CUPID: Leveraging ChatGPT for More Accurate Duplicate Bug Report Detection

TING ZHANG, Singapore Management University, Singapore
IVANA CLAIRINE IRSAN, Singapore Management University, Singapore
FERDIAN THUNG, Singapore Management University, Singapore
DAVID LO, Singapore Management University, Singapore

Duplicate bug report detection (DBRD) is a long-standing challenge in both academia and industry. Over the past decades, researchers have proposed various approaches to detect duplicate bug reports more accurately. With the recent advancement of deep learning, researchers have also proposed several approaches that leverage deep learning models to detect duplicate bug reports. It is well acknowledged that the performance of deep learning-based approaches is highly dependent on the size of the training dataset. In the bug repositories with a large number of bug reports, deep learning-based approaches have shown promising performance. However, in the bug repositories with a typical number of issues, i.e., around 10k, the existing deep learning approaches show worse performance than the traditional approaches. A recent benchmarking study on DBRD also reveals that the performance of deep learning-based approaches is not always better than the traditional approaches. However, traditional approaches have limitations, e.g., they are usually based on the bag-of-words model, which cannot capture the semantics of bug reports. To address these aforementioned challenges, we seek to leverage a state-of-the-art large language model to improve the performance of the traditional DBRD approach.

In this paper, we propose an approach called CUPID, which combines the best-performing traditional DBRD approach REP with the state-of-the-art large language model ChatGPT. Specifically, we first leverage ChatGPT under the zero-shot setting to get essential information on bug reports. We then use the essential information as the input of REP to detect duplicate bug reports. We conducted an evaluation by comparing CUPID with three existing approaches on three datasets. The experimental results show that CUPID achieves new state-of-the-art results, reaching Recall Rate@10 scores ranging from 0.59 to 0.67 across all the datasets analyzed. In particular, CUPID improves over the prior state-of-the-art approach by 6.7% - 8.7% in terms of Recall Rate@10 in the datasets. CUPID also surpassed the state-of-the-art deep learning-based DBRD approach by up to 79.2%. Additionally, our study demonstrates the significant impact of prompt engineering on the performance of CUPID. Our work highlights the potential of combining large language models to improve the performance of software engineering tasks.

CCS Concepts: • **Software and its engineering → Maintaining software**.

Additional Key Words and Phrases: ChatGPT, Duplicate Bug Reports, Information Retrieval, Large Language Models

Authors' addresses: Ting Zhang, Singapore Management University, Singapore, tingzhang.2019@phdcs.smu.edu.sg; Ivana Clairine Irsan, Singapore Management University, Singapore, ivanairsan@smu.edu.sg; Ferdian Thung, Singapore Management University, Singapore, ferdianthung@smu.edu.sg; David Lo, Singapore Management University, Singapore, davidlo@smu.edu.sg.

0004-5411/2023/8-ART1 $15.00
https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

As software systems become larger and more complex, it is inevitable that they contain bugs. *Bug reports* are the main channel for users to report bugs to developers. Most software projects use issue tracking systems, such as Bugzilla [5], Jira [10] and GitHub [6], to manage bug reports and track the progress of bug fixing. When a user finds a bug, they can submit a bug report to the issue tracking system. Then, the developers will fix the bug according to the description in the bug report. However, many bug reports are duplicates of the existing bug reports. For example, in the dataset constructed by Lazar et al. [37], duplicate bug reports represent 12.67% - 23% out of the total bug reports in a system. It is crucial to identify duplicate bug reports as soon as possible to avoid wasting developers' time and effort on fixing the same bug multiple times To improve the efficiency of bug report management, it is desirable to have an automatic approach to identify duplicate bug reports.

Over the past decades, various duplicate bug report detection (DBRD) approaches have been proposed [25, 30, 53, 59, 69]. With the rapid development of deep learning, many deep learning-based approaches have been proposed in recent years [30, 53, 69]. They have demonstrated superior performance when the bug repositories are large enough to train the deep learning models. For instance, SABD [53] achieved over 0.6 in terms of Recall Rate@20 in all the experimented datasets. One of the common characteristics of these datasets is that they all contain more than $80k$ bug reports and over $10k$ *duplicate* bug reports in the training data, which is large enough to train a deep learning model. It is well acknowledged that deep learning models require a large amount of data to achieve high precision [51]. However, bug repositories of many projects are not large enough to train a deep learning model.

Based on the dataset provided by Joshi et al. [34], it was discovered that out of the 994 studied GitHub projects that have more than 50 stars and forks, the average number of issues was 2,365. Additionally, it is interesting to note that many active projects, including those with more than $100k$ stars, have fewer than $10k$ issues. For example, till 5th May 2023, both `ohmyzsh/ohmyzsh` [11] and `axios/axios` [4] have around $4k$ issues each, while `vuejs/vue` [15] has around $10k$ issues. Therefore, we argue that most projects do not have tens of thousands of issues. The repositories with tens of thousands of issues are considered as *atypical*, while a *typical* repository contains less than or around $10k$ issues. It is essential to highlight that young and fast-growing projects, although currently having a small number of issues, require more attention in handling the DBRD challenge. For instance, `Significant-Gravitas/Auto-GPT` [12], which was initially released on March 30, 2023, now contains less than $2k$ issues while it gets $124k$ stars. A recent benchmarking study on DBRD by Zhang et al. [72] also confirms that the performance of deep learning-based approaches loses to information retrieval-based approaches when the bug repositories only contain less or around $10k$ duplicate bug reports. How to improve the performance of DBRD in the *typical* bug repositories remains an open problem.

Prior to the development of deep learning, many non-deep learning-based approaches have been proposed [32, 54, 59, 60] (we refer to them as "*traditional* approaches" in this paper). Compared to deep learning-based approaches, these approaches are more promising for detecting duplicate bug reports in typical bug repositories. However, traditional approaches rely on either the vector space model [54] or the bag-of-words model [32]. These models cannot capture the semantics of bug reports. We seek to improve the performance of non-deep learning based approaches by considering the semantics of bug reports.

Recently, large language models (LLMs), e.g., Vicuna [21], LLama 2 [65], and ChatGPT [8], have achieved outstanding performance in a multitude of natural language processing (NLP) tasks [19, 26, 50]. However, leveraging the potential of LLMs to improve DBRD's performance is

not trivial. The most straightforward way is to directly query LLMs on whether two bug reports are duplicates. However, this is impractical due to the following reasons.

(1) Time-consuming and costly. To obtain the potential master bug reports to which a given bug report may be duplicated, we must pair it with all the bug reports available in the repository. When a new bug report is submitted, all previously submitted bug reports are considered duplicate candidates. It is infeasible to query LLMs to compare the given bug report with all the bug reports in the repository, as the LLMs' response is not instantaneous. While speeding it up is possible (e.g., by running many queries at once), it quickly gets very costly for LLMs such as ChatGPT, which operates on a pay-per-use basis for their API usage.

(2) Ignorance of other bug reports in the repository. If a method only compares two bug reports at a time, it will not take into account the information present in the other bug reports stored in the repository. Therefore, it would be hard to decide the relative order of all the duplicate candidates in order to recommend the top-k duplicate candidates. Although one possibility is in addition to querying ChatGPT on whether the bug report pair is duplicated or not, we ask ChatGPT to provide a measure of how confident it is in its answer, expressed as a similarity score or confidence score. However, without considering the information from other bug reports, the similarity score will be less reliable.

(3) LLMs are generative AI techniques which are designed to generate contents. Although LLMs have achieved impressive performance in a multitude of NLP tasks, many researchers argue that LLMs are only good at language abilities but not at actual reasoning [17, 42]. Thus, to take full advantage of LLMs, we carefully design the task to ensure its suitability for LLMs. As DBRD requires some reasoning on how two bug reports are duplicated to each other, it is not suitable to query LLMs directly.

We present CUPID, which stands for leveraging ChatGPT for more accurate duplicate bug report detection. CUPID aims to tackle the challenges mentioned above when directly querying LLMs for DBRD. We propose to leverage LLMs as an intermediate step to improve the performance of the traditional DBRD approach. Based on the recent benchmarking study by Zhang et al. [72], REP [59] demonstrates the best performance in the datasets with a typical number of issues, which is also the focus of this work. Thus, we select REP as the backbone duplicate retrieval method. Specifically, CUPID leverages state-of-the-art ChatGPT to identify keywords from bug reports and then incorporate them with REP to achieve better performance. By doing so, CUPID avoids using ChatGPT to compare the given bug report with all the bug reports in the repository. Furthermore, by standing on the shoulder of the traditional DBRD approach, CUPID also takes the information of the other bug reports in the repository into consideration. In particular, $BM25F_{ext}$ used by REP calculates inverse document frequency (IDF), which is a global term-weighting scheme across all the bug reports. In addition, CUPID prompts ChatGPT to identify keywords from bug reports, which requests ChatGPT to generate a list of relevant keywords based on the content of a bug report. Compared to a decision-making task, keyword identification is closer to a generative task. Our contribution can be summarized as follows:

- **Approach:** We propose CUPID, which combines modern LLMs with the traditional DBRD technique to enhance the accuracy of DBRD in software systems with the typical number of bug reports.
- **Evaluation:** We evaluate CUPID on three datasets from open-source projects and compare CUPID with three prior state-of-the-art approaches. The experimental results indicate that CUPID surpasses the performance of these existing DBRD approaches. Notably, CUPID achieves RR@10 scores ranging from 0.59 to 0.67 across all the datasets analyzed.

- **Direction:** We show that leveraging ChatGPT indirectly in conjunction with existing approaches can be beneficial. We anticipate that this will pave the way for future research to explore innovative ways to utilize state-of-the-art techniques with traditional ones.

The structure of this paper is as follows. Section 2 introduces the background of LLMs and DBRD. Section 3 presents the details of CUPID. We describe the experimental design in Section 4. Section 5 presents the experimental results. We discuss the threats to validity in Section 6. Section 7 discusses the related work. Finally, Section 8 concludes this paper and discusses future work.

2 BACKGROUND

2.1 Large Language Models

With easier access to large-scale datasets and the rapid development of hardware, recent years have witnessed the rapid development of various large language models (LLMs) [19, 22, 26, 41, 46, 64, 65, 70]. LLMs are pre-trained on massive amounts of texts and are capable of capturing the semantics of the texts. Most of them are based on the Transformer architecture [66] and are trained with the self-supervised learning paradigm. These models have achieved great success initially in the natural language processing (NLP) field and then have been widely used in solving software engineering tasks, such as API recommendation [31], code search [28], and pull request title generation [73]. For the LLMs with less than 1 billion parameters, such as CodeBERT [28], they can be directly fine-tuned on the downstream tasks to achieve better performance.

Most recently, LLMs with billions of parameters have been proposed, such as GPT-3 [19], PaLM [22], and LLaMa [64]. These models have demonstrated exceptional performance in various NLP tasks, ranging from translation [33] to grammatical error correction [27] and even fixing program bugs [56]. For such LLMs with billions of parameters, it is infeasible to fine-tune all the parameters using common hardware. Instead, they are usually used under the few-shot or zero-shot learning paradigm. These LLMs demonstrated great potential when only a few or no examples are available for the downstream tasks [27, 49].

In this study, to solve the DBRD task, we aim to leverage the power of LLMs. Specifically, we experiment with ChatGPT [8]. Launched by OpenAI in November 2022, ChatGPT has gained extensive attention from both academia and industry [17, 36]. ChatGPT has shown to be capable of responding effectively to a wide range of tasks. In a recent study by Bang et al. [17], ChatGPT is shown to achieve remarkable zero-shot performance on multiple tasks. In particular, ChatGPT outperforms the previous state-of-the-art zero-shot models in 9 out of 13 evaluation datasets, with the studied tasks ranging from sentiment analysis to question answering. As a successor of InstructGPT [46], ChatGPT employs reinforcement learning from human feedback [23, 46, 57] to align the model's output with human instructions. Thus, the reliability and accuracy of the model can be improved over time.

2.2 Duplicate Bug Report and its Detection

In this section, we first introduce the essential concepts, including bug reports and duplicate bug reports, and finally, we discuss the task of DBRD.

Bug reports are the primary means for users to communicate a problem or request features to developers [18]. Software projects usually rely on *issue tracking systems* to collect these bug reports. While the supported fields can vary from system to system, the *textual* information is included in all the issue tracking systems. The *textual* fields in a bug report usually consist of a `summary` (`title`) and a `description`. In Bugzilla or Jira, there are also several *categorical* fields, such as `priority` (bug assignees use this to prioritize their bug), `product`, `component`, etc.

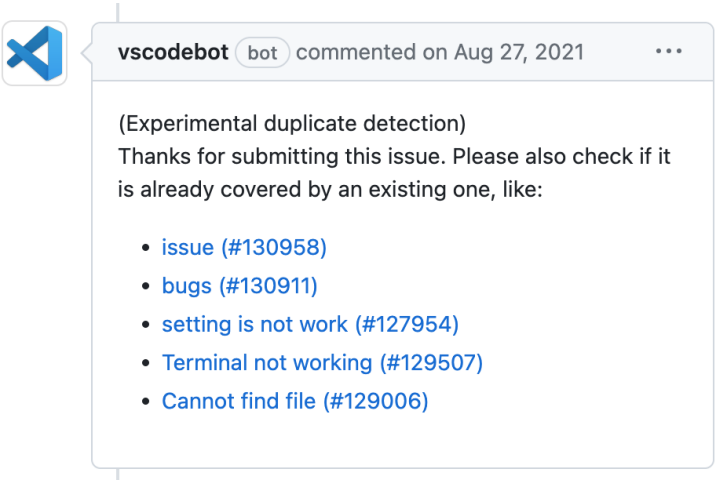


Fig. 1. An example of duplicate bug report detection in Microsoft VSCode GitHub repository: issue #131770.

DBRD aims to correctly link a duplicate bug report towards its *master* bug report. Following prior works [53, 54, 59], we denote the first submitted bug report on a specific fault as the *master* bug report and the subsequent bug reports on the same fault as *duplicates*. All the bug reports which are duplicates of each other, including *master* and *duplicates*, are in the same *bucket*. To help understand these concepts easier, we can imagine a *bucket* as a Hash Table, where the key is the master bug report, while the values are duplicate bug reports and themselves. Thus, for a unique bug, both the key and value would be just itself.

In the literature, depending on the problem setting, DBRD has been evaluated in two manners, i.e., (1) *classification* and (2) *ranking*. In the *classification* manner, the task is to classify whether two bug reports are duplicates or not. DC-CNN [30] and HINDBR [69] are two recent endeavors in this manner. In the *ranking* manner, the task is to rank the candidate bug reports according to their similarity to the given bug report. Referring to the Hash Table metaphor earlier, given a newly submitted bug report, DBRD technique finds the bucket to which it belongs (also equivalent to linking duplicate bug reports to its master). If it does not belong to any existing bucket, a new bucket in which the key and value are itself should be created. In this work, we focus on the *ranking* manner, which is more practical in real-world applications. One example of practical use is VSCODEBOT, a bot applied in the Microsoft VSCode GitHub repository. Its feature includes looking for potential duplicate issues. Figure 1 shows the duplicate issue suggestions made by VSCODEBOT on issue #131770 [9].

In the past decades, researchers have proposed various approaches to address the DBRD task in the ranking manner [45, 67]. Different DBRD approaches mainly differ in (1) feature engineering: which features in bug reports are selected and how these features are represented, and (2) similarity measurement: how to measure the similarity between two bug reports [61]. In terms of *feature engineering*, we further break down into two parts: (1) what features are selected (2) how to represent the features. All existing methods use textual information, and most of them use categorical information. Textual features, i.e., `summary` and `description`, include the most useful information about a bug. Different methods differ in which categorical features to use. To model these features, traditional methods utilize bag-of-words, character-level N-gram, or $BM25_{ext}$ to model textual features [32, 59, 61], while bag-of-words or hand-crafted methods are usually used to model

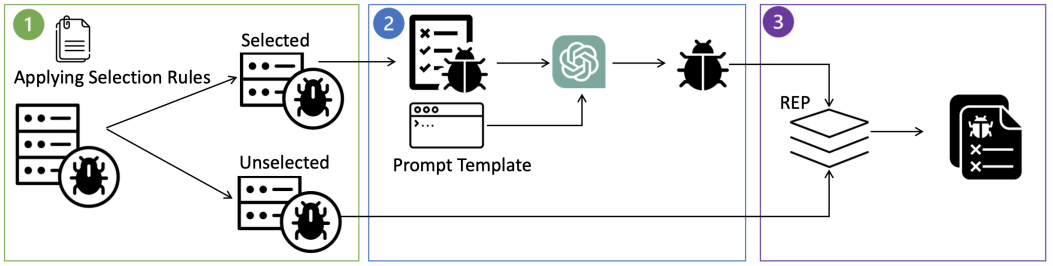


Fig. 2. CUPID contains three stages: In Stage 1, it applies selection rules to select the test bug reports that need to be processed; In Stage 2, it utilizes ChatGPT to process the selected bug reports; In Stage 3, it leverages REP to retrieve potential master bug report for each test bug report.

categorical features. Deep learning-based methods utilize word embeddings, such as GloVe [48] and word2vec [44], to represent the textual information. Other types of neural networks, such as HIN2vec [29], are used to represent categorical information. For *similarity measurement*, traditional methods usually use Cosine, Dice, and Jaccard similarity [32, 54]. While some deep learning-based models also adopt this similarity measure [25], some of them leverage neural networks to learn the similarity [53].

3 APPROACH

We propose CUPID to combine the advantages of both the traditional DBRD approach and LLM. As mentioned earlier, our work focuses on solving the DBRD challenge in the repositories with a typical number of issues; evidence shows that traditional DBRD approaches would fit more in this condition than deep learning-based approaches [72]. Figure 2 shows an overview of the proposed method. The overall process consists of three main stages: (1) Applying *selection rules* to select the bug reports that need to be processed by ChatGPT, (2) Running ChatGPT with *prompt template* to get the essential keywords of the selected bug reports, and (3) Applying REP to retrieve *potential master bug reports*.

In the sections that follow, we first introduce the datasets used in this work. Then, we describe the selection rules and prompt template used by CUPID. Finally, we introduce the REP approach.

3.1 Applying Selection Rules

Considering the computational cost of ChatGPT, we did not run ChatGPT on all the bug reports in the test dataset. Similarly, in practice, we do not need to run ChatGPT on each newly submitted bug report. To further improve efficiency while keeping accuracy, we explore and propose selection rules. These rules are based on the length and content of the bug reports, with a goal to prioritize bug reports that are harder to process by REP while reducing the number of bug reports that are fed into ChatGPT. The selection criteria are as follows:

Length: We select bug reports whose description is considered to be *long*. We consider bug reports whose description is longer than n words as *long* bug reports. We get n by calculating the 75th percentile of the length of description in the training set. The reason why we select long bug reports is that long bug reports are usually not concise and contain long stack traces and code snippets. These long bug reports would make it challenging for REP to retrieve the potential master bug reports.

Content: We select the bug reports whose description contains code snippets or URLs. We use regular expressions to match and select these bug reports. Note that after keeping long bug reports, we still have bug reports that contain code snippets or URLs. Some bug reports are very short and with the majority of the content being code snippets or URLs. For developers, this information is useful. However, for a DBRD method, this information can be hard to process. We select these bug reports because not all the code snippets and URLs are useful for REP to retrieve the potential master bug reports. We also do not directly remove code snippets or URLs. The reason is that we want to keep the original structure of the bug reports for ChatGPT to understand the language better. We then utilize ChatGPT to identify keywords from these bug reports.

3.2 Running ChatGPT with Prompt Template

After Stage 1, we run ChatGPT on the selected bug reports, i.e., either (1) the `description` is long or (2) the `description` contains code snippets or URLs.

Prompt [40] is a set of instructions that can be used to probe LLMs to generate the target outcome [17]. Prior studies have empirically shown that ChatGPT is sensitive to prompts. Thus, for different tasks, the prompts should be carefully designed to enable LLMs to demonstrate their abilities.

We craft the prompt template used by CUPID as shown below.

Prompt Template:

```
I have a bug report which contains summary and description. I want you to select
    keywords from both parts which keep the main meaning of the bug report. These
     keywords would be used for duplicate bug report detection. Output format: `
    Summary: Selected Keywords \n Description: Selected Keywords` \n\n >>>
    Summary: [Summary] \n\n >>> Description:
[Description]
```

This template is designed for a single-turn dialogue. For each bug report, we open a new dialogue with ChatGPT. After getting the response from ChatGPT, we replace the original `Summary` and `Description` in the bug report with the returned identified keywords of `Summary` and `Description`. We keep the remaining part of the bug report unchanged.

Regarding the design of the prompt template, our intuition is that we consider bug reporters are likely to have more expertise and domain knowledge than ChatGPT. Therefore, the language and terms they use when reporting bugs may be similar to each other, and this similarity can be leveraged by DBRD methods. It would benefit more not to replace the whole expression but rather select and keep the essential information for DBRD methods to process. To support our intuition, we also conduct experiments with other prompt templates and report the results in Section 5.

3.3 Retrieving Potential Master Bug Reports

Considering the superiority of REP in the task of DBRD shown in a recent study [72], especially on projects with a typical number of issues, we use REP as the DBRD approach in CUPID. Here, we briefly introduce the REP approach to make the paper self-contained. We refer the readers to the original paper [59] for more details.

As shown in Formula 1, REP is a linear combination of seven features, including **textual** features and **categorical** features.

$$REP(d, q) = \sum_{i=1}^{7} w_i \cdot feature_i \tag{1}$$

, where d is the bug report in the repository R, q, is the query (i.e., new bug report), w_i is the weight of the i-th feature, and $feature_i$ is the i-th feature. The first two features are both **textual** features, and the rest five features are **categorical** features. Figure 2 shows how to get each feature.

$$
\begin{aligned}
feature_1(d,q) &= BM25F_{ext}(d,q) \text{ //of unigrams} \\
feature_2(d,q) &= BM25F_{ext}(d,q) \text{ //of bigrams} \\
feature_3(d,q) &= \begin{cases} 1, & \text{if } d \cdot prod = q \cdot prod \\ 0, & \text{otherwise} \end{cases} \\
feature_4(d,q) &= \begin{cases} 1, & \text{if } d.comp = q.comp \\ 0, & \text{otherwise} \end{cases} \\
feature_5(d,q) &= \begin{cases} 1, & \text{if } d.type = q.type \\ 0, & \text{otherwise} \end{cases} \\
feature_6(d,q) &= \frac{1}{1+\mid d.prio - q.prio \mid} \\
feature_7(d,q) &= \frac{1}{1+\mid d.vers - q.vers \mid}
\end{aligned}
\tag{2}
$$

The first two features regard the textual similarity between two bug reports over the fields `summary` and `description`. These two textual features are calculated by $BM25F_{ext}$ between bug report d and query bug report q. $BM25F$ [52, 71] is an effective textual similarity function for retrieving documents that have structures. The authors of REP extend $BM25F$ by considering term frequencies in queries and proposed $BM25F_{ext}$.

In $feature_1$, `summary` and `description` are represented in uni-gram, while in $feature_2$, `summary` and `description` are represented in bi-gram. Thus, the input of $BM25F_{ext}$ consists of a bag of uni-grams and bi-grams in both features. For $feature_{3-5}$, they are the categorical features of `product`, `component`, and `type`, respectively. If the corresponding field value from d and q is the same, the value of the feature is 1, otherwise, it is 0. For $feature_{6-7}$, they are the categorical features of `priority` and `version`, respectively. They are calculated by the reciprocal of the distance between the corresponding field value from d and q. Overall, the REP approach contains 19 free parameters with different initial values. These parameters are tuned by gradient descent.

4 EXPERIMENTAL DESIGN

4.1 Research Questions

To understand whether CUPID performs better compared to existing state-of-the-art approaches and whether each component of CUPID is useful, we answer the following two research questions (RQs):

- **RQ1:** *How effective is CUPID compared to the state-of-the-art approaches?*
- **RQ2:** *How effective are the components of CUPID?* To answer this RQ, we conduct an ablation study on the components of CUPID. This RQ is further divided into the following sub-RQs:
  - **RQ2.1:** *How effective is the prompt template?*
  - **RQ2.2:** *How effective are the selection rules?*
  - **RQ2.3:** *How effective is ChatGPT compared to other LLMs?*

Table 1. Dataset statistics. Cupid here refers to the selected bug reports run by ChatGPT.

Dataset	Total Bugs	Train. Pairs	Valid. Pairs	Test	
				Dup. Bugs	Cupid
Spark	9,579	626	26	81	59
Hadoop	14,016	626	27	92	57
Kibana	17,016	724	28	184	114

4.2 Dataset

As mentioned in Section 1, we are concerned about boosting the performance of DBRD, especially in the bug repositories with the typical number of issues. Therefore, the target datasets are those that contain a typical number of issues. We employ three datasets, i.e., Spark, Hadoop, and Kibana datasets, which are provided by a recent benchmarking study by Zhang et al. [72]. These datasets contain around 10k issues each, which is considered a typical number of issues. These datasets are recent issues, ranging from 2018 to 2022, which addressed the age bias, i.e., the model performs differently on the recent data and old data. Spark and Hadoop are two popular open-source distributed computing frameworks. They both use Jira as their issue tracking system. Kibana is a visualization tool for Elasticsearch, and it uses GitHub as its issue tracking system. The statistics of the datasets are shown in Table 1. The duplicate and non-duplicate pairs were sampled by Zhang et al. [72]. Their ratio is 1:1. We obtained the data in the dataset provided by Zhang et al. In our experiment, we fixed the number of training and validation pairs. The number of duplicate bug reports in the test set is the bug reports we investigate. We report the performance of each approach in terms of how they perform in retrieving the master bug reports.

4.3 Evaluation Metrics

Following prior works on DBRD [16, 25, 54, 72], we only use Recall Rate@k (RR@k) as the evaluation metric, where k represents the number of bug reports to be considered. Note that a few other works have also adopted Mean Average Precision (MAP) in the DBRD literature. However, since MAP considers all of the predicted positions, it is not suitable for our case, where only the top k predictions matter. This is based on real-world practice, where developers are more likely to check the top k predictions rather than all of the predictions. A survey on practitioners' expectations towards fault localization also shows that around 98% of respondents are not willing to check the predictions beyond the top-10 to find the faulty element [35].

Furthermore, as already discussed in early work [54], the two widely used metrics in information retrieval, i.e., Precision@k and Recall@k, do not fit into how DBRD works. For each query bug report, we only have a master bug report to look for (i.e., the relevant item is only 1). Consider $k = 10$, a successful prediction would lead to Precision@10=1/10(10%), and Recall@10=1/1(100%); otherwise, we will get Precision@10=0 and Recall@10=0. Therefore, we adopt RR@k as the evaluation metric.

Following the definition in prior works [53, 55, 59, 67], RR@k is defined as the percentage of duplicate bug reports that are correctly assigned to the bucket they belong to when a model makes a top-k prediction for each test bug report. In our experiment, RR@k will measure how well DBRD techniques correctly link the duplicate bug reports to their master bug report. A higher RR@k indicates that more bug reports in the test set are correctly linked to the bucket they belong to when a model retrieves top-k prediction.

$$RecallRate = \frac{N_{recalled}}{N_{total}} \tag{3}$$

Formula 3 shows how to calculate the Recall Rate. $N_{recalled}$ refers to the number of duplicate bug reports whose bucket (master bug report) are in the suggested list (with a size of [1,2,...,k]). N_{total} refers to the number of duplicate bug reports investigated. Considering different sizes of the suggested list, i.e., k, we can get RR@k. Following the benchmarking work by Zhang et al. [72], in our case, we consider at most 10 predictions, i.e., $k = [1, 2..., 10]$.

To facilitate understanding, we show an example to help explain how RR@k is calculated. Assume that we have a test set with three duplicate bug reports, which we call B_1, B_2, and B_3, that we need to match with their master bug reports. We adopt a DBRD technique (e.g., REP) to suggest the top 10 potential master bug reports for each test bug report. If REP manages to identify B_1's master bug report in the 3rd position of the suggested list, we'll consider it a hit at the 3rd prediction. Next, B_2 is also correctly matched with its master bug report, which appears in the 5th position in the suggested list. However, REP fails to retrieve B_3's master bug report in the top-10 predictions. This means we have two successful and one failed detection. If we set $k = 1$ or $k = 2$, then the RR@k is 0, as REP does not rank the correct master bug report in the top two positions for any of the three duplicate bug reports in the test set. On the other hand, if we set $k = 3$ or $k = 4$, then the RR@k is 1/3, as REP successfully matches B_1's master bug report in the third position. Similarly, if we set $k = 5, 6, ..., 10$, then the RR@k is 2/3, as REP successfully matches the master bug reports for B_1 and B_2.

4.4 Compared Techniques

In this work, we compare Cupid with state-of-the-art DBRD techniques, which consider DBRD as a ranking problem, i.e., REP [59], Siamese Pair [25], and SABD [53].

REP [59] The details of REP can referred in Section 3.3.

Siamese Pair [25] is the first approach that leverages deep learning for DBRD. As its name suggests, Siamese Pair utilizes Siamese variants of Convolutional Neural Networks (CNN) and Recurrent Neural Networks (RNN) trained on max-margin objective to distinguish similar bugs from non-similar bugs. Siamese Pair adopts word embedding to represent the textual data as numerical vectors. Then, it employs three different types of neural networks to encode `summary`, `description`, and categorical features according to their properties. Specifically, `summary` is encoded by bi-LSTM, `description` is encoded by a CNN, while the categorical information is encoded with a single-layer neural network. When a new bug report comes, Siamese Pair encodes the bug report with the trained model. It then calculates the cosine similarity between the new bug report and each bug report in the master set and gives top k predictions.

SABD [53] is the latest deep learning-based DBRD approach. It consists of two sub-network modules, where each module compares textual and categorical data from two bug reports. In the textual sub-network, the soft-attention alignment mechanism [47] compares each word in a bug report with a fixed-length representation of all words in the other bug report. By doing so, SABD learns the joint representation of bug reports. In the categorical sub-network, each categorical field relates to a lookup table that links the field value to a real-valued vector. The output vector from each sub-module is concatenated and fed to a fully connected layer. Finally, a classifier layer, which is a logistic regression, produces the final prediction, i.e., whether the two bug reports are duplicates.

In RQ2.3, we also compare ChatGPT with other open-source LLMs. We select three LLMs, i.e., VICUNA-13B (i.e., `lmsys/vicuna-13b-v1.5` in Hugging Face library [68]), WIZARDLM-13B (i.e., `WizardLM/WizardLM-13B-V1.2`), and LLAMA 2-13B-CHAT (i.e., `meta-llama/Llama-2-13b-chat-hf`) based on their performance in MMLU benchmark on the chatbot leaderboard [1] in August 2023. Due to the limit of the computing resources, we were only able to run the LLMs containing less or equal to 13B parameters.

VICUNA-13B [21] is an open-source chatbot trained by fine-tuning LLAMA on 70k user-shared conversations collected from ShareGPT.com. VICUNA-13B was trained on top of Stanford's Alpaca [63] with three main improvements: (1) multi-turn conversations; (2) memory optimizations: it is worth noting that the max context length of VICUNA-13B was expanded to 2,048; and (3) cost reduction. We utilized the variant which fine-tuned LLAMA 2.

WIZARDLM-13B [70] propose an automatic method named *Evol-Instruct* to mass-produce open-domain instructions. Evol-Instruct starts with simple initial instructions and then re-writes them step-by-step into more complex instructions. WIZARDLM fine-tuned LLAMA with mixed generated instruction data. The experimental results show that WizardLM achieves more than 90% capacity of ChatGPT on 17 out of 29 skills. Similar to VICUNA-13B, we utilized the variant of WIZARDLM which fine-tuned LLAMA 2.

LLAMA 2-13B-CHAT [65] is a fined-tunes version of LLAMA 2 optimized for dialogue use cases. It contains three variants, i.e., 7B, 13B, and 70B parameters. LLAMA 2 follows most of the pertaining setting and model architecture from LLAMA 1 [64]. The primary architectural differences contain increased context length and grouped-query attention. Furthermore, LLAMA 2-13B-CHAT undergoes instruction tuning and RLHF. Note that although LLAMA 2-13B-CHAT can take up to 4,096 tokens, we set the max token length to 2,048 as VICUNA-13B and WIZARDLM-13B.

4.5 ChatGPT Setup

Given that ChatGPT is still fast evolving, it has undergone several iterations [**?**]. In this study, we worked on the GPT-3.5 version. To interact with ChatGPT, we used an open-sourced API [2] that creates a chat window on the ChatGPT website. It saved us from the manual labor of opening a chat window and copying the response back. Although there is an official ChatGPT API available, we were not able to use it without paying for it. Therefore, we chose to use the free version of ChatGPT, which we believe to have a wider range of users compared to the paid one. As such, our results would be more valuable as they are applicable to a wider range of users.

During the experiments, for each query bug report, we initialize a new conversation to avoid the influence of the previous conversation on other bug reports. Since ChatGPT may generate different answers for the same query, we ran ChatGPT five times for each query and aggregated the results (i.e., summing up the 5-round results) to obtain the final answer.

4.6 Implementation

To fairly compare CUPID with the baselines, we fix the training pairs for all techniques. Since there is randomness in the deep learning-based models, i.e., SIAMESE PAIR and SABD, the reported results were the average results after running them five times. The implementation details can be found in our replication package [3].

[1] https://huggingface.co/spaces/lmsys/chatbot-arena-leaderboard

Table 2. Recall Rate@k obtained on the Spark dataset. The best performance in terms of RR@10 is highlighted accordingly.

Method	RR@1	RR@2	RR@3	RR@4	RR@5	RR@10
REP	0.346	0.383	0.457	0.481	0.481	0.556
Siamese Pair	0.037	0.049	0.059	0.064	0.074	0.121
SABD	0.202	0.247	0.281	0.294	0.304	0.331
Cupid	0.346	0.395	0.432	0.469	0.481	0.593

Table 3. Recall Rate@k obtained on the Hadoop dataset. The best performance in terms of RR@10 is highlighted accordingly.

Method	RR@1	RR@2	RR@3	RR@4	RR@5	RR@10
REP	0.402	0.489	0.522	0.554	0.576	0.609
Siamese Pair	0.033	0.046	0.057	0.063	0.076	0.093
SABD	0.215	0.267	0.293	0.304	0.324	0.411
Cupid	0.391	0.511	0.565	0.576	0.609	0.652

Table 4. Recall Rate@k obtained on the Kibana dataset. The best performance in terms of RR@10 is highlighted accordingly.

Method	RR@1	RR@2	RR@3	RR@4	RR@5	RR@10
REP	0.364	0.440	0.527	0.560	0.587	0.620
Siamese Pair	0.020	0.036	0.050	0.063	0.076	0.092
SABD	0.293	0.382	0.428	0.467	0.489	0.555
Cupid	0.408	0.522	0.571	0.603	0.62	0.674

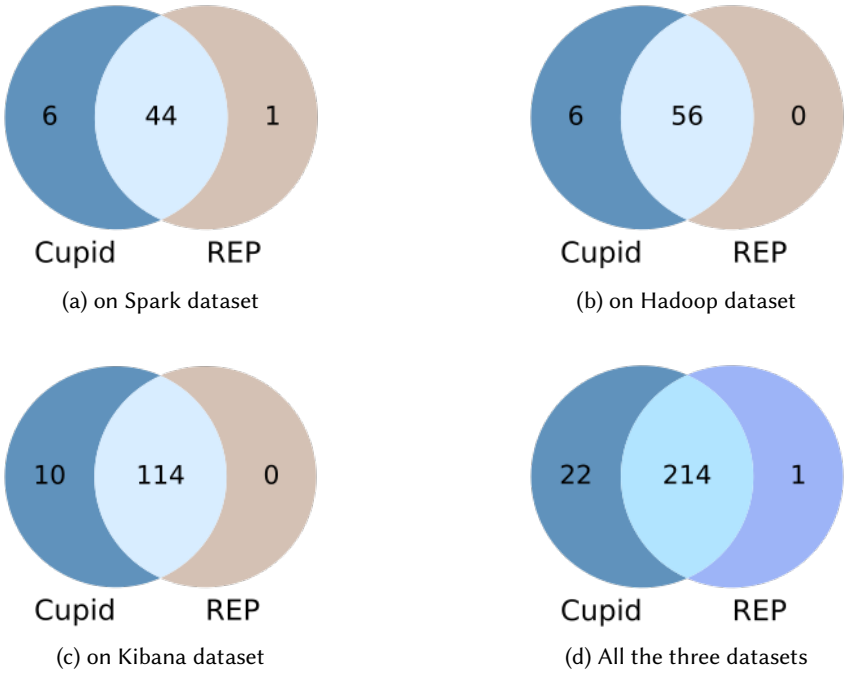
5 RESULTS

5.1 RQ1: Comparing with baselines

Table 2, 3, and 4 show the results of Cupid and the baselines on the Spark, Hadoop, and Kibana datasets. Overall, Cupid consistently improves the DBRD performance in terms of RR@10 on all three datasets, yielding an improvement of 6.7% (Spark) to 8.7% (Kibana) over the prior state-of-the-art approach REP. This improvement is obtained by successfully utilizing the language generation ability of ChatGPT to transform the bug reports into a format where only essential information is kept. In comparison with the best-performing deep learning-based approach, i.e., SABD, we observe an improvement of up to 79.2% on the Spark dataset. In the low-volume datasets, SABD and Siamese Pair lose to non-deep learning approaches, i.e., REP and Cupid.

Comparing the performance of Siamese Pair and SABD in all three datasets, we can find that Siamese Pair suffers more from the challenge of limited training data. Siamese Pair performs less than 50% of SABD in all the three datasets in terms of RR@10. We argue that when there is a lack of adequate training data, it is less meaningful to compare different deep learning-based models.

Dataset-wise, all approaches perform relatively worse on the Spark dataset and relatively better on the Kibana dataset. The observation aligns with the findings from prior studies [53]: the same



(a) on Spark dataset

(b) on Hadoop dataset

(c) on Kibana dataset

(d) All the three datasets

Fig. 3. Successful prediction Venn diagram

DBRD approach, i.e., SABD, achieves a variety of RR@10 on different datasets examined, ranging from 0.55 (on OpenOffice dataset) to 0.7 (on Netbeans dataset). It shows that the performance of a DBRD technique also depends on the dataset characteristics. This observation inspires us that it would be beneficial for each dataset if we tune the prompt template based on the characteristics of each dataset. We leave this for future work to boost the performance further.

Figure 3 shows the Venn diagrams for successful predictions made by the prior state-of-the-art method, i.e., REP, and CUPID on each dataset and all datasets combined. We see that CUPID successfully retrieves more master bug reports compared to REP. On the Hadoop and Kibana datasets, only CUPID successfully retrieved more master bug reports, while REP did not successfully retrieve more.

To demonstrate the ability of CUPID, we show an example, i.e., the query bug report is `HADOOP-17091`[7] where REP failed to predict the correct master bug report in the top-10 positions, while CUPID managed to. Figure 4 shows the `summary` and `description` of this issue. We can see that there is no natural language in the description, containing only error messages. Thus, REP considered the most possible master bug report to be `HADOOP-16648`, which also contains a large portion of the error messages. We checked the single-run result by ChatGPT. Thanks to the language understanding and generation ability of ChatGPT, CUPID identified the keywords: `Javadoc, HTML version, HTML4, HTML5, warning, comments, valid, GeneratedMessageV3, package, not found, error` from the `description` of `HADOOP-17091`. The generated shorter `description` on the query bug report has several words overlap with the `description` of the real master bug report (`HADOOP-16862`). It enables CUPID to successfully rank at the first position. Since the real master bug report has a long error message as the `description`, REP failed to retrieve it. This example

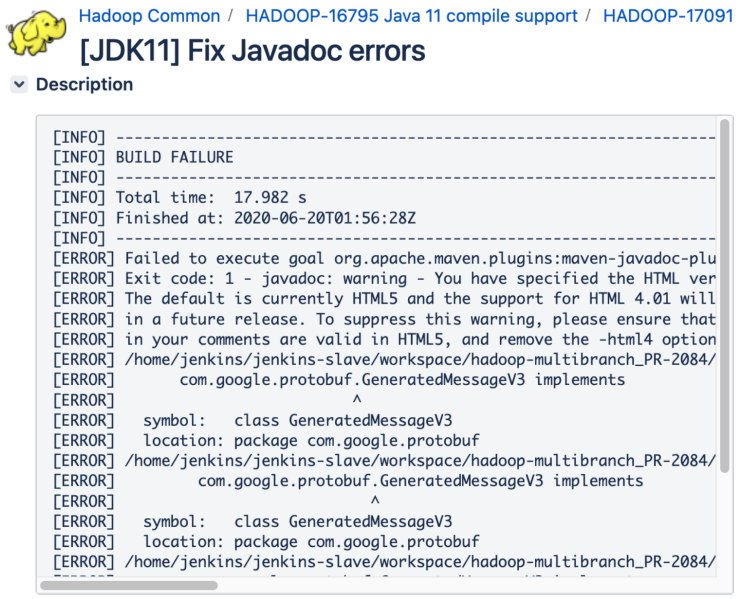


Fig. 4. The case where Cupid succeeded while REP failed: HADOOP-17091

Fig. 5. The case where Cupid failed while REP succeeded: SPARK-33661

shows that ChatGPT can be helpful when descriptions are long and contain non-natural language texts. It can generate the most important keywords, which are vital for duplicate detection.

Listing 1. Prompt Template 1

Prompt Template 1:

```
Rephrase the following bug report in five distinct styles, to avoid repetition,
    while keeping its meaning. Output format: `[1-5]. Summary: [Rephrased Summary
    ] \n Description: [Rephrased Description]` \n\n >>> Summary: [SUMMARY] \n\n
    >>> Description: [DESCRIPTION]
```

Based on Figure 3, we can also observe that on the Spark dataset, REP can actually predict a bug report, i.e., `SPARK-33661` [14] that CUPID fails to predict. Figure 5 shows the `summary` and `description` part of this issue. We checked the single-run result by ChatGPT. The identified keywords are: `Summary: Unable, load, RandomForestClassificationModel, trained, Spark 2.x`, `Description: load, RandomForestClassificationModel, trained, Spark 2.x, Spark 3.x, exception, raised, schema incompatibility, saved, data, expected, existing, random forest models, upgrade, retrain`. The selected keywords look reasonable since they keep the important identities that are related to this bug. The master bug report, which was listed at the top-1 position by CUPID is `SPARK-31169` [13]. This bug report is about different results that are obtained when building random forest models using different versions of Spark. It is clear that it is not a duplicate of `SPARK-33661`. However, it is not hard to find that in the `summary` of this issue, common words exist, such as `Random Forest`, `SparkML 2.3.3 vs 2.4.x`. In addition, in the `description` part, we can find similar words, e.g., `train,version,spark,model`, which are quite relevant to the identified keywords in `SPARK-33661`. This example shows that, in some cases, keywords are not sufficient to identify duplicate bug reports. The same set of words may lead to different errors.

Answer to RQ1: CUPID outperforms the best baseline by 6.7%, 7%, and 8.7% in terms of Recall Rate@10 on the Spark, Hadoop, and Kibana datasets, respectively.

5.2 RQ2: Ablation Study

RQ2.1: The effectiveness of Prompt Templates. We first investigate the effectiveness of different prompt templates on the Spark dataset due to the fact that it is the smallest dataset. We came up with a basic prompt template (i.e., Prompt Template 1), as shown in Listing 1.

In **Prompt Template 1**, we aim to describe the task and requirement in a simple way. We also specify the output format. The hypothesis is different stakeholders, e.g., users, developers, or testers, have different expertise and experience levels; thus, how they write bug reports would vary. Therefore, we prompt ChatGPT to get alternative bug reports. This procedure can be viewed as data augmentation [24], where the goal is to generate auxiliary samples that are semantically similar to the original sample. In the beginning, we believed prompting ChatGPT to `rephrase` the bug report should be one of the most direct ways to achieve this goal.

Thus, we further experimented with a more comprehensive **Prompt Template 2**, where we added a persona description and also included the aim of this step. This prompt template is an augmented version of **Prompt Template 1**. Listing 2 shows the template.

Table 5 shows the results of querying ChatGPT with the two templates above and with the template employed in CUPID. We observe that **Prompt Template 2**, which is more comprehensive than **Prompt Template 1**, indeed leads to a slightly better performance: it surpassed the method with Prompt Template 1 by 2.3% in terms of RR@10. Although these two templates convey very similar meanings, with one being more succinct and the other being more verbose, they did make

Listing 2. Prompt Template 2

Prompt Template 2:

```
I want you to act as a collaborator for maintaining bug reports from a software
    project. Your job is to rephrase the bug report, to avoid repetition, while
    keeping its meaning. The aim of this step is to help filter duplicate bug
    reports. You will need to write five different versions of the bug report you
     encounter. You can delete the contents you perceive useless. Output format:
    `[1-5]. Summary: [Rephrased Summary] \n Description: [Rephrased Description
    ]`. Now, your need to rephrase the following bug report: \n\n >>> Summary:
    [SUMMARY] \n\n >>> Description: [DESCRIPTION]
```

Table 5. Ablation study on different prompt templates: Recall Rate@k obtained on the Spark dataset. The best performance in terms of RR@10 is highlighted accordingly. PT is short for "prompt template".

Method	RR@1	RR@2	RR@3	RR@4	RR@5	RR@10
w/ PT 1	0.309	0.37	0.432	0.457	0.469	0.519
w/ PT 2	0.333	0.37	0.432	0.457	0.469	0.531
Cupid	0.346	0.395	0.432	0.469	0.481	0.593

an impact on the performance of running rephrased bug reports in DBRD. Using the final template in Cupid can boost the performance of these two prompt templates by 11.7% in terms of RR@10. These results indicate the significance of prompts.

At first glance, both Prompt Templates 1 and 2 seem more intuitive than the final prompt template we use (the one shown in Section 3). Prompt Template 1 and 2 queried ChatGPT to *rephrase*, while the final prompt in Cupid queried ChatGPT to *select keywords*. However, after checking the rephrased bug reports generated by ChatGPT with Prompt Template 1 and 2, we believe *rephrasing* the test bug reports is not the right direction to pursue. In retrospect, it would make more sense to rephrase all bug reports in the dataset, regardless of training or testing. However, as mentioned in Section 1, the drawback is the expenses of running ChatGPT. There is a widely-experienced error: `Too many requests in 1 hour, try again later` [1], which many users complain about. Despite the lack of an official document specifying the exact number of requests that can be made with ChatGPT per hour, this issue commonly occurs, hindering the whole DBRD process. Given the major difference between the query bug report and candidate bug reports, only rephrasing query bug reports would not make it easier to retrieve the master bug reports. In the context of traditional DBRD approaches, it could make the distance between the rephrased bug report and the master bug report further.

RQ2.2: The effectiveness of Selection Rules. Here, we also conducted experiments on Spark dataset to investigate the effectiveness of selection rules.

Table 6 shows how many bug reports need to query ChatGPT after adopting (1) no-selection rules, (2) selection by length, and (3) selection by length and content. If we only use *length* as the selection criteria, we will only need to run ChatGPT on 40.7%, 27.2%, and 41.8% of the original test bug reports in Spark, Hadoop, and Kibana dataset, respectively. While the computational cost would be reduced, it is essentially a trade-off: we want to achieve both efficiency and accuracy, which can be contradictory in some cases. We want to take full advantage of ChatGPT with minimal computational costs. Other than length, we also identify the *content* criteria. After adopting both length and content criteria, the bug reports needed to be processed by ChatGPT increased and accounted for 72.8%, 62%, 62%, which still saved more than 25% bug reports from processing.

Table 6. Number of test bugs and bugs that need to run after adopting selection rules.

Selection Rule	Dataset		
	Spark	Hadoop	Kibana
None	81	92	184
Length	33	25	77
Length+Content	59	57	114

Table 7. Ablation study on selection rules: Recall Rate@k obtained on the Spark dataset.

Selection	RR@1	RR@2	RR@3	RR@4	RR@5	RR@10
✗	0.333	0.395	0.432	0.469	0.481	0.58
✓	0.346	0.395	0.432	0.469	0.481	0.593

Table 8. Ablation study on ChatGPT: Recall Rate@k obtained on the Spark dataset.

Model	RR@1	RR@2	RR@3	RR@4	RR@5	RR@10
VICUNA-13B	0.358	0.395	0.432	0.432	0.457	0.506
WIZARDLM-13B	0.370	0.395	0.420	0.444	0.469	0.568
LLAMA 2-13B-CHAT	0.296	0.358	0.383	0.407	0.42	0.494
ChatGPT	0.333	0.395	0.432	0.469	0.481	0.58

Table 7 shows the corresponding results of applying selection rules. Comparing the performance of no selection rules, i.e., querying all the test bug reports with ChatGPT, and applying *both* selection rules, we can observe that after applying the rules, RR@10 improves by 2.2%. Despite only making a small improvement, it frees at least 25% of the bug reports in the test set from querying ChatGPT. Here, we do not only save the computational cost but also improve accuracy.

RQ2.3: The effectiveness of ChatGPT. Here, we have conducted experiments on the SPARK dataset to assess the efficacy of ChatGPT in comparison to other open-source LLMs. In a similar manner, we executed all three LLMs five times, while the generated responses across all five runs remained the same. We adapted the final version of the prompt template from CUPID, making slight modifications to ensure compatibility with the appropriate prompt format for each LLM. Listing 3 shows the prompt templates employed by each LLM. The main element, denoted as *select keywords*, remains consistent across all templates, while only the formats differ.

Table 8 shows the results with the comparison among ChatGPT and the other three LLMs. We can observe that WIZARDLM-13B can achieve a similar performance as ChatGPT with only 2% drop in terms of RR@10. VICUNA-13B and LLAMA 2-13B-CHAT perform worse compared to ChatGPT. The good performance of WIZARDLM-13B makes it promising to use an open-source LLM for the DBRD task. Further investigation on when and why open-source LLMs lose to ChatGPT can be put to take full advantage of the latest advancement of LLMs.

Answer to RQ2: Both prompt templates and selection rules are effective in improving the performance of CUPID. Furthermore, ChatGPT is better than the three selected open-source LLMs.

Listing 3. Prompt Template for Vicuna-13B, WizardLM-13B and Llama 2-13B-Chat

Prompt Template for Vicuna-13B and WizardLM-13B:

```
"A chat between a curious user and an artificial intelligence assistant. The
    assistant gives helpful, detailed, and polite answers to the user's questions
    .\n\nUSER: I will given you a bug report summary and a bug report description
    . You need extract the usefull keywords from summary and description,
    respectively. These keywords would be used for duplicate bug report detection
    . You need to reply like this:\nSummary Keywords:\nDescription Keywords:\
    nASSISTANT: Sure!</s>\nUSER:Bug report summary: {}\n\nBug report description:
     {}\n\nASSISTANT: Summary Keywords:",
```

Prompt Template for Llama 2-13B-Chat:

```
"<<SYS>>\nA chat between a curious user and an artificial intelligence assistant.
    The assistant gives helpful, detailed, and polite answers to the user's
    questions.\n<</SYS>>\n[INST]\nUser: I will given you a bug report summary and
     a bug report description. Note that the description can include log/error
    message or stack traces. You need extract the usefull keywords from summary
    and description, respectively. These keywords would be used for duplicate bug
     report detection. You need to reply like this:\nSummary Keywords:\
    nDescription Keywords:\nBug report summary: {}\n\nBug report description: {}\
    n\n[/INST]\n"
```

6 THREATS TO VALIDITY

Internal. The main internal threat is whether there is data leakage in ChatGPT. However, since we do not have access to the training data of ChatGPT, we cannot verify whether there is data leakage in ChatGPT. Even so, since we did not directly use ChatGPT to compare whether two bug reports are duplicates, instead, we utilize ChatGPT indirectly, which may not benefit much from memorizing the training data. Furthermore, we noticed that ChatGPT did not exhibit unrealistic perfect performance, e.g., reaching 0.9 at RR@10, across different prompt structures. It suggests that it is less likely for ChatGPT to rely solely on the memorization of its training data. Thus, we believe this threat is minimal.

External. The primary external threat is the generalizability of our findings. Our focus in this study is on datasets with a typical number of bug reports, roughly $10k$ issues. Therefore, our results may not extend to datasets with a significantly greater number of bug reports, such as those containing tens of thousands of issues. Nevertheless, we believe that our findings remain valuable for the majority of projects. This is supported by the fact that in a dataset of 994 high-quality projects from GitHub, each project contains an average of $2k$ issues [34].

7 RELATED WORK

Except for the works that studied the DBRD task, other automated bug report management tasks have also attracted much research interest [38, 58, 73, 75]. In this section, we briefly introduce several studies on automated bug report management tasks, including bug component assignment [58], developer assignment [38], and issue title generation [74]. Su et al. [58] propose a learning-to-rank framework that leverages the correct bug assignment history to assign the most appropriate component to a bug. Instead of only using the features from bug reports, their approach also derives rich features from this knowledge graph. Lee et al. [38] focus on assigning a bug report to the most appropriate developer. They propose a framework (i.e., LBT-P) that applies LLMs, such as RoBERTa [41], to extract semantic information. LBT-P uses knowledge distillation to compress an LLM into a small and fast model. Additionally, it also introduces knowledge preservation fine-tuning

to handle the challenge of catastrophic forgetting [43] of LLM. Although the issue is not a new concept, the automatic issue title generation task has not been studied until recently [20, 74]. Zhang et al. [74] propose to leverage the state-of-the-art BART [39] model to generate issue titles for bug reports. The experimental results show that fine-tuning BART can better generate issue titles than the prior state-of-the-art approach, i.e., iTAPE [20], based on sequence-to-sequence model [62].

8 CONCLUSION AND FUTURE WORK

In this work, we focus on the task of DBRD, especially on projects with a typical number of bug reports. We investigated how to combine the advantages of both traditional DBRD approach and LLMs and proposed Cupid. Cupid leverages ChatGPT to identify keywords as the input of the state-of-the-art traditional DBRD approach REP. We conduct a comprehensive evaluation on three datasets and compare Cupid with three baselines. The experimental results show that Cupid outperforms the state-of-the-art DBRD techniques in terms of Recall Rate@10 on all the datasets. Particularly, Cupid achieves high Recall Rate@10 scores ranging from 0.59 to 0.67 on all the datasets investigated.

In the future, we plan to investigate the ability of ChatGPT in Cupid under the setting of few-shot in-context learning. Furthermore, we are also interested in plugging ChatGPT into other bug report management tasks, such as bug component assignment and developer assignment.

AVAILABILITY

Our replication package is publically available at https://anonymous.4open.science/r/Cupid/.

REFERENCES

[1] (2) "too many requests in 1 hour. try again later." really annoying when will this be fixed? : Chatgpt. https://www.reddit.com/r/ChatGPT/comments/zm9g0n/too_many_requests_in_1_hour_try_again_later/, 2023. (Accessed on 05/05/2023).

[2] acheong08/chatgpt: Reverse engineered chatgpt api. https://github.com/acheong08/ChatGPT, 2023. (Accessed on 05/05/2023).

[3] Anonymized repository - anonymous github. https://anonymous.4open.science/r/Cupid/README.md, 2023. (Accessed on 05/05/2023).

[4] axios/axios: Promise based http client for the browser and node.js. https://github.com/axios/axios, 2023. (Accessed on 05/05/2023).

[5] Bugzilla. https://www.bugzilla.org/, 2023. (Accessed on 05/05/2023).

[6] Github. https://github.com/, 2023. (Accessed on 05/05/2023).

[7] [hadoop-17091] [jdk11] fix javadoc errors - asf jira. https://issues.apache.org/jira/browse/HADOOP-17091, 2023. (Accessed on 05/05/2023).

[8] Introducing chatgpt. https://openai.com/blog/chatgpt, 2023. (Accessed on 05/05/2023).

[9] Issue #131770 · microsoft/vscode. https://github.com/microsoft/vscode/issues/131770, 2023. (Accessed on 05/05/2023).

[10] Jira | issue & project tracking software | atlassian. https://www.atlassian.com/software/jira, 2023. (Accessed on 05/05/2023).

[11] ohmyzsh/ohmyzsh. https://github.com/ohmyzsh/ohmyzsh, 2023. (Accessed on 05/05/2023).

[12] Significant-gravitas/auto-gpt. https://github.com/Significant-Gravitas/Auto-GPT, 2023. (Accessed on 05/05/2023).

[13] [spark-31169] random forest in sparkml 2.3.3 vs 2.4.x - asf jira. https://issues.apache.org/jira/browse/SPARK-31169, 2023. (Accessed on 05/05/2023).

[14] [spark-33661] unable to load randomforestclassificationmodel trained in spark 2.x - asf jira. https://issues.apache.org/jira/browse/SPARK-33661, 2023. (Accessed on 05/05/2023).

[15] vuejs/vue: This is the repo for vue 2. for vue 3, go to https://github.com/vuejs/core. https://github.com/vuejs/vue, 2023. (Accessed on 05/05/2023).

[16] Amoui, M., Kaushik, N., Al-Dabbagh, A., Tahvildari, L., Li, S., and Liu, W. Search-based duplicate defect detection: An industrial experience. In *2013 10th Working Conference on Mining Software Repositories (MSR)* (2013), IEEE, pp. 173–182.

[17] Bang, Y., Cahyawijaya, S., Lee, N., Dai, W., Su, D., Wilie, B., Lovenia, H., Ji, Z., Yu, T., Chung, W., et al. A multitask, multilingual, multimodal evaluation of chatgpt on reasoning, hallucination, and interactivity. *arXiv preprint arXiv:2302.04023* (2023).

[18] Bettenburg, N., Just, S., Schröter, A., Weiss, C., Premraj, R., and Zimmermann, T. What makes a good bug report? In *Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of software engineering* (2008), pp. 308–318.

[19] Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al. Language models are few-shot learners. *Advances in neural information processing systems 33* (2020), 1877–1901.

[20] Chen, S., Xie, X., Yin, B., Ji, Y., Chen, L., and Xu, B. Stay professional and efficient: automatically generate titles for your bug reports. In *Proceedings of the 35th IEEE/ACM International Conference on Automated Software Engineering* (2020), pp. 385–397.

[21] Chiang, W.-L., Li, Z., Lin, Z., Sheng, Y., Wu, Z., Zhang, H., Zheng, L., Zhuang, S., Zhuang, Y., Gonzalez, J. E., Stoica, I., and Xing, E. P. Vicuna: An open-source chatbot impressing gpt-4 with 90%* chatgpt quality, March 2023.

[22] Chowdhery, A., Narang, S., Devlin, J., Bosma, M., Mishra, G., Roberts, A., Barham, P., Chung, H. W., Sutton, C., Gehrmann, S., et al. Palm: Scaling language modeling with pathways. *arXiv preprint arXiv:2204.02311* (2022).

[23] Christiano, P. F., Leike, J., Brown, T., Martic, M., Legg, S., and Amodei, D. Deep reinforcement learning from human preferences. *Advances in neural information processing systems 30* (2017).

[24] Dai, H., Liu, Z., Liao, W., Huang, X., Wu, Z., Zhao, L., Liu, W., Liu, N., Li, S., Zhu, D., et al. Chataug: Leveraging chatgpt for text data augmentation. *arXiv preprint arXiv:2302.13007* (2023).

[25] Deshmukh, J., Annervaz, K., Podder, S., Sengupta, S., and Dubash, N. Towards accurate duplicate bug retrieval using deep learning techniques. In *2017 IEEE International conference on software maintenance and evolution (ICSME)* (2017), IEEE, pp. 115–124.

[26] Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of NAACL-HLT* (2019), pp. 4171–4186.

[27] Fang, T., Yang, S., Lan, K., Wong, D. F., Hu, J., Chao, L. S., and Zhang, Y. Is chatgpt a highly fluent grammatical error correction system? a comprehensive evaluation. *arXiv preprint arXiv:2304.01746* (2023).

[28] Feng, Z., Guo, D., Tang, D., Duan, N., Feng, X., Gong, M., Shou, L., Qin, B., Liu, T., Jiang, D., et al. Codebert: A pre-trained model for programming and natural languages. In *Findings of the Association for Computational Linguistics: EMNLP 2020* (2020), pp. 1536–1547.

[29] Fu, T.-y., Lee, W.-C., and Lei, Z. Hin2vec: Explore meta-paths in heterogeneous information networks for representation learning. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management* (2017), pp. 1797–1806.

[30] He, J., Xu, L., Yan, M., Xia, X., and Lei, Y. Duplicate bug report detection using dual-channel convolutional neural networks. In *Proceedings of the 28th International Conference on Program Comprehension* (2020), pp. 117–127.

[31] Irsan, I. C., Zhang, T., Thung, F., Kim, K., and Lo, D. Multi-modal api recommendation. In *2023 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)* (2023), IEEE.

[32] Jalbert, N., and Weimer, W. Automated duplicate detection for bug tracking systems. In *2008 IEEE International Conference on Dependable Systems and Networks With FTCS and DCC (DSN)* (2008), IEEE, pp. 52–61.

[33] Jiao, W., Wang, W., Huang, J.-t., Wang, X., and Tu, Z. Is chatgpt a good translator? a preliminary study. *arXiv preprint arXiv:2301.08745* (2023).

[34] Joshi, S. D., and Chimalakonda, S. Rapidrelease-a dataset of projects and issues on github with rapid releases. In *2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR)* (2019), IEEE, pp. 587–591.

[35] Kochhar, P. S., Xia, X., Lo, D., and Li, S. Practitioners' expectations on automated fault localization. In *Proceedings of the 25th International Symposium on Software Testing and Analysis* (2016), pp. 165–176.

[36] Kung, T. H., Cheatham, M., Medenilla, A., Sillos, C., De Leon, L., Elepaño, C., Madriaga, M., Aggabao, R., Diaz-Candido, G., Maningo, J., et al. Performance of chatgpt on usmle: Potential for ai-assisted medical education using large language models. *PLoS digital health 2*, 2 (2023), e0000198.

[37] Lazar, A., Ritchey, S., and Sharif, B. Generating duplicate bug datasets. In *Proceedings of the 11th working conference on mining software repositories* (2014), pp. 392–395.

[38] Lee, J., Han, K., and Yu, H. A light bug triage framework for applying large pre-trained language model. In *37th IEEE/ACM International Conference on Automated Software Engineering* (2022), pp. 1–11.

[39] Lewis, M., Liu, Y., Goyal, N., Ghazvininejad, M., Mohamed, A., Levy, O., Stoyanov, V., and Zettlemoyer, L. Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics* (2020), pp. 7871–7880.

[40] Liu, P., Yuan, W., Fu, J., Jiang, Z., Hayashi, H., and Neubig, G. Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing. *ACM Computing Surveys 55*, 9 (2023), 1–35.

[41] Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., Levy, O., Lewis, M., Zettlemoyer, L., and Stoyanov, V. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692* (2019).

[42] Mahowald, K., Ivanova, A. A., Blank, I. A., Kanwisher, N., Tenenbaum, J. B., and Fedorenko, E. Dissociating language and thought in large language models: a cognitive perspective. *arXiv preprint arXiv:2301.06627* (2023).

[43] McCloskey, M., and Cohen, N. J. Catastrophic interference in connectionist networks: The sequential learning problem. In *Psychology of learning and motivation*, vol. 24. Elsevier, 1989, pp. 109–165.

[44] Mikolov, T., Chen, K., Corrado, G., and Dean, J. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781* (2013).

[45] Nguyen, A. T., Nguyen, T. T., Nguyen, T. N., Lo, D., and 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), pp. 70–79.

[46] Ouyang, L., Wu, J., Jiang, X., Almeida, D., Wainwright, C., Mishkin, P., Zhang, C., Agarwal, S., Slama, K., Ray, A., et al. Training language models to follow instructions with human feedback. *Advances in Neural Information Processing Systems 35* (2022), 27730–27744.

[47] Parikh, A., Täckström, O., Das, D., and Uszkoreit, J. A decomposable attention model for natural language inference. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing* (2016), pp. 2249–2255.

[48] Pennington, J., Socher, R., and Manning, C. D. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)* (2014), pp. 1532–1543.

[49] Qin, C., Zhang, A., Zhang, Z., Chen, J., Yasunaga, M., and Yang, D. Is chatgpt a general-purpose natural language processing task solver? *arXiv preprint arXiv:2302.06476* (2023).

[50] Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., Sutskever, I., et al. Language models are unsupervised multitask learners. *OpenAI blog 1*, 8 (2019), 9.

[51] Riazi, M. S., Rouani, B. D., and Koushanfar, F. Deep learning on private data. *IEEE Security & Privacy 17*, 6 (2019), 54–63.

[52] Robertson, S., Zaragoza, H., and Taylor, M. Simple bm25 extension to multiple weighted fields. In *Proceedings of the thirteenth ACM international conference on Information and knowledge management* (2004), pp. 42–49.

[53] Rodrigues, I. M., Aloise, D., Fernandes, E. R., and Dagenais, M. A soft alignment model for bug deduplication. In *Proceedings of the 17th International Conference on Mining Software Repositories* (2020), pp. 43–53.

[54] Runeson, P., Alexandersson, M., and Nyholm, O. Detection of duplicate defect reports using natural language processing. In *29th International Conference on Software Engineering (ICSE'07)* (2007), IEEE, pp. 499–510.

[55] Schütze, H., Manning, C. D., and Raghavan, P. *Introduction to information retrieval*, vol. 39. Cambridge University Press Cambridge, 2008.

[56] Sobania, D., Briesch, M., Hanna, C., and Petke, J. An analysis of the automatic bug fixing performance of chatgpt. *arXiv preprint arXiv:2301.08653* (2023).

[57] Stiennon, N., Ouyang, L., Wu, J., Ziegler, D., Lowe, R., Voss, C., Radford, A., Amodei, D., and Christiano, P. F. Learning to summarize with human feedback. *Advances in Neural Information Processing Systems 33* (2020), 3008–3021.

[58] Su, Y., Xing, Z., Peng, X., Xia, X., Wang, C., Xu, X., and Zhu, L. Reducing bug triaging confusion by learning from mistakes with a bug tossing knowledge graph. In *2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE)* (2021), IEEE, pp. 191–202.

[59] Sun, C., Lo, D., Khoo, S.-C., and Jiang, J. Towards more accurate retrieval of duplicate bug reports. In *2011 26th IEEE/ACM International Conference on Automated Software Engineering (ASE 2011)* (2011), IEEE, pp. 253–262.

[60] Sun, C., Lo, D., Wang, X., Jiang, J., and Khoo, S.-C. A discriminative model approach for accurate duplicate bug report retrieval. In *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering-Volume 1* (2010), pp. 45–54.

[61] Sureka, A., and Jalote, P. Detecting duplicate bug report using character n-gram-based features. In *2010 Asia Pacific software engineering conference* (2010), IEEE, pp. 366–374.

[62] Sutskever, I., Vinyals, O., and Le, Q. V. Sequence to sequence learning with neural networks. *Advances in neural information processing systems 27* (2014).

[63] Taori, R., Gulrajani, I., Zhang, T., Dubois, Y., Li, X., Guestrin, C., Liang, P., and Hashimoto, T. B. Stanford alpaca: An instruction-following llama model, 2023.

[64] Touvron, H., Lavril, T., Izacard, G., Martinet, X., Lachaux, M.-A., Lacroix, T., Rozière, B., Goyal, N., Hambro, E., Azhar, F., et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971* (2023).

[65] Touvron, H., Martin, L., Stone, K., Albert, P., Almahairi, A., Babaei, Y., Bashlykov, N., Batra, S., Bhargava, P., Bhosale, S., et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288* (2023).

[66] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. Attention is all you need. *Advances in neural information processing systems 30* (2017).

[67] Wang, X., Zhang, L., Xie, T., Anvik, J., and 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), pp. 461–470.

[68] Wolf, T., Debut, L., Sanh, V., Chaumond, J., Delangue, C., Moi, A., Cistac, P., Rault, T., Louf, R., Funtowicz, M., Davison, J., Shleifer, S., von Platen, P., Ma, C., Jernite, Y., Plu, J., Xu, C., Scao, T. L., Gugger, S., Drame,

M., Lhoest, Q., and Rush, A. M. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations* (Online, Oct. 2020), Association for Computational Linguistics, pp. 38–45.

[69] Xiao, G., Du, X., Sui, Y., and Yue, T. Hindbr: Heterogeneous information network based duplicate bug report prediction. In *2020 IEEE 31st International Symposium on Software Reliability Engineering (ISSRE)* (2020), IEEE, pp. 195–206.

[70] Xu, C., Sun, Q., Zheng, K., Geng, X., Zhao, P., Feng, J., Tao, C., and Jiang, D. Wizardlm: Empowering large language models to follow complex instructions. *arXiv preprint arXiv:2304.12244* (2023).

[71] Zaragoza, H., Craswell, N., Taylor, M. J., Saria, S., and Robertson, S. E. Microsoft cambridge at trec 13: Web and hard tracks. In *Trec* (2004), vol. 4, pp. 1–1.

[72] Zhang, T., Han, D., Vinayakarao, V., Irsan, I. C., Xu, B., Thung, F., Lo, D., and Jiang, L. Duplicate bug report detection: How far are we? *ACM Transactions on Software Engineering and Methodology* (2022).

[73] Zhang, T., Irsan, I. C., Thung, F., Han, D., Lo, D., and Jiang, L. Automatic pull request title generation. In *2022 IEEE International Conference on Software Maintenance and Evolution (ICSME)* (2022), IEEE Computer Society, pp. 71–81.

[74] Zhang, T., Irsan, I. C., Thung, F., Han, D., Lo, D., and Jiang, L. 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* (2022), pp. 1637–1641.

[75] Zhou, J., Zhang, H., and Lo, D. 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)* (2012), IEEE, pp. 14–24.