug Report Quality Using ChatGPT

Lili Bo
lilibo@yzu.edu.com
[1] Yangzhou University
Yangzhou, China
[2] Yunnan Key Laboratory of Software Engineering
Kunming, China

Wangjie Ji
MZ120220952@stu.yzu.edu.cn
Yangzhou University
Yangzhou, China

Xiaobing Sun*
xbsun@yzu.edu.cn
Yangzhou University
Yangzhou, China

Ting Zhang
tingzhang.2019@phdcs.smu.edu.sg
Singapore Management University
Singapore, Singapore

Xiaoxue Wu
xiaoxuewu@yzu.edu.cn
Yangzhou University
Yangzhou, China

Ying Wei
008639@yzu.edu.cn
Yangzhou University
Yangzhou, China

## ABSTRACT

Bug reports, containing crucial information such as the Observed Behavior (OB), the Expected Behavior (EB), and the Steps to Reproduce (S2R), can help developers localize and fix bugs efficiently. However, due to the increasing complexity of some bugs and the limited experience of some reporters, many bug reports miss this crucial information. Although machine learning (ML)-based and information retrieval (IR)-based approaches have been proposed to detect and supplement the missing information in bug reports, the performance of these approaches depends heavily on the size and quality of bug report datasets.

In this paper, we present ChatBR, an approach for automated assessment and improvement of bug report quality using ChatGPT. First, we fine-tune a BERT model using manually annotated bug reports to create a sentence-level multi-label classifier to assess the quality of bug reports by detecting the presence of OB, EB, and S2R. Second, we use ChatGPT in a zero-shot setup to generate the missing information (OB, EB, and S2R) to improve the quality of bug reports. Finally, the output of ChatGPT is fed back into the classifier for verification until ChatGPT generates the missing information. Experimental results demonstrate ChatBR's superiority in both detecting and generating missing information in bug reports. For detection, ChatBR surpasses the state-of-the-art method, improving precision by 25.38% to 29.20%. In generating missing information, ChatBR achieves an average semantic similarity of 77.62% between generated and original content across six diverse projects. Furthermore, ChatBR can generate more than 99.9% of high-quality bug reports (i.e., bug reports that are full of OB, EB, and S2R) within five ChatGPT runs.

## CCS CONCEPTS

• **Software and its engineering** → **Maintaining software**.

## KEYWORDS

Bug Report, ChatGPT, Pre-trained Models, Large Language Models

**ACM Reference Format:**
Lili Bo, Wangjie Ji, Xiaobing Sun, Ting Zhang, Xiaoxue Wu, and Ying Wei. 2024. ChatBR: Automated Assessment and Improvement of Bug Report Quality Using ChatGPT. In *39th IEEE/ACM International Conference on Automated Software Engineering (ASE '24), October 27-November 1, 2024, Sacramento, CA, USA.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3691620.3695518

## 1 INTRODUCTION

Bug reports are vital for developers during software development as they document unexpected software behaviors encountered by users [53]. OB (Observed Behavior), EB (Expected Behavior), and S2R (Steps to Reproduce) offer important information that is highly useful for developers to trigger and fix bugs. Specifically, EB describes the correct or intended functionality of the software system. OB details the observed performance or functionality that deviates from the EB. S2R outlines the specific actions required to consistently recreate the reported issue. Missing the above information may waste the time of developers in understanding and reasoning about bugs for correct fixes.

Many approaches have been proposed to assess and improve the quality of bug reports [4, 13, 20, 21]. Some approaches typically depend on heuristic rules and expert knowledge for detecting crucial information in bug reports, but they are difficult to extend with the increasing rules and software scale. Zhang et al. [45] present an information retrieval (IR)-based approach to detect the missing information and use historically similar bug reports to enrich them. Unfortunately, historically similar bug reports are not always available. Moreover, directly adding sentences extracted from similar bug reports can cause semantic incoherence. In addition, machine

*Xiaobing Sun is the Corresponding author


*ASE '24, October 27-November 1, 2024, Sacramento, CA, USA*

ACM ISBN 979-8-4007-1248-7/24/10
https://doi.org/10.1145/3691620.3695518

learning (ML)-based approaches are proposed and have shown their good performance in assessing bug reports. For example, Song et al. [36] develop multiple binary classification models to detect missing OB, EB, and S2R information in bug reports. However, ML-based approaches for assessing bug reports face limitations due to two primary factors. First, these approaches struggle to effectively capture the deep semantics of natural language context. Second, their performance heavily depends on large-scale, high-quality bug report datasets for training, which are not always available.

In this paper, we propose a novel approach (ChatBR) based on ChatGPT to alleviate the above limitations.

To improve the accuracy of assessing bug reports, we adopt the pre-trained model that relies on the attention mechanism to capture the deep semantics of natural language context effectively. Only a small number of bug reports are needed to fine-tune the pre-trained model, which can save much time in collecting high-quality bug reports. Furthermore, the generated bug reports are iteratively returned to the detector to check for missing information, improving the accuracy of assessing bug reports and the quality of final bug reports.

To ensure the semantic coherence of the improved bug report description, we use ChatGPT, a large language model (LLM), to generate a bug report context that is easy to read. First, ChatGPT has a natural ability to generate a context that is semantically coherent. Second, ChatGPT contains rich bug-related knowledge, which alleviates the dependence on similar bug reports. Considering the resource limitation of using ChatGPT, we utilize prompt engineering for guiding ChatGPT to output accurate bug reports. Furthermore, only incomplete bug reports (i.e., the bug reports missing OB, EB, or S2R) are input to ChatGPT to minimize the number of ChatGPT runs.

In summary, ChatBR (Automated assessment and improvement of Bug Report quality using ChatGPT) is a two-step approach. It consists of a detector and a generator. The detector is a fine-tuned, pre-trained model used to detect OB, EB, or S2R in bug reports. ChatGPT serves as a generator for missing information in bug reports. The detector iteratively verifies ChatGPT's output, ensuring only complete bug reports (containing OB, EB, and S2R simultaneously) are presented to users. We trained and evaluated the detector using Song et al.'s dataset [36]. The generator's performance was assessed on bug reports from six open-source projects. Results demonstrate that ChatGPT-generated information achieves 73.17% to 85.05% semantic similarity with original information across these projects. Notably, ChatGPT successfully generates missing information for 99.9% of bug reports within five runs.

The main contributions of this paper can be summarised as follows:

- We propose a two-step framework consisting of a detector (combining data augmentation and a fine-tuned pre-trained model) and a generator (using ChatGPT) to assess and improve the quality of bug reports by automatically supplementing the missing OE, EB, and S2R.
- Experimental results demonstrate ChatBR's superiority in both detecting and generating missing information in bug reports. For detection, ChatBR surpasses the state-of-the-art approach, improving precision by 25.38% to 29.20%. In generating missing information, ChatBR achieves an average semantic similarity of 77.62% between generated and original content across six diverse projects.
- Our source code and datasets are publicly available to facilitate further research.[1]

**Listing 1: Excerpt from example bug report in AspectJ**

```
{
    "ID": "384398",
    "Title": "Type Mismatch error when using inner classes
      contained in generic types within ITDs",
    "Description": "Please see attached example project.
      I get the following (strange) compiler error: Type mismatch:
     cannot convert from A<T>.InnerA<> to A.InnerA Aspect.aj
     /AspectJInnerclassInGenericTypeBug/src/de/example line 12. "
}
```

**Legend:** OB EB S2R

**Listing 2: Excerpt from refactored example bug report**

```
{
    "ID": "384398",
    "Title": "Type Mismatch error when using inner classes
      contained in generic types within ITDs",
    "Description": "Please see attached example project .",
    "OB": "I get the following (strange) compiler error: Type
      mismatch: cannot convert from A<T>.InnerA<> to A.InnerA
      Aspect.aj /AspectJInnerclassInGenericTypeBug/src/de/example
      line 12.",
    "EB": "",
    "S2R": ""
}
```

The paper is structured as follows. Section 2 illustrates our approach through a motivating example. Section 3 describes ChatBR in detail. Section 4 presents the experimental design. Section 5 shows the experimental results and conducts a deep analysis of the results. We discuss threats to validity in Section 6. Section 7 summarizes the related work. Finally, Section 8 concludes our work.

## 2 MOTIVATING EXAMPLE

In this section, we use a motivating example to illustrate our approach.

Listing 1 presents an excerpt from a sample bug report in the AspectJ project [2]. The structural elements of the report are color-coded for clarity. To address the issues outlined in this report, developers must carefully read and comprehend the entire document.

Listing 2 shows an excerpt from the refactored example bug report in Listing 1. We can see that the bug report in Listing 1 only describes the OB information (i.e., *"Type Mismatch error"*), while EB and S2R are missing. This is determined by the detector, a pre-trained model fine-tuned with the bug report dataset after data augmentation.

[1] https://github.com/jiwangjie/ChatBR

**Listing 3: Excerpt from improved example bug report**

```
{
    "ID": "384398",
    "Title": "Type Mismatch error when using inner classes
      contained in generic types within ITDs",
    "Description": "Please see attached example project .",
    "OB": "When attempting to compile the code, a type mismatch
      error occurs with the message: 'Type mismatch: cannot convert
       from A<T>.InnerA<> to A.InnerA Aspect.aj /
      AspectJInnerclassInGenericTypeBug/src/de/example line 12.'",
    "EB": "The compiler should not produce a type mismatch error
      when dealing with inner classes contained in generic types
      within ITDs .",
    "S2R": "1. Compile the provided example project . 2. inspect
      the compiler error generated when using inner classes
      contained in generic types within ITDs."
}
```

Existing work generates the missing information by adding historically similar bug report information directly to the current bug report [45]. This can disrupt the semantic coherence. LLMs are transformer-based neural networks that can predict the next token based on the preceding context [35]. They can perform the assigned tasks with prompts. Leveraging the vast capabilities of LLMs and prompt engineering, we design prompt templates for ChatGPT to generate the missing information.

Listing 3 showcases an excerpt from an enhanced bug report. It demonstrates how ChatGPT generates the EB and S2R for the original bug report shown in Listing 1. Upon manual verification, the generated EB and S2R prove to be accurate and satisfactory. In this instance, ChatGPT run three times to produce the missing EB and S2R elements.

# 3 APPROACH

## 3.1 Overview

Figure 1 shows the overview of ChatBR. It consists of three phases: the training phase, the detection phase, and the generation phase. In the training phase, data augmentation techniques are adopted to alleviate the problem of data imbalance, thereby training a detector, i.e., the pre-trained BERT model [16]. In the detection phase, the detector is used to detect the missing elements (i.e., OB, EB, and S2R) in bug reports. In the generation phase, the prompt templates are designed to guide ChatGPT in generating the missing information. In addition, the improved bug reports are returned to the detector to check whether ChatGPT successfully generates the missing information.

## 3.2 Training Phase

In this phase, we aim to construct a detector by training a multi-label classifier. It consists of three steps: data preprocessing, data augmentation, and fine-tuning a pre-trained BERT model.

The first step is data preprocessing. The initial bug reports are from the publicly available dataset shared by Song et al. [36]. We extract the ID, title, and description from each bug report, removing non-textual elements (e.g., images, URLs) and code information (e.g., code blocks, stack traces) as per Song et al.'s method [36]. We then apply the sentence tokenizer from the Natural Language Toolkit [5] to split bug report descriptions into individual sentences. To improve generalization, we remove duplicate sentences and those with fewer than 5 characters, considering the latter uninformative. Given BERT's 512-token input limit, we split longer sentences accordingly. The resulting dataset comprises labeled bug report sentences, each with a triple indicating the presence (1) or absence (0) of OB, EB, and S2R, respectively. This process reveals a significant data imbalance issue (as shown in Table 1).

To alleviate the data imbalance problem, data augmentation is conducted in the second step. Since semantic changes between the augmented text and the original text may result in label inconsistencies, we opt for token-level data augmentation [40]. Typically, token-level data augmentation in natural language can be categorized into four types of operations [42]:

- **Synonym Replacement:** Select a random word from the text and substitute it with its synonym from a predefined dictionary.
- **Random Insertion:** Select a random word from the text and insert its synonym from a predefined dictionary at a random position in the text.
- **Random Swap:** Exchange the positions of two randomly chosen words within the text.
- **Random Deletion:** Remove a word selected at random from the text.

To avoid altering the original semantics by inadvertently deleting key tokens in sentences, we only conduct the former three operations: synonym replacement, random insertion, and random swap. To enhance the generalization capability of the augmented data samples, we modify two key operations: synonym replacement and random insertion. Specifically, we randomly select 0 to $n$ tokens to replace or insert, where $n = \lambda \times \#\text{tokens}$. Ciborowska et al. [15] show that the value $\lambda$ set to 0.1 produces the best results. In addition, we use the WordNet toolkit [34] to obtain synonyms. After the second step, we obtain an augmented dataset, which consists of sentences of bug reports.

The third step is fine-tuning a pre-trained BERT model with the augmented dataset to construct a detector. Given that a single sentence may contain multiple types of information (OB, EB, and S2R), we treat the detection of missing information in bug reports as a multi-label classification task. Recent pre-trained models have demonstrated outstanding performance in the natural language processing (NLP) domain, particularly in natural language understanding and classification tasks, surpassing previous machine learning and deep learning techniques. Since bug reports are text-based descriptions written by developers, we regard them as natural language texts. Thus, we fine-tune BERT, a widely recognized pre-trained model, on the augmented dataset.

## 3.3 Detection Phase

In this phase, the constructed detector is used to detect whether bug reports miss critical information, i.e., OB, EB, and S2R. In our work, the bug reports can be divided into *complete* bug reports and *incomplete* bug reports. A complete bug report contains OB, EB, and S2R in the sentences of the bug report. An incomplete bug report is one that misses OB, EB, or S2R in the sentences of the bug report. In

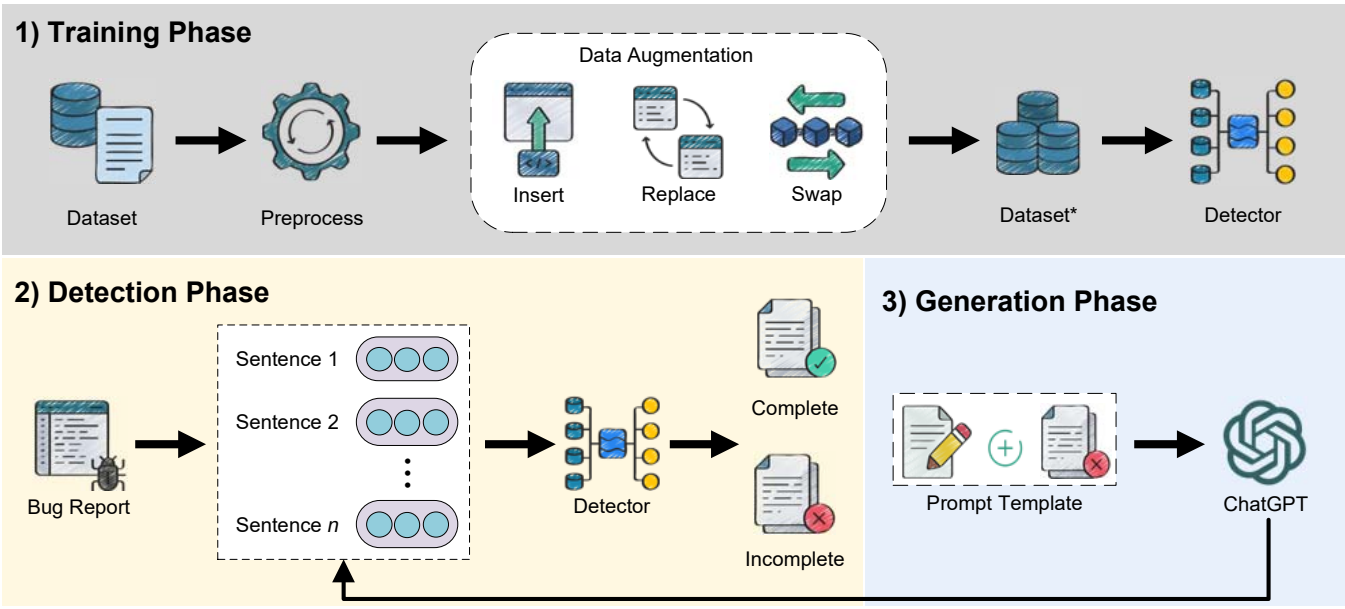


**Figure 1: Overview of ChatBR**

order to save the computational cost of ChatGPT in the generation phase, it can only run on incomplete bug reports. Therefore, it is necessary to detect missing information accurately.

For each incoming bug report, we apply the same preprocessing steps used in the training phase to extract individual sentences. The detector then predicts a label for each sentence. This label is a triple, representing the presence (1) or absence (0) of OB, EB, and S2R, as a sentence may contain multiple elements. We then aggregate these labels across all sentences in the bug report. If any of the three elements (OB, EB, or S2R) is missing from the entire report (i.e., no sentence contains that element), the report is considered incomplete. Only incomplete bug reports proceed to the next phase: generation.

## 3.4 Generation Phase

*3.4.1 Designing Prompt Templates.* Prompt [30] is a set of instructions that guides LLMs to produce a specific target result [3]. Different prompts can lead to various performances on the same task [27, 49]. Therefore, it is crucial to design precise prompts. In our work, we use a single-round dialogue interaction to design prompts and adopt the following prompting strategies to create a prompt template suitable for ChatBR: (1) providing important task-related details or context as much as possible; (2) assigning LLMs a specified role for our task; (3) using separators in the prompt to indicate different parts of the input [10, 30]; (4) the output of LLMs should conform to a common JSON format for better analysis; and (5) limited by the length of input tokens of LLMs, the prompt should express the task both accurately and concisely.

Our experimental prompt template, shown in Listing 4, is designed based on the aforementioned prompting strategies. The task involves generating missing information (OB, EB, and S2R) for incomplete bug reports, with the LLM playing the role of a software engineer. We provide definitions for OB, EB, and S2R as expert knowledge. The model's task is to infer appropriate details from the given bug report's context and supplement it with clear, complete OB/EB/S2R sentences. We specify that the output should conform to a JSON format, including fields for *ID*, *Title*, *Description*, *OB*, *EB*, and *S2R*. The input (incomplete bug report) is similarly reorganized into this JSON format. *ID* and *Title* remain unchanged. Using the sentence labels obtained during the detection phase, we populate the corresponding fields. To maintain semantic consistency, sentences retain their original order. Sentences with multiple elements appear in multiple fields, while those without OB, EB, or S2R are placed in the *Description* field to provide context for generating missing information.

*3.4.2 Running ChatGPT with Prompts.* Based on the designed prompt template, ChatGPT can run efficiently to generate improved bug reports containing missing information. In order to prevent being affected by the generated results, a new ChatGPT dialogue is reopened for each bug report. After receiving a response from ChatGPT, we first check whether the generated content conforms to the required JSON format. If the responses do not satisfy the JSON format requirement, it is necessary to run ChatGPT again for generation. Otherwise, they are returned to the detector to check whether ChatGPT generated the missing information. This is an iterative process until the detector detects all the OB, EB, and S2R.

# 4 EXPERIMENTAL DESIGN

## 4.1 Research Questions

In essence, ChatBR consists of a detector and a generator. To evaluate the performance of ChatBR, we design the following five research questions:

**Listing 4: Final Prompt Template**

**Prompt Template:**

- Your role is a senior software engineer, you are very good at analyzing and writing bug reports. You should provide clear and informative sentences for the following categories:
- Observed Behavior(OB): This section should describe the relevant software behavior, actions, output, or results. Avoid vague sentences like "the system does not work."
- Expected Behavior(EB): This part should articulate what the software should or is expected to do, using phrases like "should ...", "expect ...", or "hope ...". Avoid suggestions or recommendations for bug resolution.
- Steps to Reproduce(S2R): Include user actions or operations that can potentially lead to reproducing the issue. Use phrases like "to reproduce," "steps to reproduce," or "follow these steps."
- The bug report may lack sufficient details in the OB, EB, and S2R sections. Your task is to infer the appropriate details based on the context and supplement the bug report to ensure it contains clear and complete OB/EB/S2R sentences and improve the wording of these sentences for clarity where possible.
- Respond in JSON format as follows:

{"id": "", "title": "", "description": "", "OB": "", "EB": "", "S2R": ""}

**<BUG REPORT>...</BUG REPORT>**

- **RQ1:** *How effective is ChatBR's detector in identifying missing information?*
  We compare ChatBR's detector with BEE[36], the most relevant state-of-the-art approach. Both use the same initial dataset and incorporate data augmentation. While BEE employs three binary classification SVM models [23], ChatBR takes a different approach. This question aims to evaluate ChatBR's performance in detecting missing information in bug reports relative to BEE.
- **RQ2:** *How effective is ChatBR's generator in generating missing information?*
  Given that many existing bug reports lack crucial information (OB, EB, and S2R), we assess ChatBR's ability to improve incomplete reports. This question explores the generator's effectiveness in producing missing details. Two sub-questions further investigate influencing factors:
  - **RQ2.1:** *How does the number of missing elements affect ChatBR's effectiveness?*
  - **RQ2.2:** *How does the type of missing elements affect ChatBR's effectiveness?*
- **RQ3:** *How efficient is ChatBR in generating missing information?*
  Efficiency is crucial for practical adoption. This question evaluates ChatBR's practicality by examining the number of times ChatGPT needs to be run. Two sub-questions explore influencing factors:
  - **RQ3.1:** *How does the number of missing elements affect ChatBR's efficiency?*
  - **RQ3.2:** *How does the type of missing elements affect ChatBR's efficiency?*
- **RQ4:** *How do different prompts affect the generation results?*
  We investigate the impact of prompt design on ChatGPT's output. Starting with a basic template, we refine it by assigning ChatGPT the role of a senior software engineer skilled in bug report writing. Finally, we incorporate the specific requirement for OB, EB, and S2R based on ChatGPT best practices [11].
- **RQ5:** *How effective is ChatGPT compared to other LLMs?*
  We compare ChatGPT with two open-source LLMs: Vicuna-7B [14] and Llama2-7B [39]. Llama2 is one of the most popular LLMs and has not been specifically fine-tuned, which to some extent represents the foundational capabilities of the open-source model; Vicuna-7B is a fine-tuned model that has three improvements: (1) multi-turn conversations, (2) memory optimization, (3) cost reduction. Due to resource constraints, we limit our comparison to models with 7B or fewer parameters. All models undergo 5-round experiments using the same final prompt template.

## 4.2 Dataset

**Table 1: Dataset of training and evaluating the detector**

| Label | | | Preproccess | | After Data Augmentation |
|---|---|---|---|---|---|
| OB | EB | S2R | Before | After | |
| 0 | 0 | 0 | 87,398 | 51,559 | 51,559 |
| 1 | 0 | 0 | 7,738 | 7,642 | 42,487 |
| 0 | 0 | 1 | 5,001 | 4,897 | 33,988 |
| 0 | 1 | 0 | 1,502 | 1,487 | 33,525 |
| 1 | 0 | 1 | 1,483 | 1,481 | 35,269 |
| 1 | 1 | 0 | 239 | 239 | 32,191 |
| 1 | 1 | 1 | 46 | 46 | 15,547 |
| 0 | 1 | 1 | 25 | 25 | 7,681 |
| **Total** | | | **103,432** | **67,376** | **252,247** |

*4.2.1 Training and evaluation dataset of the detector.* Table 1 presents the dataset used for training and evaluating the detector. The first three columns show the labels representing the types of bug report sentences. Columns 4 and 5 display the number of bug report sentences before and after data preprocessing, while column 6 shows the number after data augmentation. We utilized the publicly available dataset from Song et al. [36], encompassing 35 real-world projects, over 5,000 bug reports, and 116,000 bug report sentences (including titles). After preprocessing, the number of bug report sentences decreased to 67,376. Following data augmentation, this number increased to 252,247. We randomly selected 70% of the data for training and the remaining 30% for testing. As evident from Table 1, the degree of data augmentation varies for different types of bug report sentences. For instance, no augmentation was performed on sentences lacking OB, EB, and S2R, although they constitute the majority of the dataset. Conversely, multiple augmentation operations were conducted on sentences containing OB, EB, and S2R to increase their proportion in the dataset.

*4.2.2 Evaluation dataset of the generator.* To evaluate the performance of the generator in ChatBR, we constructed a new dataset distinct from the one used for training the detector. Figure 2 illustrates the distribution of bug reports in this new dataset. It encompasses six widely-used projects: AspectJ, Birt, Eclipse, JDT, SWT, and Tomcat [7, 8, 51]. We randomly selected 600 bug reports from each project, totaling 3,600 bug reports, which were then input

into the detector from the training phase. The detector identified 193 complete bug reports containing OB, EB, and S2R information. Due to ChatGPT's input length limitation, we removed bug reports exceeding 4,096 characters, resulting in 171 complete bug reports.

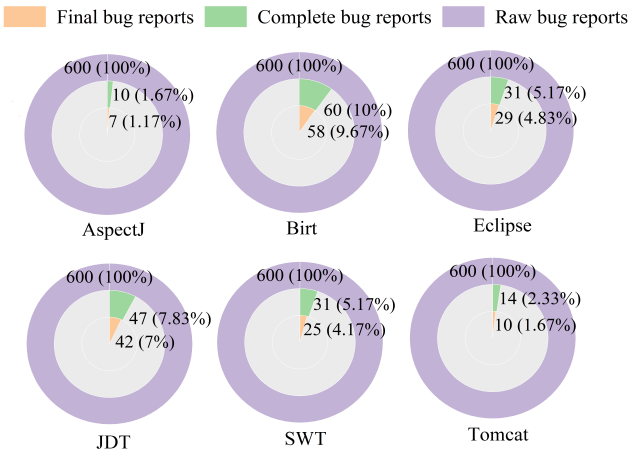


**Figure 2: Distribution of bug reports for evaluating the generator**

Using these 171 final bug reports, we synthesized incomplete reports by manually deleting OB, EB, and S2R information. This deletion strategy involved removing one or two pieces of information from OB, EB, and S2R, resulting in six variants of incomplete bug reports for each complete one. Considering the cost of running ChatGPT ($171 \times 6 \times 5 \times 5 = 25,650$) [2], we randomly selected half of the final bug reports from each project. Ultimately, 87 bug reports were used for evaluating the generator.

## 4.3 Evaluation Metrics

We employ four metrics to assess the effectiveness of ChatBR's detector: precision, accuracy, recall, and F1-score. To evaluate the generator's effectiveness, we utilize semantic similarity [24, 43], specifically cosine similarity. In our experiment, we use Word2Vec [32] to convert words into vectors. Following the prior work [29], we average the word vectors in a sentence to obtain the sentence embedding.

- **Accuracy** evaluates the proportion of correctly labeled sentences. It is calculated as:

$$Accuracy = \frac{TP + TN}{TP + FP + TN + FN} \tag{1}$$

- **Recall** measures the proportion of actual positive cases that were correctly identified. It is calculated as:

$$Recall = \frac{TP}{TP + FN} \tag{2}$$

[2]The number 171, 6, 5, 5 in this equation represent the number of bug reports, the number of variants for each bug report, the maximum number of runs per bug report, and 5-round experiments. Thus, the result 25,650 is the maximum number of ChatGPT runs in our experiments.

- **Precision** represents the fraction of true elements among the detected ones. It is defined as:

$$Precision = \frac{TP}{TP + FP} \tag{3}$$

- **F1-Score** is the harmonic mean of Recall and Precision, calculated as:

$$F1 - Score = 2 \times \left( \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \right) \tag{4}$$

- **Cosine Similarity** measures the semantic similarity between the sentence with generated missing information and the original sentence. It is calculated using:

$$Similarity(A, B) = \frac{A \cdot B}{\|A\| \|B\|} \tag{5}$$

$A \cdot B$ is the dot product of the two vectors. $\|A\|$ and $\|B\|$ is the Euclidean norm of the vector.

## 4.4 Experimental Setup

We set the following parameters to fine-tune the BERT model: hidden size is 768, and the number of attention layers is 12. All parameters are optimized using Adam, and the initial learning rate is set to 0.001. During training, the batch size is set to 64, and the maximum length of the encoder is set to 512. To optimize cost efficiency when using ChatGPT, we implemented a limit on the maximum number of runs per bug report. Specifically, for each incomplete bug report, we allowed up to five separate runs of ChatGPT to generate the desired output. Each run represents a new dialogue initiated with ChatGPT, providing a fresh attempt to produce the required information. To ensure fairness and reliability in our results, we repeated this entire process for five rounds across all experiments.

All experiments are performed on a 32G RAM Intel(R) Xeon(R) Gold 5318Y CPU@2.10GHz and an NVIDIA A30 GPU. Besides, we use the OpenAI API to interact with *GPT-3.5-turbo* in our experiments.

# 5 EXPERIMENTAL RESULTS

## 5.1 RQ1: How effective is ChatBR's detector in identifying missing information?

**Table 2: Performance of ChatBR and BEE in detecting missing information of bug reports**

| Approach | Element | Precision | Accuracy | Recall | F1-Score |
|---|---|---|---|---|---|
| BEE | OB | 0.7260 | 0.9470 | 0.8790 | 0.7956 |
| | EB | 0.7000 | 0.9920 | 0.9840 | 0.8163 |
| | S2R | 0.7200 | 0.9740 | 0.9080 | 0.8049 |
| ChatBR (without DA) | OB | 0.7400 | 0.9344 | 0.7201 | 0.7300 |
| | EB | 0.7346 | 0.9862 | 0.7022 | 0.7180 |
| | S2R | 0.7886 | 0.9564 | 0.6986 | 0.7409 |
| ChatBR | OB | 0.9798 | 0.9835 | 0.9873 | 0.9835 |
| | EB | 0.9920 | 0.9966 | 0.9984 | 0.9952 |
| | S2R | 0.9817 | 0.9900 | 0.9914 | 0.9982 |

Table 2 illustrates the performance of ChatBR and BEE in detecting missing information in bug reports. The table is divided into three sections, presenting the performance of BEE, ChatBR without data augmentation, and ChatBR with data augmentation.

Overall, ChatBR demonstrates superior performance across all metrics—precision, accuracy, recall, and F1-score—compared to BEE for different types of missing elements. Notably, ChatBR improves precision by 25.38%, 29.20%, and 26.17% for OB, EB, and S2R information, respectively. Remarkably, even without data augmentation, ChatBR's performance is competitive with BEE's optimal results.

This significant improvement can be attributed to the choice of the pre-trained BERT model. Previous research has shown that transformer-based pre-trained models excel at capturing intrinsic semantic relationships between texts [1, 16]. In contrast, the approach of converting a multi-label classification task into multiple binary classification tasks, as employed by BEE, potentially overlooks the inherent relationships between internal features of multi-label samples [44].

Furthermore, the superior performance of ChatBR with data augmentation, compared to both its non-augmented version and BEE (which uses a different data augmentation strategy), underscores the effectiveness of ChatBR's data augmentation operations. Our augmentation techniques are more targeted and domain-specific than those used in BEE.

In conclusion, the combination of fine-tuning BERT on a balanced dataset achieved through our targeted data augmentation significantly enhances ChatBR's effectiveness in detecting missing information in bug reports.

## 5.2 RQ2: How effective is ChatBR's generator in generating missing information?

*5.2.1 RQ2.1 How does the number of missing elements affect ChatBR's effectiveness?* Figure 3 illustrates the semantic similarity between ChatGPT-generated information and original information for bug reports with varying numbers of missing elements. Red triangles represent results for bug reports missing one element, while blue triangles indicate those missing two elements. For example, in the OB line, the red point shows the semantic similarity score between generated and original OB information for bug reports missing only OB. The blue point represents the average semantic similarity score for bug reports missing two elements, including those lacking both OB and EB, and those missing both OB and S2R.

Overall, ChatGPT generates missing information with high semantic similarity scores (0.7841 on average) to the original bug reports when one element is missing. There are slight differences among the six software projects. However, missing one element consistently results in higher semantic similarity between generated and original bug reports compared to missing two elements (i.e., any two of OB, EB, and S2R), as evidenced by the red triangles appearing outside the blue ones. This suggests that missing more elements may significantly affect ChatGPT's effectiveness. For instance, in the SWT project, compared to bug reports missing only one element, those missing any two elements show decreased semantic similarity scores from 0.8040, 0.8025, and 0.7725 to 0.7942, 0.7865, and 0.7611, respectively.

*5.2.2 RQ2.2 How does the type of missing elements affect ChatBR's effectiveness?* Table 3 presents the semantic similarity between ChatGPT-generated elements and original elements for bug reports with varying types of missing elements across six software projects. The first column lists various scenarios for generating OB, EB, and

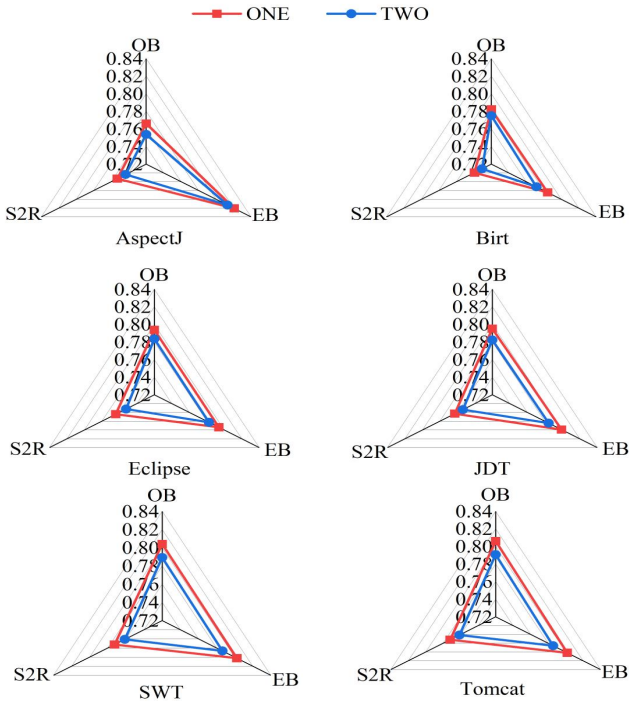


**Figure 3: Semantic similarity between ChatGPT-generated information and original information for bug reports with varying numbers of missing elements**

S2R using ChatGPT. For instance, the "OB" row shows the semantic similarity between the generated and original OB when only OB is missing. The "OB_EB(OB)" row indicates the semantic similarity between the generated and original OB when both OB and EB are missing.

**Table 3: Semantic similarity between ChatGPT-generated missing elements and original elements for bug reports with varying types of missing elements**

| Scenario | AspectJ | Birt | Eclipse | JDT | SWT | Tomcat |
|---|---|---|---|---|---|---|
| OB | 0.7661 | 0.7822 | 0.7938 | 0.7952 | 0.8040 | 0.8061 |
| OB_EB(OB) | 0.7535 | 0.7772 | 0.7872 | 0.7842 | 0.7950 | 0.7964 |
| OB_S2R(OB) | 0.7541 | 0.7729 | 0.7801 | 0.7805 | 0.7835 | 0.7850 |
| OB(avg) | 0.7579 | 0.7774 | 0.7870 | 0.7867 | 0.7942 | 0.7958 |
| EB | 0.8207 | 0.7844 | 0.7938 | 0.7992 | 0.8025 | 0.8019 |
| OB_EB(EB) | 0.8149 | 0.7714 | 0.7837 | 0.7817 | 0.7833 | 0.7815 |
| EB_S2R(EB) | 0.8110 | 0.7718 | 0.7813 | 0.7872 | 0.7896 | 0.7899 |
| EB(avg) | 0.8155 | 0.7758 | 0.7863 | 0.7894 | 0.7918 | 0.7911 |
| S2R | 0.7530 | 0.7393 | 0.7642 | 0.7629 | 0.7725 | 0.7720 |
| OB_S2R(S2R) | 0.7429 | 0.7336 | 0.7500 | 0.7510 | 0.7574 | 0.7574 |
| EB_S2R(S2R) | 0.7440 | 0.7276 | 0.7547 | 0.7563 | 0.7649 | 0.7654 |
| S2R(avg) | 0.7466 | 0.7335 | 0.7563 | 0.7568 | 0.7650 | 0.7649 |

The data reveals that EB achieves the highest average semantic similarity (0.7917), followed by OB (0.7832) and S2R (0.7538). This suggests that OB and EB are relatively easier for ChatGPT to generate, while S2R presents the greatest challenge. Two factors

contribute to the ease of generating OB and EB: (1) OB and EB typically express opposite meanings in bug reports, allowing inference from one to the other; (2) S2R, which describes user actions when discovering a bug, often implies OB and EB, aiding ChatGPT in their generation. Conversely, S2R generation is more difficult due to: (1) the limited training of most LLMs on software bug reports, resulting in better performance on generic tasks but poorer results on bug report generation; (2) the complexity and diversity of software usage scenarios, which complicate accurate reasoning about user actions.

Interestingly, ChatGPT's effectiveness in generating elements varies across software projects. For example, the semantic similarity of generated OB ranges from 0.7661 for AspectJ to 0.8061 for SWT.

## 5.3 RQ3: How efficient is ChatBR in generating missing information?

*5.3.1 RQ 3.1: How does the number of missing information affect ChatBR's efficiency?* Figure 4 illustrates the number of ChatGPT runs required to generate missing information for bug reports lacking one or two elements in a 5-round experiment. The x-axis represents the number of ChatGPT runs, with a maximum of 5. The y-axis shows the number of bug reports for which ChatGPT successfully generated missing information. Note that the total number of bug reports is 435 ($87 \times 5 = 435$) due to the 5-round experimental design.

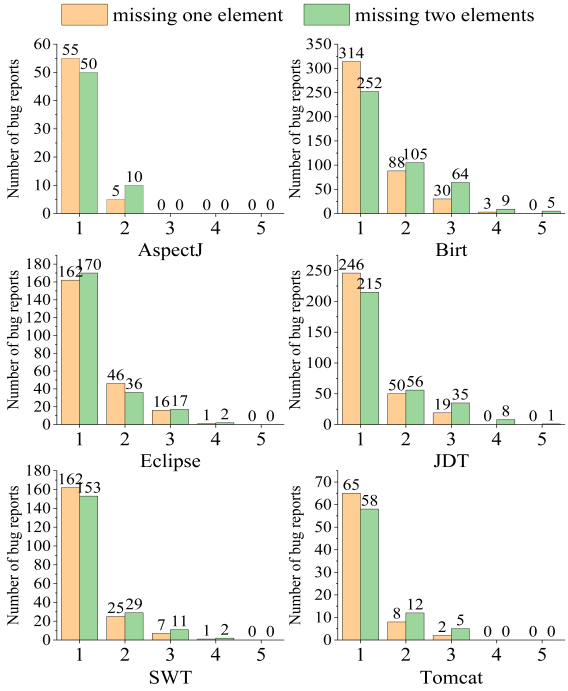


**Figure 4: Number of ChatGPT runs required to generate missing information for bug reports with one or two absent elements**

The results demonstrate that ChatBR can generate missing information for 99.62% of bug reports missing one element within three ChatGPT runs. Moreover, ChatBR successfully generates missing information for approximately 76.93% of these bug reports in a single run. For bug reports missing two elements, ChatBR generates the missing information for 97.93% within three runs, a slight decrease of 1.69% compared to the single-element case. Additionally, ChatBR generates missing information for about 68.81% of two-element missing reports in a single run, an 8.12% decrease compared to the single-element scenario. These findings indicate that as the number of missing elements in bug reports increases, more ChatGPT runs are required to generate the missing information. In other words, when more information is absent from bug reports, ChatGPT has less context to work with, which impacts ChatBR's efficiency.

*5.3.2 RQ3.2: How does the type of missing information affect ChatBR's efficiency?* Figure 5 illustrates the proportion of bug reports improved by ChatBR after a single ChatGPT run for various missing element types. Figure 5(a) presents results for bug reports missing one element type, while Figure 5(b) shows results for those missing two element types. The x-axis represents different software projects, and the y-axis indicates the proportion of improved bug reports.

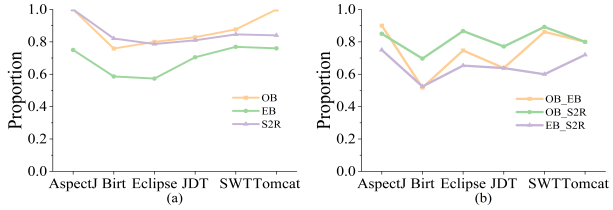


**Figure 5: Proportion of bug reports enhanced by ChatBR using a single ChatGPT run, categorized by missing element types**

Figure 5(a) reveals that, overall, the proportion of improved bug reports missing EB is lower than those missing OB or S2R. This suggests that ChatBR faces greater challenges in generating EB with a single ChatGPT run compared to OB and S2R. In other words, for bug reports missing only one element, absent EB information has the most significant impact on ChatBR's efficiency. Additionally, the number of ChatGPT runs required to generate different element types varies within the same software project. For instance, in the SWT project, the proportions of missing OB, EB, and S2R that can be generated by a single ChatGPT run are 87.69%, 76.92%, and 84.62%, respectively.

Figure 5(b) demonstrates that the proportion of improved bug reports missing both EB and S2R is the lowest, followed by those missing both OB and EB. Bug reports missing both OB and S2R show the highest proportion of improvement. This indicates that for bug reports missing two elements, the absence of both EB and S2R has the most substantial impact on ChatBR's efficiency.

## 5.4 RQ4: How do different prompts affect the generation results?

Listings 5 and 6 present the basic Prompt Template 1 and the improved Prompt Template 2, respectively. Prompt Template 1 succinctly outlines the tasks, requirements, and expected output format for ChatGPT. Prompt Template 2 builds upon this foundation by assigning ChatGPT the role of a senior software engineer adept at writing and modifying bug reports, aligning with our task of bug report quality improvement.

**Listing 5: Prompt Template 1**

**Prompt Template 1:**
– Your task is to infer the appropriate details based on the context and supplement the bug report to ensure it contains clear and complete OB(Observed Behavior), EB(Expected Behavior), and S2R(Steps to Reproduce) sentences. Also, improve the wording of these sentences for clarity where possible.
– Respond in JSON format as follows:
{"id": "", "title": "", "description": "", "OB": "", "EB": "", "S2R": ""}
**<BUG REPORT>...</BUG REPORT>**

**Listing 6: Prompt Template 2**

**Prompt Template 2:**
– Your role is a senior software engineer, and you are very good at analyzing and writing bug reports. The bug report may lack sufficient details in the OB(Observed Behavior), EB(Expected Behavior), and S2R(Steps to Reproduce).
– Your task is to infer the appropriate details based on the context and supplement the bug report to ensure it contains clear and complete OB/EB/S2R sentences. Also, improve the wording of these sentences for clarity where possible.
– Respond in JSON format as follows:
{"id": "", "title": "", "description": "", "OB": "", "EB": "", "S2R": ""}
**<BUG REPORT>...</BUG REPORT>**

The final prompt template, shown in Listing 4, incorporates best practices for ChatGPT usage and includes definitions of OB, EB, and S2R based on Prompt Template 2. These additions aim to enhance ChatGPT's understanding of crucial information, potentially leading to improved results.

Figure 6 illustrates the effectiveness of these prompt templates in terms of semantic similarity between generated and original information. The orange, green, and purple polylines represent Prompt Template 1, Prompt Template 2, and the final prompt template, respectively. Across all six software projects, the purple polyline consistently appears above the others, demonstrating the superior performance of the final prompt template. This suggests that providing specific definitions of OB, EB, and S2R enables ChatGPT to better understand its task and generate results more closely aligned with the ground truth.

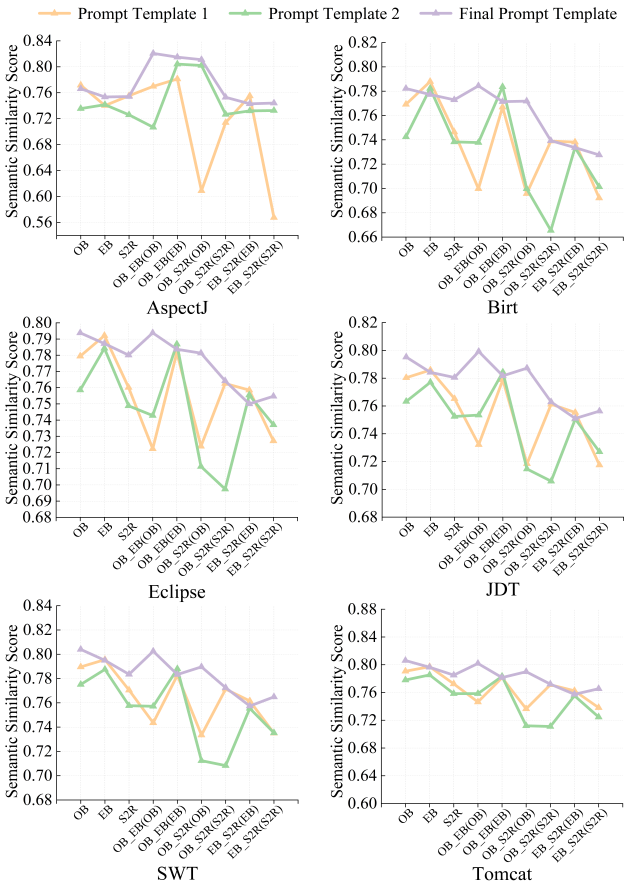


**Figure 6: Effectiveness of different prompts**

Table 4 compares the efficiency of different prompt templates by measuring the number of ChatGPT runs required. These results are aggregated from 5-round experiments, with "$rn, n \in \{1, 2..., 5\}$" representing the total number of ChatGPT runs for each prompt. The final prompt template consistently achieves the desired output within three runs, while Prompt Template 1 and Prompt Template 2 often require up to five runs. Notably, for the AspectJ project, the final prompt template generates the missing information in just two runs. Prompt Template 2 shows a slight improvement over Prompt Template 1, as evidenced by its ability to generate missing information within three runs for the Tomcat project.

## 5.5 RQ5: How effective is ChatGPT compared with other LLMs?

Table 5 compares the effectiveness of ChatGPT with two other LLMs, Llama2-7B and Vicuna-7B, in terms of semantic similarity between generated and original information. For each project, the table presents semantic similarity scores for scenarios where one element (Columns 3-5) or two elements (Columns 6-11) are missing. Specifically, column 6 shows the semantic similarity between generated and original OB when both OB and EB are missing.

**Table 4: Efficiency of different prompts**

| Project | Scenario | Prompt Template 1 | | | | | Prompt Template 2 | | | | | Final Prompt Template | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | r1 | r2 | r3 | r4 | r5 | r1 | r2 | r3 | r4 | r5 | r1 | r2 | r3 | r4 | r5 |
| AspectJ | OB | 17 | 2 | 0 | 1 | 0 | 18 | 2 | 0 | 0 | 0 | 20 | 0 | 0 | 0 | 0 |
| | EB | 14 | 4 | 0 | 0 | 2 | 14 | 4 | 1 | 0 | 1 | 15 | 5 | 0 | 0 | 0 |
| | S2R | 12 | 5 | 2 | 0 | 1 | 15 | 5 | 0 | 0 | 0 | 20 | 0 | 0 | 0 | 0 |
| | OB_EB | 15 | 4 | 1 | 0 | 0 | 18 | 1 | 1 | 0 | 0 | 18 | 2 | 0 | 0 | 0 |
| | OB_S2R | 15 | 4 | 0 | 0 | 1 | 17 | 1 | 1 | 1 | 0 | 17 | 3 | 0 | 0 | 0 |
| | EB_S2R | 9 | 2 | 3 | 4 | 2 | 14 | 1 | 0 | 0 | 5 | 15 | 5 | 0 | 0 | 0 |
| Birt | OB | 114 | 18 | 8 | 1 | 4 | 127 | 14 | 3 | 1 | 0 | 110 | 30 | 5 | 0 | 0 |
| | EB | 89 | 22 | 13 | 7 | 14 | 93 | 20 | 10 | 5 | 17 | 85 | 34 | 23 | 3 | 0 |
| | S2R | 95 | 28 | 9 | 5 | 8 | 119 | 17 | 5 | 2 | 2 | 119 | 24 | 2 | 0 | 0 |
| | OB_EB | 103 | 27 | 11 | 1 | 3 | 126 | 15 | 2 | 1 | 1 | 75 | 41 | 23 | 4 | 2 |
| | OB_S2R | 103 | 26 | 11 | 4 | 1 | 123 | 16 | 4 | 2 | 0 | 101 | 32 | 10 | 1 | 1 |
| | EB_S2R | 78 | 33 | 10 | 6 | 18 | 79 | 29 | 11 | 11 | 15 | 76 | 32 | 31 | 4 | 2 |
| Eclipse | OB | 62 | 11 | 1 | 0 | 1 | 67 | 7 | 0 | 0 | 1 | 60 | 13 | 2 | 0 | 0 |
| | EB | 44 | 17 | 6 | 2 | 6 | 52 | 7 | 9 | 3 | 4 | 43 | 20 | 12 | 0 | 0 |
| | S2R | 53 | 12 | 6 | 3 | 1 | 63 | 8 | 4 | 0 | 0 | 59 | 13 | 2 | 1 | 0 |
| | OB_EB | 54 | 15 | 1 | 4 | 1 | 67 | 3 | 4 | 1 | 0 | 56 | 12 | 7 | 0 | 0 |
| | OB_S2R | 66 | 9 | 0 | 0 | 0 | 69 | 5 | 0 | 1 | 0 | 65 | 8 | 2 | 0 | 0 |
| | EB_S2R | 38 | 14 | 5 | 6 | 12 | 51 | 14 | 4 | 5 | 1 | 49 | 16 | 8 | 2 | 0 |
| JDT | OB | 86 | 17 | 1 | 1 | 0 | 94 | 10 | 1 | 0 | 0 | 87 | 15 | 3 | 0 | 0 |
| | EB | 65 | 29 | 4 | 2 | 5 | 76 | 16 | 4 | 2 | 7 | 74 | 20 | 11 | 0 | 0 |
| | S2R | 71 | 16 | 4 | 3 | 11 | 91 | 9 | 4 | 1 | 0 | 85 | 15 | 5 | 0 | 0 |
| | OB_EB | 82 | 20 | 2 | 0 | 1 | 90 | 14 | 1 | 0 | 0 | 67 | 23 | 14 | 1 | 0 |
| | OB_S2R | 92 | 7 | 4 | 1 | 1 | 93 | 12 | 0 | 0 | 0 | 81 | 15 | 9 | 0 | 0 |
| | EB_S2R | 52 | 30 | 8 | 3 | 12 | 67 | 15 | 6 | 3 | 14 | 67 | 18 | 12 | 7 | 1 |
| SWT | OB | 55 | 10 | 0 | 0 | 0 | 60 | 5 | 0 | 0 | 0 | 57 | 8 | 0 | 0 | 0 |
| | EB | 43 | 10 | 5 | 2 | 5 | 50 | 11 | 1 | 2 | 1 | 50 | 8 | 6 | 1 | 0 |
| | S2R | 47 | 10 | 3 | 1 | 4 | 56 | 5 | 3 | 1 | 0 | 55 | 9 | 1 | 0 | 0 |
| | OB_EB | 57 | 8 | 0 | 0 | 0 | 57 | 7 | 1 | 0 | 0 | 56 | 9 | 0 | 0 | 0 |
| | OB_S2R | 55 | 7 | 2 | 0 | 1 | 59 | 5 | 1 | 0 | 0 | 58 | 7 | 0 | 0 | 0 |
| | EB_S2R | 37 | 13 | 5 | 0 | 10 | 51 | 9 | 4 | 0 | 1 | 39 | 13 | 11 | 2 | 0 |
| Tomcat | OB | 22 | 1 | 2 | 0 | 0 | 25 | 0 | 0 | 0 | 0 | 25 | 0 | 0 | 0 | 0 |
| | EB | 16 | 5 | 2 | 2 | 0 | 17 | 6 | 2 | 0 | 0 | 19 | 5 | 1 | 0 | 0 |
| | S2R | 13 | 3 | 1 | 1 | 7 | 24 | 1 | 0 | 0 | 0 | 21 | 3 | 1 | 0 | 0 |
| | OB_EB | 18 | 3 | 2 | 1 | 1 | 20 | 5 | 0 | 0 | 0 | 20 | 5 | 0 | 0 | 0 |
| | OB_S2R | 23 | 1 | 0 | 1 | 0 | 23 | 1 | 1 | 0 | 0 | 20 | 4 | 1 | 0 | 0 |
| | EB_S2R | 14 | 6 | 1 | 2 | 2 | 16 | 7 | 2 | 0 | 0 | 18 | 3 | 4 | 0 | 0 |

*Note: "r*n*" is the abbreviation of "run *n* times"

**Table 5: Effectiveness of different LLMs**

| LLM | Project | OB | EB | S2R | OB_EB | | OB_S2R | | EB_S2R | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | OB | EB | OB | S2R | EB | S2R |
| Llama2-7B | AspectJ | 0.0847 | 0.0000 | 0.0419 | 0.0264 | 0.0382 | 0.0282 | 0.0422 | 0.0000 | 0.0000 |
| | Birt | 0.1685 | 0.0527 | 0.0808 | 0.0770 | 0.0867 | 0.0298 | 0.0325 | 0.0399 | 0.0292 |
| | Eclipse | 0.1748 | 0.0309 | 0.0669 | 0.0790 | 0.0731 | 0.0557 | 0.0510 | 0.0000 | 0.0000 |
| | JDT | 0.1036 | 0.1071 | 0.0547 | 0.0652 | 0.0549 | 0.0177 | 0.0158 | 0.0237 | 0.0250 |
| | SWT | 0.1762 | 0.0738 | 0.1613 | 0.0761 | 0.0736 | 0.0438 | 0.0409 | 0.1213 | 0.1186 |
| | Tomcat | 0.2474 | 0.0400 | 0.0000 | 0.1974 | 0.1831 | 0.1372 | 0.1435 | 0.1580 | 0.1580 |
| Vicuna-7B | AspectJ | 0.1505 | 0.0000 | 0.0000 | 0.2387 | 0.2225 | 0.0000 | 0.0000 | 0.2197 | 0.1744 |
| | Birt | 0.3403 | 0.2056 | 0.0971 | 0.3182 | 0.2850 | 0.3811 | 0.3490 | 0.1848 | 0.1771 |
| | Eclipse | 0.4179 | 0.1611 | 0.0000 | 0.3723 | 0.3806 | 0.2469 | 0.2503 | 0.2657 | 0.2730 |
| | JDT | 0.2448 | 0.1437 | 0.0274 | 0.3452 | 0.3062 | 0.1629 | 0.1245 | 0.1527 | 0.1390 |
| | SWT | 0.3339 | 0.1704 | 0.0581 | 0.4331 | 0.3658 | 0.0676 | 0.0595 | 0.3084 | 0.2977 |
| | Tomcat | 0.1732 | 0.1474 | 0.0000 | 0.3068 | 0.3037 | 0.0000 | 0.0000 | 0.1567 | 0.1631 |
| ChatGPT | AspectJ | 0.7661 | 0.8207 | 0.7530 | 0.7535 | 0.8149 | 0.7541 | 0.7429 | 0.8110 | 0.7440 |
| | Birt | 0.7822 | 0.7844 | 0.7393 | 0.7772 | 0.7714 | 0.7729 | 0.7336 | 0.7718 | 0.7276 |
| | Eclipse | 0.7938 | 0.7938 | 0.7642 | 0.7872 | 0.7837 | 0.7801 | 0.7500 | 0.7813 | 0.7547 |
| | JDT | 0.7952 | 0.7992 | 0.7629 | 0.7842 | 0.7817 | 0.7805 | 0.7510 | 0.7872 | 0.7563 |
| | SWT | 0.8040 | 0.8025 | 0.7725 | 0.7950 | 0.7833 | 0.7835 | 0.7574 | 0.7896 | 0.7649 |
| | Tomcat | 0.8061 | 0.8019 | 0.7720 | 0.7964 | 0.7815 | 0.7850 | 0.7574 | 0.7899 | 0.7654 |

ChatGPT significantly outperforms both Llama2-7B and Vicuna-7B. The maximum semantic similarity score for ChatGPT-generated information reaches 0.8207, compared to 0.2474 for Llama2-7B and 0.4179 for Vicuna-7B. Upon closer examination, we found that the outputs from Vicuna-7B and Llama2-7B often deviate from the required JSON format specified in the prompts. Moreover, some of their outputs merely replicate the input prompts. We hypothesize that the inferior performance of Vicuna-7B and Llama2-7B may be attributed to their smaller model sizes, which likely impede their ability to effectively process and respond to complex bug reports.

# 6 THREATS TO VALIDITY

**Internal Threats.** Internal threats to our study relate to the data leakage of ChatGPT. However, there is no result that the generated information is the same as the original information in our experiment. This indicates that ChatGPT does not rely only on the memory for the training data. In fact, ChatGPT has a strong reasoning ability based on the relevant project information in similar domains or target projects. It can reason out the missing information related to bug reports accurately. In fact, fine-tuning LLMs can further improve the effectiveness and efficiency of generating missing information, which has been studied in other fields [22, 38].

**External Threats.** The main external threat to our study is the generality of ChatBR. Our experiment is conducted on a dataset of bug reports from six publicly available software projects. The limitation of the research domain and types of bug reports may affect the effectiveness of bug report assessment and improvement, especially for specific types of bug reports. Nevertheless, our approach is generic and can be extended to other software projects.

# 7 RELATED WORK

## 7.1 Assessing Bug Report Quality

The quality of bug reports has a great impact on a series of software testing activities, such as bug localization and bug fixing. Thus, there have been some studies on bug report quality assessment. Most of them utilize Machine Learning (ML) or heuristic rules to classify the quality of bug reports by extracting all kinds of metrics related to bug reports. Fan et al. [17] extracted features from 5 dimensions (i.e., the reporter experience, collaboration network, completeness, readability, and text) and used a random forest classifier to identify valid bug reports. Zimmermann et al. [53] designed CUEZILLA, a prototype tool to assess bug report quality by selecting the information that the bug fixer expects the user to provide as features and using supervised machine learning algorithms to train a prediction model. Besides, it provides recommendations to bug reporters to make better bug reports. Different from CUEZILLA, Karim et al. [25] built and assessed classification models using four different text classification techniques to predict key features from historical bug-fixing knowledge. Chaparro et al. [9] developed three versions of DeMIBuD using regular expressions, heuristic rules and Natural Language Processing (NLP), and Machine Learning (ML) to detect missing OB, EB, and S2R elements automatically. Unlike the above approaches, we assess the quality of bug reports by detecting the absence of crucial information in bug reports using a fine-tuned BERT pre-trained model. The goal of this strategy is to leverage the performance and advanced features of BERT to improve the efficiency and effectiveness of bug report quality assessment.

## 7.2 Improving Bug Report Quality

Existing works on improving the quality of bug reports mainly focus on improving or adding crucial information entries, such as replication steps, execution trajectories, and problem descriptions. Feng et al. [18] proposed AdbGPT, which makes use of few-shot and Chain of Thought (CoT) [41] to generate S2Rs in a developer-like manner by feeding XML textual information into LLMs. Zhao et al. [50] proposed ReCDroid+ based on the previous ReCDroid,

which employed an HTML parser [31], CNN (Convolutional Neural Network) [26], and LSTM (Long Short-Term Memory Network) [19] to extract crashes and S2R sentences. In addition, improving the bug report title has attracted much attention from researchers. For example, Chen et al. [12] proposed iTAPE, which utilizes an approach based on the Seq2Seq model [37] to generate report titles. Zhang et al. [47] proposed to generate titles for bug reports by fine-tuning the BART [28] model. Different from the above approaches, our approach is to improve the quality of reports by generating the missing crucial information in bug reports based on existing information and predefined prompt templates. The goal of this strategy is to leverage the outperforming capacity of ChatGPT for understanding and generating natural language text that is special to our task and requirement.

## 8 CONCLUSION

In this work, we focus on using ChatGPT to improve the quality of bug reports. Given the cost limitation of using ChatGPT, we aim to minimize the number of ChatGPT runs by assessing the quality of bug reports accurately. We propose ChatBR, a two-step strategy to improve the quality of bug reports that combines a pre-trained model and ChatGPT to generate missing OB, EB, and S2R information. The experimental results show that ChatBR significantly outperforms the baseline method in assessing bug report quality, showing an improvement of 25.38% to 29.20% in precision. In the task of generating missing information, ChatBR achieves an average semantic similarity of 77.62% between the generated content and the original information across six diverse real-world software projects.

In the future, we plan to conduct a survey with expert developers to further explore the reliability and validity of ChatBR for other downstream tasks, e.g., bug localization [6, 33], bug fixing [52], and duplicate bug report detection [46, 48]. In addition, we plan to investigate the performance of LLMs in improving the quality of bug reports in the few-shot and CoT settings.

## ACKNOWLEDGMENTS

We thank all the anonymous reviewers for their constructive comments and valuable suggestions as well as all the teachers and students who participated in this experiment. This research is supported by the National Natural Science Foundation of China (No. 62202414, No. 61972335 and No. 62002309), the Open Foundation of Yunnan Key Laboratory of Software Engineering (No. 2023SE201), the Six Talent Peaks Project in Jiangsu Province (No. RJFW-053); the Jiangsu "333" Project and Yangzhou University Top-level Talents Support Program (2019), Postgraduate Research & Practice Invocation Program of Jiangsu Province (No. KYCX23_3563).

## REFERENCES

[1] Wasi Uddin Ahmad, Saikat Chakraborty, Baishakhi Ray, and Kai-Wei Chang. 2021. Unified Pre-training for Program Understanding and Generation. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2021, Online, June 6-11, 2021*, Kristina Toutanova, Anna Rumshisky, Luke Zettlemoyer, Dilek Hakkani-Tür, Iz Beltagy, Steven Bethard, Ryan Cotterell, Tanmoy Chakraborty, and Yichao Zhou (Eds.). Association for Computational Linguistics, 2655–2668. https://doi.org/10.18653/V1/2021.NAACL-MAIN.211

[2] AspectJ. 2023. AspectJ. https://bugs.eclipse.org/bugs/show_bug.cgi?id=384398. 2023.

[3] Yejin Bang, Samuel Cahyawijaya, Nayeon Lee, Wenliang Dai, Dan Su, Bryan Wilie, Holy Lovenia, Ziwei Ji, Tiezheng Yu, Willy Chung, et al. 2023. A multitask, multilingual, multimodal evaluation of chatgpt on reasoning, hallucination, and interactivity. *arXiv preprint arXiv:2302.04023* (2023).

[4] Nicolas Bettenburg, Sascha Just, Adrian Schröter, Cathrin Weiss, Rahul Premraj, and Thomas Zimmermann. 2008. What makes a good bug report?. In *Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of software engineering*. 308–318.

[5] Steven Bird and Edward Loper. 2004. NLTK: The Natural Language Toolkit. In *Proceedings of the 42nd Annual Meeting of the Association for Computational Linguistics, Barcelona, Spain, July 21-26, 2004 - Poster and Demonstration*. ACL. https://aclanthology.org/P04-3031/

[6] Lili Bo, Yue Li, Xiaobing Sun, Xiaoxue Wu, and Bin Li. 2023. VulLoc: vulnerability localization based on inducing commits and fixing commits. *Frontiers Comput. Sci.* 17, 3 (2023), 173207. https://doi.org/10.1007/S11704-022-1729-X

[7] Junming Cao, Shouliang Yang, Wenhui Jiang, Hushuang Zeng, Beijun Shen, and Hao Zhong. 2021. BugPecker: locating faulty methods with deep learning on revision graphs. In *Proceedings of the 35th IEEE/ACM International Conference on Automated Software Engineering* (Virtual Event, Australia) *(ASE '20)*. Association for Computing Machinery, New York, NY, USA, 1214–1218. https://doi.org/10.1145/3324884.3418934

[8] Partha Chakraborty, Mahmoud Alfadel, and Meiyappan Nagappan. 2024. RLocator: Reinforcement Learning for Bug Localization. *IEEE Transactions on Software Engineering* (2024), 1–14. https://doi.org/10.1109/TSE.2024.3452595

[9] Oscar Chaparro, Jing Lu, Fiorella Zampetti, Laura Moreno, Massimiliano Di Penta, Andrian Marcus, Gabriele Bavota, and Vincent Ng. 2017. Detecting missing information in bug descriptions. In *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*. 396–407.

[10] ChatGPT. 2023. ChatGPT. https://openai.com/blog/chatgpt. 2023.

[11] ChatGPT. 2023. ChatGPT. https://platform.openai.com/docs/guides/prompt-engineering. 2023.

[12] Songqiang Chen, Xiaoyuan Xie, Bangguo Yin, Yuanxiang Ji, Lin Chen, and Baowen Xu. 2020. Stay professional and efficient: automatically generate titles for your bug reports. In *Proceedings of the 35th IEEE/ACM International Conference on Automated Software Engineering*. 385–397.

[13] Xin Chen, He Jiang, Xiaochen Li, Tieke He, and Zhenyu Chen. 2018. Automated quality assessment for crowdsourced test reports of mobile applications. In *2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 368–379.

[14] Wei-Lin Chiang, Zhuohan Li, Zi Lin, Ying Sheng, Zhanghao Wu, Hao Zhang, Lianmin Zheng, Siyuan Zhuang, Yonghao Zhuang, Joseph E Gonzalez, et al. 2023. Vicuna: An open-source chatbot impressing gpt-4 with 90%* chatgpt quality. *See https://vicuna. lmsys. org (accessed 14 April 2023)* 2, 3 (2023), 6.

[15] Agnieszka Ciborowska and Kostadin Damevski. 2023. Too Few Bug Reports? Exploring Data Augmentation for Improved Changeset-based Bug Localization. *arXiv preprint arXiv:2305.16430* (2023).

[16] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805* (2018).

[17] Yuanrui Fan, Xin Xia, David Lo, and Ahmed E. Hassan. 2020. Chaff from the Wheat: Characterizing and Determining Valid Bug Reports. *IEEE Transactions on Software Engineering* 46, 5 (2020), 495–525.

[18] Sidong Feng and Chunyang Chen. 2023. Prompting Is All Your Need: Automated Android Bug Replay with Large Language Models. *arXiv preprint arXiv:2306.01987* (2023).

[19] Felix A Gers, Jürgen Schmidhuber, and Fred Cummins. 2000. Learning to forget: Continual prediction with LSTM. *Neural computation* 12, 10 (2000), 2451–2471.

[20] Rui Hao, Yang Feng, James A Jones, Yuying Li, and Zhenyu Chen. 2019. CTRAS: Crowdsourced test report aggregation and summarization. In *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*. IEEE, 900–911.

[21] Jianjun He, Ling Xu, Yuanrui Fan, Zhou Xu, Meng Yan, and Yan Lei. 2020. Deep learning based valid bug reports determination and explanation. In *2020 IEEE 31st International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 184–194.

[22] Jeremy Howard and Sebastian Ruder. 2018. Universal language model fine-tuning for text classification. *arXiv preprint arXiv:1801.06146* (2018).

[23] Vikramaditya Jakkula. 2006. Tutorial on support vector machine (svm). *School of EECS, Washington State University* 37, 2.5 (2006), 3.

[24] Derry Jatnika, Moch Arif Bijaksana, and Arie Ardiyanti Suryani. 2019. Word2vec model analysis for semantic similarities in english words. *Procedia Computer Science* 157 (2019), 160–167.

[25] Md Rejaul Karim, Akinori Ihara, Eunjong Choi, and Hajimu Iida. 2019. Identifying and predicting key features to support bug reporting. *Journal of Software: Evolution and Process* 31, 12 (2019), e2184.

[26] Yoon Kim. 2014. Convolutional neural networks for sentence classification. *arXiv preprint arXiv:1408.5882* (2014).

[27] Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. 2022. Large language models are zero-shot reasoners. *Advances in neural information processing systems* 35 (2022), 22199–22213.

[28] Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Ves Stoyanov, and Luke Zettlemoyer. 2019. Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. *arXiv preprint arXiv:1910.13461* (2019).

[29] Haixia Liu. 2017. Sentiment analysis of citations using word2vec. *arXiv preprint arXiv:1704.00177* (2017).

[30] Pengfei Liu, Weizhe Yuan, Jinlan Fu, Zhengbao Jiang, Hiroaki Hayashi, and Graham Neubig. 2023. Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing. *Comput. Surveys* 55, 9 (2023), 1–35.

[31] lxml. 2023. lxml. https://lxml.de/tutorial.html. 2014.

[32] Tomás Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient Estimation of Word Representations in Vector Space. In *1st International Conference on Learning Representations, ICLR 2013, Scottsdale, Arizona, USA, May 2-4, 2013, Workshop Track Proceedings*, Yoshua Bengio and Yann LeCun (Eds.). http://arxiv.org/abs/1301.3781

[33] Zhen Ni, Lili Bo, Bin Li, Tianhao Chen, Xiaobing Sun, and Xiaoxue Wu. 2022. An approach of method-level bug localization. *IET Software* 16, 4 (2022), 422–437.

[34] Roberto Poli, Michael Healy, and Achilles Kameas. 2010. *Theory and applications of ontology: Computer applications.* Springer.

[35] Advait Sarkar, Andrew D Gordon, Carina Negreanu, Christian Poelitz, Sruti Srinivasa Ragavan, and Ben Zorn. 2022. What is it like to program with artificial intelligence? *arXiv preprint arXiv:2208.06213* (2022).

[36] Yang Song and Oscar Chaparro. 2020. Bee: A tool for structuring and analyzing bug reports. In *Proceedings of the 28th ACM joint meeting on european software engineering conference and symposium on the foundations of software engineering*. 1551–1555.

[37] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. 2014. Sequence to Sequence Learning with Neural Networks. In *Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014, December 8-13 2014, Montreal, Quebec, Canada*, Zoubin Ghahramani, Max Welling, Corinna Cortes, Neil D. Lawrence, and Kilian Q. Weinberger (Eds.). 3104–3112. https://proceedings.neurips.cc/paper/2014/hash/a14ac55a4f27472c5d894ec1c3c743d2-Abstract.html

[38] Robert Tinn, Hao Cheng, Yu Gu, Naoto Usuyama, Xiaodong Liu, Tristan Naumann, Jianfeng Gao, and Hoifung Poon. 2023. Fine-tuning large neural language models for biomedical natural language processing. *Patterns* 4, 4 (2023).

[39] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. 2023. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288* (2023).

[40] Yaqing Wang, Quanming Yao, James T Kwok, and Lionel M Ni. 2020. Generalizing from a few examples: A survey on few-shot learning. *ACM computing surveys (csur)* 53, 3 (2020), 1–34.

[41] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed H. Chi, Quoc V. Le, and Denny Zhou. 2022. Chain-of-Thought Prompting Elicits Reasoning in Large Language Models. In *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*.

[42] Jason Wei and Kai Zou. 2019. Eda: Easy data augmentation techniques for boosting performance on text classification tasks. *arXiv preprint arXiv:1901.11196* (2019).

[43] Xinli Yang, David Lo, Xin Xia, Lingfeng Bao, and Jianling Sun. 2016. Combining Word Embedding with Information Retrieval to Recommend Similar Bug Reports. In *2016 IEEE 27th International Symposium on Software Reliability Engineering (ISSRE)*. 127–137. https://doi.org/10.1109/ISSRE.2016.33

[44] Min-Ling Zhang, Yu-Kun Li, Xu-Ying Liu, and Xin Geng. 2018. Binary relevance for multi-label learning: an overview. *Frontiers of Computer Science* 12 (2018), 191–202.

[45] Tao Zhang, Jiachi Chen, He Jiang, Xiapu Luo, and Xin Xia. 2017. Bug report enrichment with application of automated fixer recommendation. In *2017 IEEE/ACM 25th International Conference on Program Comprehension (ICPC)*. IEEE, 230–240.

[46] Ting Zhang, DongGyun Han, Venkatesh Vinayakarao, Ivana Clairine Irsan, Bowen Xu, Ferdian Thung, David Lo, and Lingxiao Jiang. 2023. Duplicate bug report detection: How far are we? *ACM Transactions on Software Engineering and Methodology* 32, 4 (2023), 1–32.

[47] Ting Zhang, Ivana Clairine Irsan, Ferdian Thung, DongGyun Han, David Lo, and Lingxiao Jiang. 2022. iTiger: an automatic issue title generation tool. In *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 1637–1641.

[48] Ting Zhang, Ivana Clairine Irsan, Ferdian Thung, and David Lo. 2023. Cupid: Leveraging chatgpt for more accurate duplicate bug report detection. *arXiv preprint arXiv:2308.10022* (2023).

[49] Ting Zhang, Ivana Clairine Irsan, Ferdian Thung, and David Lo. 2024. Revisiting Sentiment Analysis for Software Engineering in the Era of Large Language Models. *ACM Trans. Softw. Eng. Methodol.* (Sept. 2024). https://doi.org/10.1145/3697009 Just Accepted.

[50] Yu Zhao, Ting Su, Yang Liu, Wei Zheng, Xiaoxue Wu, Ramakanth Kavuluru, William GJ Halfond, and Tingting Yu. 2022. Recdroid+: Automated end-to-end crash reproduction from bug reports for android apps. *ACM Transactions on Software Engineering and Methodology (TOSEM)* 31, 3 (2022), 1–33.

[51] Jian Zhou, Hongyu Zhang, and David Lo. 2012. Where should the bugs be fixed? More accurate information retrieval-based bug localization based on bug reports. In *2012 34th International Conference on Software Engineering (ICSE)*. 14–24. https://doi.org/10.1109/ICSE.2012.6227210

[52] Zhou Zhou, Lili Bo, Xiaoxue Wu, Xiaobing Sun, Tao Zhang, Bin Li, Jiale Zhang, and Sicong Cao. 2022. SPVF: security property assisted vulnerability fixing via attention-based models. *Empirical Software Engineering* 27, 7 (2022), 171.

[53] Thomas Zimmermann, Rahul Premraj, Nicolas Bettenburg, Sascha Just, Adrian Schroter, and Cathrin Weiss. 2010. What makes a good bug report? *IEEE Transactions on Software Engineering* 36, 5 (2010), 618–643.