

SimBug : A similarity-driven approach to detect duplicate bug reports from multimodal bug reports

Nishat Sultana, Liakot Babu, Sean McClary
William and Mary, Williamsburg, VA, USA
nsultana@wm.edu, lbabu@wm.edu, sdmccclary@wm.edu

Abstract—Detecting duplicate bug reports is critical in software maintenance and quality assurance, especially for Android applications, where multimodal bug reports (text and images) are prevalent. This paper introduces SimBug, a similarity-driven framework leveraging multimodal inputs to identify and rank duplicate bug reports. By employing the Llava model and a carefully designed Few-shot prompting approach with tailored prefixes, we generate similarity scores between bug reports and rank them using standard metrics such as Mean Reciprocal Rank (MRR) and Mean Average Precision (MaP). The study demonstrates how meticulously crafted prompts, supporting a chain of thought reasoning tailored to Android-specific application descriptions, significantly improve the retrieval effectiveness of duplicate bug reports.

Index Terms—Duplicate Bug Report, Large Language Model, Information Retrieval(IR)

I. INTRODUCTION

In software development, the volume of user-reported bug reports can become overwhelming, especially for widely used applications. Developers can face thousands of reports in a short time, including duplicates or unrelated issues, which hinder their ability to prioritize and resolve problems efficiently. Modern software maintenance and quality assurance face significant challenges due to the inundation of duplicate bug reports in bug-tracking systems. These duplicates create inefficiencies, delay critical fixes, and hinder prioritization in large-scale software projects. The complexity intensifies in environments where bug reports include multimodal data—combining textual descriptions with images. Addressing this challenge requires an innovative solution capable of analyzing and integrating both modalities to accurately identify and rank duplicate reports. Bug tracking systems are inundated with duplicate bug reports, which can hinder the efficiency of software maintenance processes. Accurate detection of these duplicates, especially in a multimodal setting involving textual descriptions and images, remains an unresolved challenge. Images with texts proved to be very effective in this DBRD field[1]. This project introduces SimBug, a similarity-driven framework designed to detect duplicate bug reports in multimodal contexts, specifically for Android applications. SimBug leverages LLaVA, a state-of-the-art multimodal large language model, to process and analyze both textual and visual information. By employing carefully crafted 1-shot prompts with tailored prefixes, the model computes similarity scores between bug reports, effectively identifying duplicates.

These scores are ranked using robust metrics, such as Mean Reciprocal Rank (MRR) and Mean Average Precision (MAP), to evaluate the system’s effectiveness. A key innovation of SimBug lies in its approach to prompting. The framework incorporates chain-of-thought reasoning, enabling the model to provide not only similarity scores but also the rationale behind its decisions. This method ensures that the model leverages its pretrained knowledge in a targeted manner, aligning its reasoning with application-specific contexts and bug characteristics. The project tackles several challenges, including the careful selection of bug report corpora to avoid overwhelming the model and the design of prefixes that enrich contextual understanding. By focusing on these aspects, SimBug optimizes the performance of LLaVA without requiring extensive fine-tuning or large-scale multimodal datasets. The significance of this work extends beyond duplicate detection. It highlights the potential of multimodal large language models in streamlining software maintenance workflows, reducing redundancy, and improving developer productivity. SimBug represents a step forward in applying AI-driven solutions to real-world challenges, offering a robust and scalable approach for the effective management of bug-tracking systems. Duplicate bug reports, in particular, represent a significant challenge. If multiple reports of the same issue can be identified and grouped, developers can focus on implementing a single solution, saving valuable time and resources. Detecting duplicate bug reports is far from straightforward due to the inherent complexities of modern software systems and the diversity of user submissions.

Bug reports often vary in quality, with users having different technical expertise and communication styles. This variation can lead to incomplete, inaccurate, or ambiguous descriptions. Furthermore, bug reports frequently include a mix of modalities, such as textual descriptions, screenshots, or even videos, each requiring different processing techniques. Effectively combining and interpreting these multimodal inputs adds another layer of difficulty to the task.

To address this challenge, we introduce SimBug, a novel approach for detecting duplicate bug reports. SimBug leverages the power of large multimodal language models (LLMs) and advanced prompt engineering to process textual and visual information in bug reports simultaneously. Our approach involves designing optimized one-shot prompting with tailored prefixes to guide the LLM in generating similarity scores between bug reports. These scores are then used to rank potential duplicates effectively. In this paper, we make the

following contributions:

- 1) a one-shot with prefix prompt design is proposed to detect duplicate bug reports in 2937 issues across 30 projects.
- 2) the proposed system uses both visual and textual data from user bug reports.
- 3) The designed prompts lead to similarity scores that achieve comparable results to baseline methods, and a high similarity score for reports known to be duplicates.

To address this, we propose SimBug, a framework that:

- Leverages multimodal bug reports (text and images).
- Utilizes Few-shot prompting with the Llava model to compute similarity scores.
- Ranks similarity scores to identify duplicates, evaluated using MRR and MaP metrics.

Our approach focuses on designing prompts tailored to Android application descriptions and bug characteristics, enabling effective chain-of-thought reasoning.

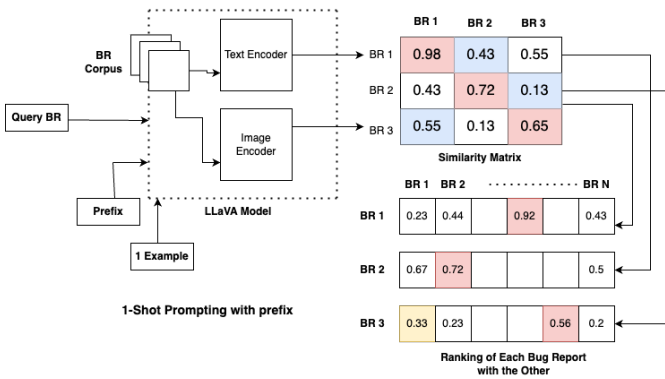


Fig. 1. SimBug Overview

The rest of this paper is organized as follows. In Section 2, we will define key concepts discussed in this research. Section 3 explains our methodology. The design of the experiment is explained in Sections 4 and 5. In Section 6 we evaluate our results. We explore threats to validity in Section 7. Section 8 describes other researchers attempts at detecting duplicate bug reports. We conclude in Section 9

II. BACKGROUND

A. The Growing Challenge of Duplicate Bug Reports

In modern software development, bug-tracking systems are central to managing issues, ensuring quality, and maintaining user satisfaction. However, one persistent challenge is the proliferation of duplicate bug reports. These duplicates often stem from different users reporting the same issue in varied formats, descriptions, and contexts. Such redundancy complicates the prioritization of tasks, delays issue resolution, and burdens software teams with unnecessary overhead. Addressing duplicate bug reports is therefore a critical need for improving the efficiency and accuracy of software maintenance workflows.

B. Multimodal Nature of Bug Reports

As applications, particularly Android apps, grow in complexity, bug reports have evolved from being purely textual to multimodal, including images such as screenshots or visual representations of issues. These visual elements provide critical context that textual descriptions alone cannot capture, such as UI inconsistencies or graphical glitches. While this multimodal approach enriches the information available for debugging, it also introduces challenges in processing and analyzing diverse data types. Traditional methods that rely solely on textual analysis are often insufficient to fully leverage this rich, multimodal data.

C. Limitations of Existing Approaches

Existing systems for detecting duplicate bug reports predominantly rely on text-based similarity measures, such as cosine similarity or embedding comparisons. While effective in simple scenarios, these approaches fall short when confronted with the multimodal nature of modern bug reports. Additionally, many of these methods lack the reasoning capability to align seemingly dissimilar textual and visual data effectively. Although embedding models like CLIP have attempted to bridge the gap between textual and visual inputs, their inability to incorporate contextual and task-specific knowledge remains a significant limitation.

D. The Role of Large Language Models (LLMs)

The advent of Large Language Models (LLMs) has revolutionized natural language processing, offering powerful tools for reasoning, summarization, and contextual understanding. Multimodal LLMs, such as LLaVA, extend these capabilities by integrating text and visual data, making them particularly suited for analyzing complex, multimodal bug reports. These models can process textual descriptions alongside images, generating similarity scores that encapsulate both modalities. Additionally, LLMs excel at incorporating tailored contextual information through prompting, enabling domain-specific applications without the need for extensive fine-tuning.

E. The Importance of Prompting Techniques

Prompting has emerged as a practical and scalable solution for leveraging LLMs in tasks with limited training data. By providing structured inputs, such as tailored prefixes and examples, prompting enables LLMs to perform specific tasks effectively without additional training. In the context of duplicate bug report detection, prompts can guide the model to consider not only textual and visual inputs but also the underlying reasoning, ensuring more accurate similarity scores. Techniques such as 1-shot prompting with prefixes have demonstrated remarkable success in aligning model outputs with the intended task.

Given the challenges of managing multimodal bug reports and the limitations of existing approaches, there is a pressing need for a framework that combines state-of-the-art multimodal processing with robust reasoning capabilities. Such a framework must be scalable, able to handle diverse datasets,

and optimized for identifying and ranking duplicate bug reports effectively. The development of SimBug addresses this need, leveraging LLMs and advanced prompting techniques to create a similarity-driven framework tailored for modern bug-tracking systems. By integrating textual and visual modalities and employing metrics like MRR and MAP, SimBug sets a new benchmark for duplicate bug detection in software development workflows. In this section, we will explain and introduce key concepts that will be discussed in this paper, such as bug reports, multi modality, prompting, few-shot, and prefixes.

III. APPROACH

Our approach involves detecting duplicate bug reports through multimodal information that combines text and images. The process starts with a query bug report, which contains both textual descriptions and associated images representing the issue. These inputs are processed separately by a Text Encoder for the textual component and an Image Encoder for the visual component. To enhance the reasoning and relevance of the output, a carefully designed prefix is provided, containing contextual information about Android applications and bug characteristics. Additionally, an example of a similarity score between two bug reports is used to guide the model in generating similarity scores. This approach leverages multimodal data and advanced prompting techniques to compute similarity scores effectively and accurately rank duplicate bug reports for software maintenance workflows. To implement this methodology, we use the LLaVA multimodal LLM, which integrates text and image processing capabilities with advanced reasoning techniques, making it a highly effective tool for multimodal bug report detection[2].

A. Prefix

The prefix is a meticulously crafted set of instructions that combines tailored descriptions to provide the model with essential contextual understanding[3]. It includes an application-specific description highlighted in blue, offering details about the purpose and functionality of the app, such as its usage in offline data collection and support for various input types. The bug description, emphasized in red, outlines potential issues in Android applications, including crashes, poor performance, and unexpected behaviors. The bug report details, marked in green, describe the structure and contents of bug reports, such as text, reproduction steps, device information, and images like screenshots. Finally, the similarity score explanation, shown in purple, provides a definition and criteria for measuring the similarity of bug reports, focusing on the location of the error, user actions, and application responses. This structured prefix ensures the LLM has a clear understanding of the task, leveraging both contextual and multimodal data to improve the accuracy of duplicate detection. The figure 2. represents a prefix that is fed into the Large Language Model (LLM) alongside each 1-shot prompt to enhance the detection of duplicate bug reports. The testing of prefix has gone through

multiple phases and the one in the figure was the best in the entire batch.

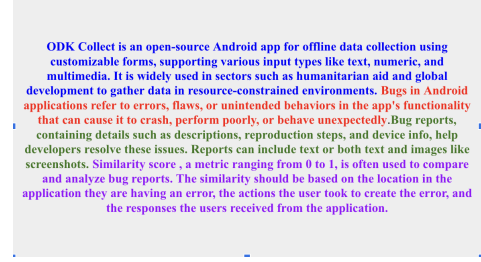


Fig. 2. Example of a Prefix for an Android application named ODK Collect

B. 1-shot Prompt

We used carefully designed prompts that introduce a "Chain-of-Thought" approach to guide the model in generating similarity scores for bug reports[4]. Like the prefix the prompt has gone through multiple stages of testing and this was the best of the batch. This particular structure of prompt provided the best output that were pretty close to the ground truth and was satisfactory. We have taken design ideas and from multiple designated research papers[5]. The first part of the prompt

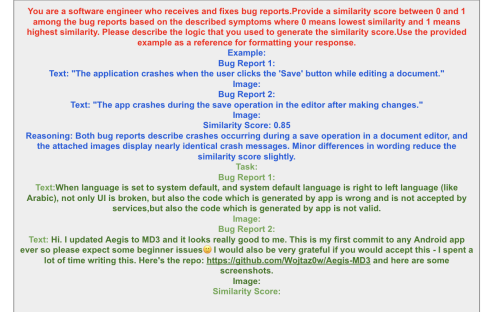


Fig. 3. An Example 1-Shot Prompt

in red establishes the task, asking the model to compute a similarity score between 0 and 1, where 0 indicates the lowest similarity and 1 indicates the highest. It emphasizes the need for justification, requiring the model to explain the reasoning behind its similarity score. To enhance understanding, the prompt includes a real-world example in blue, showing two bug reports with their textual descriptions, associated images, and a similarity score of 0.85. The reasoning for this score is explicitly provided, detailing how both reports describe similar crashes with minor differences in wording, justified through embeddings generated by the CLIP model. This example serves as a reference for the model's response format, ensuring alignment with the intended logic. The second part of the prompt in green presents two new bug reports the first one is the query and the second one the the corpus that needs to compare with the query and instructs the model to generate a similarity score along with a rationale, reinforcing the structured reasoning process.

C. Selection of the Bug Report Corpus for the Prompt

The selection of bug report in the corpus that will be given to compare with the query bug reports were a much tricky spot to handle. Where too many options made the LLM confused but also too less options results in onvious outputs without any variations. So we have tested with one query bug report with its true duplicate in 2,3,4 and 5 bug report corpus. We have just taken the similarity score for the true duplicate bug report. Only 2 bug reports per prompt to maximize the similarity score. This minimal context approach ensures better focus and relevance, allowing the model to effectively evaluate the similarity between the target reports without the noise introduced by additional bug reports.

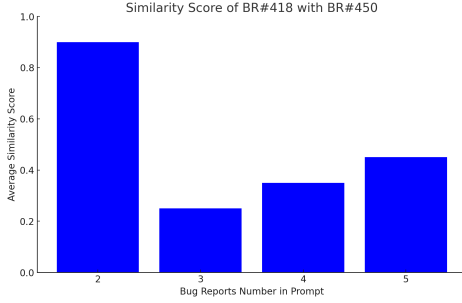


Fig. 4. Average similarity score of Bug Report #418 with Bug Report #450 across varying numbers of bug reports in the prompt on Android Application Aegis

Figure 4 illustrates the average similarity score of Bug Report #418 compared to Bug Report #450 across different numbers of bug reports included in the prompt (2, 3, 4, and 5). The y-axis represents the average similarity score, while the x-axis indicates the number of bug reports in the prompt. When only 2 bug reports are included, the similarity score is the highest, close to 0.9. As the number of bug reports increases (3, 4, and 5), the similarity scores drop significantly, indicating that the model’s performance in accurately detecting duplicates diminishes with additional bug reports in the prompt.

D. Selecting the Prompting Technique

Given the limited availability of multimodal data, as discussed in Section 4.1, it was neither feasible to fine-tune nor pre-train the large Llava model effectively. Consequently, we opted for the high-road approach of leveraging prompting as a solution. To achieve optimal results, we rigorously evaluated various prompting techniques, experimenting with different combinations of context, examples, and prefixes tailored to our task. Through this systematic evaluation, we identified the best-performing prompting strategy, which maximized the model’s ability to accurately compute similarity scores and rank duplicate bug reports, demonstrating the strength of prompting in overcoming data scarcity challenges. The figure demonstrates the comparison of similarity scores generated using different prompting techniques to evaluate one bug report (BR#418) from the Android application Aegis against its own report using the Llava model. The techniques include

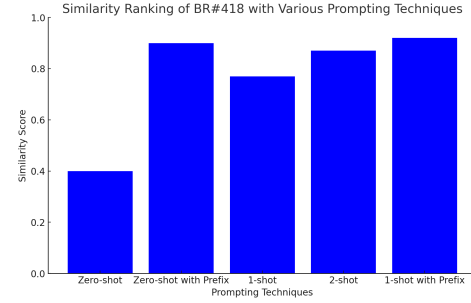


Fig. 5. Comparison Between Multiple Prompting Techniques

zero-shot, zero-shot with prefix, 1-shot, 2-shot, and 1-shot with prefix. Among these, 1-shot with prefix achieved the highest similarity score, nearly reaching 1.0, indicating the most accurate alignment with the query bug report. The inclusion of a carefully designed prefix provided contextual information tailored to the Android application and bug report descriptions, significantly improving the model’s reasoning capabilities. In contrast, zero-shot prompting without prefix yielded the lowest score, reflecting the importance of additional context. Although 2-shot and 1-shot without prefix performed better than zero-shot, they fell short of the precision achieved by 1-shot with prefix. This result highlights the effectiveness of combining a single example with a tailored prefix to enhance similarity detection, making 1-shot with prefix the optimal choice for this task.

E. Extracting Similarity Score from Model Response

The model’s responses are systematically stored in a CSV file for analysis, with each bug report pair being evaluated using the prompt three times. To ensure reliability, the similarity scores from these three runs are averaged to produce a single value for each pair. To automate the extraction of similarity scores from the CSV file, a Python script was developed that employs a regular expression to precisely locate and extract the similarity score values from the responses. This approach ensures consistency and accuracy in processing the results, enabling a streamlined analysis of the model’s performance across all bug report pairs.

IV. EXPERIMENTAL DESIGN

A. Research Questions

Leveraging multimodal information in detecting duplicate bug reports using Large Language Models (LLMs) involves combining textual descriptions and visual elements, such as screenshots, to provide a comprehensive understanding of reported issues. By utilizing the multimodal capabilities of advanced LLMs, like Llava, we can process and integrate both text and image data to compute similarity scores between bug reports. These models can be prompted with tailored prefixes and example-driven queries to analyze correlations between textual and visual inputs, enhancing the accuracy of duplicate detection. This approach enables LLMs to overcome limitations of single-modal analysis, ensuring a more robust

and context-aware detection mechanism for duplicate bug reports in complex software systems. Based on this context we have designed our first research question.

1) **RQ1**: How can we leverage Multi Modal information in the detection of Duplicate Bug Reports using Large Language Models (LLMs)?

Prompting in the context of detecting duplicate bug reports with multimodal information allows Large Language Models (LLMs) to leverage both textual and visual data without the need for extensive fine-tuning. By designing tailored prompts that include specific contextual information, such as application descriptions, bug details, and carefully selected examples, the model can effectively process and reason about the relationships between bug reports. Prompting enables the model to generate similarity scores by integrating the multimodal information and ranking potential duplicates. This approach demonstrates high performance by utilizing the LLM’s pre-trained knowledge, making it a practical solution for scenarios with limited data availability for fine-tuning or pretraining. As we have a scarcity of multi modal data we have decided to go with prompting and designed our second research question based on that.

2) **RQ2**: How Prompting performs in the context of detecting duplicate bug reports having multi modal information?

B. Dataset

For this project, we utilized a real-world dataset comprising 30 Android applications, with issues from their GitHub repositories categorized as bug reports. The dataset includes a total of 2,937 bug reports, out of which 500 are identified as duplicate reports. Additionally, 144 bug reports, including their duplicates, contain both textual descriptions and images, providing multimodal information. These 144 multimodal bug reports span across 26 of the 30 applications, highlighting the diverse nature of the dataset while emphasizing the potential for leveraging multimodal data in duplicate detection. This dataset serves as the foundation for evaluating our approach to detecting duplicate bug reports using multimodal information. Each bug report has a unique issue number.

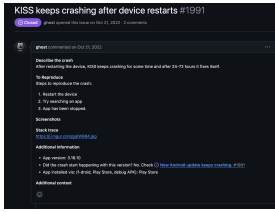


Fig. 6. Bug Reports with only Texts

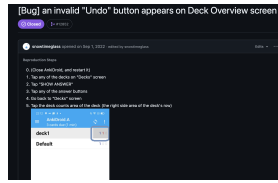


Fig. 7. Bug Reports with Text and Images

The images below shows two different bug reports with and without images. The first bug report, titled "[Bug] an invalid 'Undo' button appears on Deck Overview screen #12852," describes an issue in the AnkiDroid application where an invalid "Undo" button is displayed on the deck overview screen. It includes detailed reproduction steps and a screenshot

highlighting the issue. The second bug report, titled "KISS keeps crashing after device restarts #1991," documents a recurring crash in the KISS application following a device restart, with the crash resolving itself after 24-72 hours. It provides steps to reproduce the issue, a link to the stack trace, and additional context regarding the app version and installation method.

V. EXPERIMENTAL SETUP

The table below provides key parameters and configurations used for the experimental setup in evaluating the model’s performance on detecting duplicate bug reports. The model utilized is LLaVA-1.6-Vicuna-7B, running on an Nvidia A40 and Tesla-V100 GPU alternatively. The temperature setting for generation is set to 0.7, ensuring a balance between creativity and coherence. The output is capped at 1024 generated tokens, with the model quantized to 4-bit precision for efficient inference. For handling multimodal inputs, an image concatenation strategy of Grid alignment is employed. The testing subset includes 144 bug reports containing both text and images, spanning across 26 different applications, reflecting the diversity of the dataset. This setup ensures the evaluation captures both textual and visual aspects of the bug reports effectively.

Parameters	Value
Model	LLaVA-1.6-Vicuna-7B
Device	Nvidia-A40, Tesla-V100
Temperature	0.7
Generated Tokens	1024
Quantization	4-bit
Image Concatenation Strategy	Grid
Testing Subset	144 Bug Reports with Text and Images From 26 Applications

TABLE I
EXPERIMENTAL SETUP PARAMETERS AND CONFIGURATIONS

A. Baseline

CLIP (Contrastive Language-Image Pretraining) serves as the baseline model for detecting duplicate bug reports in a multimodal context, leveraging its capability to generate embeddings from both textual and visual inputs. CLIP is trained on a large-scale dataset of image-text pairs, enabling it to align textual and visual modalities within a shared embedding space. This alignment allows CLIP to compute cosine similarity scores between bug reports by analyzing their textual descriptions and associated screenshots. In the context of multimodal bug report detection, CLIP processes the textual description of a bug and the corresponding visual input, such as screenshots, to produce high-dimensional embeddings. These embeddings represent the semantic and visual content of the bug reports, enabling a direct comparison between pairs of reports. The cosine similarity between the embeddings is used as a metric to determine the degree of similarity, where higher

scores indicate closer alignment and potential duplication. While CLIP’s embedding-based approach provides a solid foundation for multimodal analysis, its limitation lies in its inability to incorporate reasoning or contextual details specific to bug reports. It treats the task as a straightforward similarity comparison, lacking the capability to integrate detailed application-specific contexts or chain-of-thought reasoning. Despite this, CLIP remains a reliable and computationally efficient baseline for multimodal bug report detection, offering robust initial insights into the similarity of bug reports based on their combined textual and visual features.

B. LLaVa Alternatives

1) *Open Source LLMs*: **LLama3**, the latest iteration of the LLaMA series, offers advanced capabilities in natural language understanding and generation, making it a strong contender for multimodal duplicate bug report detection[6]. While LLama3 is primarily a text-based model, its extensibility allows for integration with external vision encoders to process visual inputs such as screenshots. This hybrid approach can enable LLama3 to handle the multimodal requirements of bug report detection effectively. With its fine-tuned attention mechanisms and scalability, LLama3 can analyze complex textual descriptions, incorporating nuanced contexts such as application-specific details and user actions leading to bugs. By integrating embeddings from images and text, LLama3 can generate similarity scores and provide detailed justifications, enabling it to compete with dedicated multimodal models like LLaVA. **ChatGPT**, particularly the versions with vision capabilities like GPT-4 Vision, is a robust alternative for multimodal duplicate bug report detection[7]. With its native ability to process both textual and visual inputs, ChatGPT can directly analyze bug descriptions alongside screenshots to identify duplicates[8]. The model’s extensive reasoning and conversational abilities make it highly effective in chain-of-thought prompting, allowing for detailed justifications of similarity scores between bug reports. Moreover, ChatGPT’s pre-trained knowledge and adaptability ensure that it can generate meaningful insights, even in zero-shot or few-shot scenarios. By tailoring prompts with application-specific prefixes and leveraging ChatGPT’s multimodal capabilities, developers can achieve accurate and context-aware bug report similarity detection, aligning its performance closely with LLaVA’s multimodal approach.

VI. RESULTS AND FINDINGS

A. Running LLaVa with Prompt Template

To check how well Llava performs, we have tested Llava on our 1-shot prompt with prefix for a query bug report 1264 from the android application Aegis with a corpus of bug reports among those bug reports 418 is the true duplicate for bug report 450. The selection criterion for corpus bug reports is to check all the bug reports available before the query bug report to give it a fair shot. Figure 8 gives an image of the corpus selection Our prompting technique has assigned the true duplicate the highest score as predicted. This

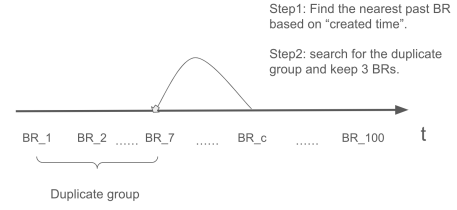


Fig. 8. Selection of the Bug Report Corpus for the LLM Prompt

result underscores the model’s capability to accurately rank bug reports based on their similarity, effectively distinguishing the true duplicate from unrelated or loosely related bug reports. Such findings highlight the robustness of the similarity scoring mechanism in identifying duplicate bug reports in a structured and multimodal evaluation. The figure illustrates the

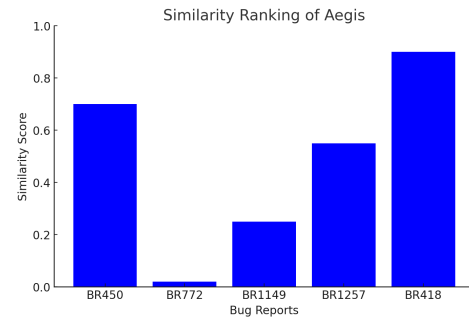


Fig. 9. Similarity Measure of Bug Report #1264(Where the True Duplicate is Bug Report #418) with 1-shot Prompting with Prefix

similarity scores of Bug Report #1264 with other bug reports (BR450, BR772, BR1149, BR1257, and BR418) using the 1-shot prompting technique with a carefully designed prefix. The similarity measure is computed to identify duplicate bug reports, with Bug Report #418 being the true duplicate. The results show that BR418 achieves the highest similarity score, nearing 1.0, correctly identifying it as the duplicate. In contrast, other bug reports exhibit significantly lower similarity scores, with BR450 scoring moderately high (0.7) and others (BR772, BR1149, and BR1257) having much lower values. This plot demonstrates the effectiveness of 1-shot prompting with prefix in accurately distinguishing true duplicates from non-duplicates in a multimodal bug detection framework.

B. Responses From the Model

Shared textual descriptions and visual evidence contribute to high similarity scores when the reports exhibit overlapping contexts, such as similar UI issues or identical symptoms.As LLMs provides diverse set of responses the figure below represents some of the finest response that we got from the model.The responses not only has a similarity score but also a reasoning why the similarity score was scored. The figure 11 is one of best responses that we got form out prompting to LLaVa.

Based on the symptoms described, I would estimate a similarity score of 0.7. Both images show a user interface with a two-factor authentication app, which is the primary functionality of Aegis Authenticator. The similarity is high because both images are related to the same app and share a common purpose.

Fig. 10. Response One

Based on the symptoms provided, the cosine similarity score between the two bug reports would be quite high, likely around 0.9 or 1. This is because both reports describe the same issue with the Aegis Authenticator app, including the problem of a blank screen with a red button, the inability to log in, and the need to reinstall the app. The similarity in the symptoms and the context of the issue suggests that these are two different instances of the same problem.

Fig. 11. Response Two

The similarity score for these bug reports is 0.85. This score is calculated using the following logic:

1. "UI Broken": Both bug reports mention that the UI is broken. The app's behavior when set to the system default language results in UI issues, which is consistent across both reports.
2. "Language Handling": Both reports mention that the code generated by the app is wrong when using system default language, especially when set to right to left languages like Arabic. This is a critical issue that both reports agree on.
3. "Code Validity": Both reports state that the generated code is not valid. This is a critical issue that both reports share.
4. "User Experience": Both reports mention that there are beginner issues, suggesting that the user is new to the development process or the app itself.
5. "Code Sharing": One report explicitly states the user has shared the code, which is a detail that both reports share.

Fig. 12. Response Three

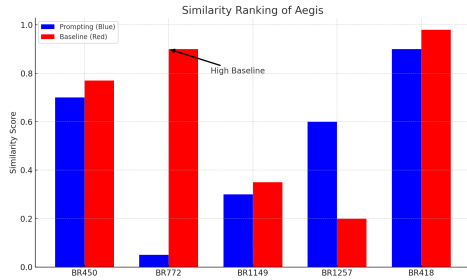


Fig. 13. Comparison of similarity scores for Aegis bug reports using Prompting(Blue) and Baseline(Red) methods,

C. Comparing with Baselines

The baseline for comparing our results utilizes embeddings generated by the CLIP model[9]. This approach processes both the textual descriptions and the corresponding images from the bug reports, converting them into embeddings within a shared multimodal space. The cosine similarity between these embeddings is then calculated to measure the degree of similarity between bug reports. By leveraging the CLIP model's ability to align textual and visual inputs, the baseline establishes a robust metric for evaluating the similarity of multimodal bug reports. This comparison serves as a foundational standard against which the performance of our model and prompting-based approach is assessed, ensuring a comprehensive evaluation of the duplicate detection framework.

The plot illustrates the similarity ranking of bug reports for the Aegis application, comparing the scores generated by two methods: Prompting (Blue) and Baseline (Red). Each pair of bars represents a specific bug report (e.g., BR450, BR772, BR1149, BR1257, BR418), with the y-axis showing the similarity scores. For most bug reports, the similarity scores generated by the baseline approach are slightly higher than those generated by prompting. Notably, in the second pair (BR772), the baseline similarity score reaches 0.9, significantly outperforming the prompting score of 0.05. This is highlighted by a thick arrow pointing to this pair, labeled as "High Baseline." The comparison emphasizes the differences between the two methods, showcasing cases where the baseline model excels and areas where prompting-based methods

may need improvement. The final pair (BR418) demonstrates near-perfect similarity scores for both methods, highlighting their ability to identify true duplicates effectively.

VII. EVALUATION

To evaluate our information retrieval[10] technique we have used two different kind of metrics called MRR and MaP. Mean Reciprocal Rank (MRR), MRR measures how quickly the first relevant result appears in the ranking. It is calculated as the reciprocal of the rank of the first relevant result, averaged across all queries. A high MRR indicates that relevant results are consistently ranked at the top. Mean Average Precision (MAP), MAP evaluates the overall precision of the ranking by averaging the precision scores at each relevant result's rank for a query and then averaging these scores across all queries. A higher MAP reflects better ranking of all relevant results, not just the top one[11].

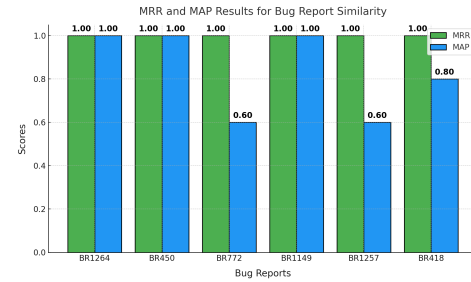


Fig. 14. MRR and MAP Results for Bug Report Similarity for application Aegis. The green bars represent the Mean Reciprocal Rank (MRR), indicating the system's ability to rank the most relevant bug report first. The blue bars represent the Mean Average Precision (MAP), highlighting the overall precision of ranking all relevant bug reports for each query.

The results in the Figure 13. indicate that the system demonstrates excellent performance in identifying the most relevant bug reports, as shown by the perfect MRR (1.0) for all queries. This means that for every query, the top-ranked bug report is always the most relevant, showcasing the system's ability to prioritize the best match effectively. However, the MAP values vary across queries, reflecting differences in the precision of ranking all relevant bug reports. Queries such as BR1264, BR450, and BR1149 achieve perfect MAP (1.0), indicating that all relevant reports are ranked in the correct order. In contrast, queries like BR772 and BR1257 have lower MAP scores of 0.6, suggesting that while the top results are accurate, other relevant bug reports are not ranked as effectively. Query BR418 achieves a moderate MAP (0.8), showing relatively better precision but still leaving room for improvement. Overall, the system excels in identifying the most relevant report but could benefit from enhanced precision in ranking additional relevant reports for certain queries.

VIII. THREATS TO VALIDITY

A. External Validity

A significant threat to the validity of this study is the limited number of duplicate bug reports available in the dataset. With

only 500 duplicate reports out of a total of 2,937 and 144 multimodal reports, the framework’s performance may not generalize well to larger datasets or systems with higher duplication rates. This scarcity of duplicates can skew the evaluation metrics, as the framework is tested in a context where true duplicates are relatively rare. To address this, we designed a robust prompting mechanism that maximizes the utility of each available duplicate by ensuring consistent and accurate similarity scoring.

B. Internal Validity

The reliance on LLaVA-7B, a smaller multimodal large language model, introduces constraints on the model’s capacity to process complex multimodal inputs. While smaller models are more computationally efficient, they may lack the reasoning depth and generalization capabilities of larger models. This can lead to lower quality similarity scores, especially for nuanced cases where textual and visual data require intricate contextual understanding. To mitigate this, we employed carefully designed 1-shot prompts with tailored prefixes to provide additional contextual information, effectively enhancing the model’s reasoning capabilities despite its smaller size.

IX. RELATED WORK

The detection of duplicate bug reports has been a persistent challenge in software maintenance, with several prior studies proposing various approaches to address this issue. Traditional methods largely rely on textual similarity measures, such as cosine similarity, TF-IDF, and embedding-based approaches using models like BERT or its variants. These methods effectively handle textual descriptions but fail to incorporate visual elements, which are crucial in multimodal bug reports. Duplicate bug report detection (DBRD) has been a long-standing challenge addressed by both traditional and deep learning-based methods. While traditional approaches like REP struggle to capture the semantics of bug reports, and deep learning approaches falter in smaller repositories, recent advancements, such as combining traditional methods with state-of-the-art LLMs like ChatGPT, have shown significant improvements, achieving state-of-the-art results in Recall Rate@10 across diverse datasets[7]. Issue reports are critical for maintaining and improving software, but traditional text-based unimodal models for automated issue classification often yield suboptimal accuracy. Recent advancements in multimodal techniques, which integrate text, images, and code from issue reports, have demonstrated significant performance improvements, achieving 5.07% to 14.12% higher F1-scores compared to unimodal models across multiple projects[12]. Video-based bug reports are becoming increasingly common for documenting GUI-centered bugs, but their management poses challenges due to the nuanced visual patterns required for accurate analysis. Recent advancements, such as the Janus approach, leverage vision transformers and adaptive video alignment techniques to effectively capture subtle visual and textual patterns. Janus achieves state-of-the-art performance with an mRR of 89.8% and mAP of 84.7%,

outperforming prior work by approximately 9% in duplicate detection tasks derived from video-based bug reports[13]. As screen-recordings and screenshots become prevalent in reporting GUI-centered bugs, managing these visual artifacts, especially in crowd-sourced testing scenarios, has become increasingly challenging. Techniques like Tango address this problem by leveraging computer vision, optical character recognition, and text retrieval to detect duplicate video-based bug reports. In evaluations across 4,860 detection tasks, Tango achieves 83% accuracy in ranking duplicate videos within the top-2 results, and user studies indicate it reduces developer effort by over 60%, highlighting its effectiveness and practicality[14]. LLaVA, a multimodal extension of CLIP, addresses this gap by integrating text and visual inputs natively. Although LLaVA has been applied in diverse domains, its application to bug report similarity detection, particularly in multimodal contexts, is novel. Existing studies using LLaVA focus on general-purpose visual-text alignment tasks rather than the domain-specific and structured task of duplicate bug report detection.

Feature	CUPID	Multimodal Issue Classification	Janus	Tango	SimBug
Modality	Text	Text, Images, Code	Text, Video	Video	Text, Images
Technique	LLM + Traditional Approach	Multimodal Model	Vision Transformers + Video Alignment	Computer Vision + OCR + Text Retrieval	Multimodal LLM + Prompting
Type of Bug Reports	Text-based bug reports	Issue reports with text, images, and code	Video-based bug reports	Video-based bug reports	Text and image-based bug reports
Evaluation Metric	Recall Rate@10	F1-Score	mRR/mAP	Top-2 Accuracy	MRR/MAP
Performance	0.602 - 0.654	5.07% - 14.12% higher than unimodal models	89.8% / 84.7%	83% (Top-2)	1.0 MRR, 1.0/0.6/0.8 MAP across dataset

TABLE II
COMPARATIVE ANALYSIS OF DUPLICATE BUG REPORT DETECTION APPROACHES

X. CONCLUSION AND FUTURE WORK

This study introduces SimBug, a similarity-driven framework designed to detect duplicate bug reports in multimodal contexts, leveraging the capabilities of the LLaVA-7B model and advanced prompting techniques. By combining textual and visual data, SimBug effectively computes similarity scores and ranks bug reports using robust metrics like Mean Reciprocal Rank (MRR) and Mean Average Precision (MAP). Despite operating with a smaller model and a dataset limited in duplicate bug reports, the framework demonstrates strong performance,

particularly in identifying the most relevant duplicate. Key to this success is the use of tailored 1-shot prompts with prefixes, which guide the model’s reasoning process and enhance its contextual understanding. Additionally, averaging similarity scores across multiple runs ensures the reliability of results, mitigating the effects of variability in the model’s responses. These findings underscore the potential of multimodal LLMs combined with effective prompting to streamline bug report management, improve software maintenance workflows, and reduce redundancy in issue tracking systems.

While SimBug achieves significant milestones in duplicate bug report detection, there are several avenues for future exploration. Expanding the dataset to include a broader range of applications, platforms, and multimodal contexts will enhance the framework’s generalizability and robustness. Exploring the integration of larger multimodal models, such as LLaVA-13B or hybrid models combining text and vision-specific encoders, could further improve performance, especially for complex or nuanced bug reports. Additionally, the development of more sophisticated prompting techniques, including dynamic or adaptive prompts tailored to specific queries, holds promise for achieving even greater accuracy. Future work should also evaluate the computational efficiency and scalability of the framework in real-world scenarios, particularly in large-scale software projects. Incorporating user feedback and additional evaluation metrics, such as processing time and user satisfaction, will ensure that the framework is practical and impactful in diverse software development environments.

REFERENCES

- [1] J. Wang, M. Li, S. Wang, T. Menzies, and Q. Wang, “Images don’t lie: Duplicate crowdtesting reports detection with screenshot information,” *Information and Software Technology*, vol. 110, pp. 139–155, 2019.
- [2] H. Liu, C. Li, Y. Li, and Y. J. Lee, “Improved baselines with visual instruction tuning,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2024, pp. 26 296–26 306.
- [3] “Prompt prefixing,” 2024. [Online]. Available: <https://www.promptlayer.com/glossary/prompt-prefixing#:~:text=Prompt%20context%for%20the%20interaction.>
- [4] J. Wei, X. Wang, D. Schuurmans, M. Bosma, F. Xia, E. Chi, Q. V. Le, D. Zhou *et al.*, “Chain-of-thought prompting elicits reasoning in large language models,” *Advances in neural information processing systems*, vol. 35, pp. 24 824–24 837, 2022.
- [5] S. Schulhoff, M. Ilie, N. Balepur, K. Kahadze, A. Liu, C. Si, Y. Li, A. Gupta, H. Han, S. Schulhoff *et al.*, “The prompt report: A systematic survey of prompting techniques (2024),” *arXiv preprint arXiv:2406.06608*.
- [6] A. Dubey, A. Jauhri, A. Pandey, A. Kadian, A. Al-Dahle, A. Letman, A. Mathur, A. Schelten, A. Yang, A. Fan *et al.*, “The llama 3 herd of models,” *arXiv preprint arXiv:2407.21783*, 2024.
- [7] J. Achiam, S. Adler, S. Agarwal, L. Ahmad, I. Akkaya, F. L. Aleman, D. Almeida, J. Altenschmidt, S. Altman, S. Anadkat *et al.*, “Gpt-4 technical report,” *arXiv preprint arXiv:2303.08774*, 2023.
- [8] OpenAI, “Chatgpt website,” <https://chat.openai.com>, 2024, accessed: 2024-12-10.
- [9] A. Radford, J. W. Kim, C. Hallacy, A. Ramesh, G. Goh, S. Agarwal, G. Sastry, A. Askell, P. Mishkin, J. Clark *et al.*, “Learning transferable visual models from natural language supervision,” in *International conference on machine learning*. PMLR, 2021, pp. 8748–8763.
- [10] GeeksforGeeks, “What is information retrieval?” 2022, accessed: 2024-12-11. [Online]. Available: <https://www.geeksforgeeks.org/what-is-information-retrieval/>
- [11] L. Monigatti, “Evaluation metrics for search and recommendation systems,” May 2024, accessed: 2024-12-11. [Online]. Available: <https://weaviate.io/blog/retrieval-evaluation-metrics>
- [12] C. Kwak, P. Jung, and S. Lee, “A multimodal deep learning model using text, image, and code data for improving issue classification tasks,” *Applied Sciences*, vol. 13, no. 16, p. 9456, 2023.
- [13] Y. Yan, N. Cooper, O. Chaparro, K. Moran, and D. Poshyvanyk, “Semantic gui scene learning and video alignment for detecting duplicate video-based bug reports,” in *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*, 2024, pp. 1–13.
- [14] N. Cooper, C. Bernal-Cárdenas, O. Chaparro, K. Moran, and D. Poshyvanyk, “It takes two to tango: Combining visual and textual information for detecting duplicate video-based bug reports,” in *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. IEEE, 2021, pp. 957–969.