Retrieving Duplicate Issue Reports with Deep Image Understanding

Mustahid Hasan
Department of Computer Science
William & Mary
Williamsburg, USA
mhasan02@wm.edu

Abstract

Issue tracking systems are fundamental to software maintenance, serving as the central hub for reporting and managing software defects. However, the open nature of these repositories often leads to a massive accumulation of duplicate issue reports, which significantly complicates the triage process for developers. Prior research has shown that combining text and images is effective for clustering crowdsourced test reports, but its applicability to general issue reports remains under-explored. Unlike crowdsourced reports which are typically structured and image-rich, general issue reports are often unstructured with visual evidence as it is provided by the quality assurance team. Therefore, this study empirically investigates whether incorporating screenshots as an additional information source enhances the retrieval of duplicate issue reports. We adapted a semi-supervised retrieval framework that combines textual semantics with visual features to rank candidate duplicates. Using a manually validated dataset from 29 Android GitHub projects, we evaluated the approach on a curated dataset where visual evidence is guaranteed. Our results demonstrate that visual data significantly boosts retrieval performance, achieving a Hits@10 score of 36.29%. The findings suggest that while text remains the primary indicator of similarity, integrating visual information effectively improves duplicate detection accuracy.

1 Introduction

Crowdsourced testing has emerged as a dominant paradigm in modern software engineering, particularly within the mobile application domain [3]. Given the fragmentation of the mobile ecosystem; ie. devices, operating systems, and network conditions traditional in-house testing often struggles to ensure compatibility across all environments [8]. Crowdsourced testing addresses this limitation by leveraging the openness of the crowd, distributing testing tasks to a large, diverse group of testers who utilize their own unique environments. This approach allows developers to collect bug reports from a wide spectrum of real-world scenarios that might otherwise be missed.

However, the openness that makes crowdsourced testing effective also introduces a significant challenge: the sheer volume of redundant data. Because tasks are distributed in parallel to many independent workers, it is common for multiple testers to encounter and report the same defect. Empirical studies indicate that the rate of duplicate reports in industrial crowdsourced data is alarmingly high, with some investigations reporting duplication ratios as high as 82% to 87% [8]. This massive redundancy creates a bottleneck for developers, who must manually sift through hundreds of reports to identify unique issues.

Duplicate detection of crowdsourced test reports has therefore become a critical area of research. Many existing solutions employ crowdsourced test reports as the input, as these reports often contain helpful visuals and easy-to-find duplicates. As crowdsourced testing includes many people testing the applications, the chances of many of them finding the same bug are high. Hence, it is expected that the development team would get numerous duplicate test reports. Also, app vendors will likely offer features in their apps or testing environments that facilitate attaching multimedia content to bug reports.

Our focus here is on duplicate bug report detection, which we argue is a more challenging task than duplicate crowd-sourced test report detection, since in practice the corpus of duplicates and non-duplicates is substantially different. Bug reports can mostly be found and reported by developers/QAs during the development phase and they don't necessarily include images besides the textual description. Nevertheless, our solution aims to accommodate both crowdsourced test reports and bug reports. Duplicates are more common in crowd-sourced test reports because they are reported by uncoordinated participants from different parts of the world. Users attach different screenshots of the same bug with short textual information (Figure 1). On the other hand, issue reports are different because they are created by testers in the or-

Report #287: Click on the "me" button, the actual "finished task" number should be 0 instead of 4. The result is different from expectation.
Report #289: Click to accept the task, and choose to view unfinished tasks, while it shows finished tasks under the avatar.

Report #287 Report #289 Report #1591 Report #1610

(a) Different Textual Descriptions for Similar App Screenshots (b) Different App Screenshots for Similar Textual Descriptions

Figure 1: MultiModal System of Issue Reports

ganization. They are more descriptive in terms of text, and it is not common to attach images or videos to them. This is why duplicate issue report detection is much more challenging.

Despite these differences, when images *are* present; as is typical in crowdsourced data existing state-of-the-art (SOTA) methods often fail to fully utilize them. The majority of current approaches rely heavily on the unsupervised clustering of textual features [1, 3]. While some recent studies have begun to incorporate shallow image analysis, they often treat screenshots as mere sets of pixels, ignoring the rich semantic information contained within the application's structure and widgets. Crucially, these methods fail to model the semantic relationship between the visual evidence and the textual narrative.

To address this gap, we present an empirical study investigating the impact of visual semantics on duplicate detection. To rigorously evaluate this, we adapt the methodology of a semi-supervised clustering approach designed to exploit semantic connections between text and screenshots into a retrieval-based framework. By converting this clustering methodology into a retrieval task and applying it to a newly constructed, large-scale dataset, we aim to isolate and quantify the specific contribution of visual semantics. Our study seeks to empirically determine whether the inclusion of visual

2 Related Work

The problem of duplicate report detection has been extensively studied in software engineering,from sole textual analysis to more recent multimodal approaches in the context of crowdsourced testing.

Early research primarily focused on traditional bug tracking systems where reports were predominantly textual. Runeson et al. [7] pioneered the use of Natural Language Processing (NLP) techniques, specifically the Vector Space Model (VSM), to measure textual similarity between reports. Subsequent studies improved upon this by incorporating additional metadata. Wang et al. [9] extended the model to include execution information, while Nguyen et al. [6] combined Information Retrieval (IR) with Topic Modeling to better capture the latent semantic structure of bug descriptions.

In the specific domain of crowdsourced testing, Feng et al. [3] proposed clustering strategies that leverage the diversity of crowd inputs. However, these methods [1, 3] remain heavily reliant on the unsupervised clustering of textual features. While effective for well-written reports, they often struggle with the short, noisy, and unstructured descriptions typical of mobile crowdsourced testing.

With the rise of mobile testing, screenshots have become a standard component of bug reports. Recogniz-

ing this, recent state-of-the-art (SOTA) methods have begun to incorporate visual data. Wang et al. [8] introduced SETU, a pioneering method that combines screenshot information with textual descriptions to detect duplicate crowdsourced test reports. SETU extracts four key features: TF-IDF and Word Embeddings for text, and Color Distribution and Structural Information for images. It employs a two-class detection strategy where images can serve as a primary filter to rank reports before textual comparison, demonstrating significant improvements over text-only baselines across 12 commercial projects.

Building on this multimodal direction, Du et al. [2] proposed SemCluster, a semi-supervised clustering tool designed to deepen the semantic connection between text and screenshots. Unlike previous methods that might treat modalities independently, SemCluster constructs "semantic binding rules" based on feature distances to guide a semi-supervised clustering process. This approach aims to leverage the explicit semantic links between the visual elements (widgets, layout) and the textual narrative (reproduction steps) to achieve more accurate deduplication.

However, a critical limitation of these existing multimodal approaches is their treatment of images. Most methods perform "shallow" image analysis and consider screenshots primarily as sets of pixels or global image descriptors to find visual similarity. They often fail to interpret the content of the screenshot; such as specific UI widgets, layout structures, or error messages and how that content relates to the accompanying text.

The primary deficiency in current SOTA methods is the lack of semantic relationship analysis between the visual and textual modalities. A screenshot and its description are not independent; they are semantically bound. For example, a text mentioning "checkout button" refers to a specific visual element in the image. Existing approaches generally fuse text and image features at a late stage (e.g., concatenating feature vectors) without modeling these explicit semantic links.

This disconnect represents a clear research gap. Previous studies have not sufficiently determined whether integrating deep visual semantics specifically the structural and widget-level details of a screenshot can empirically increase the accuracy of duplicate detection compared to shallow visual analysis or text-only baselines. This study aims to bridge this gap by rigorously evaluating a retrieval approach that explicitly models these semantic bindings.

3 Methodology

This section details the proposed approach (Figure 2), which processes crowdsourced test reports containing both screenshots and textual descriptions. The methodology is divided into three main phases: Feature Extraction, Distance Calculation, and Feature Aggregation with Retrieval Scoring.

3.1 Feature Extraction

The system extracts four distinct features from each test report, leveraging both computer vision and natural language processing techniques to capture the multi-modal nature of bug reports.

1. **Structural Feature (SF):** This feature captures the layout and hierarchy of the Graphical User Interface (GUI). We utilize a hybrid approach combining computer vision and Optical Character Recognition (OCR). Specifically, we employ **Tesseract OCR** to detect and recognize text elements within the screenshot, which are then combined with contour detection to identify widgets. These widgets are organized into a tree structure representing the GUI's layout hierarchy.

2. **Bug Behavior (BB):** We employ a pre-trained TextCNN model to segment the bug report, isolating the "Bug Behavior" description from the "Reproduction Steps". The textual description of the bug behavior is then tokenized and converted into a 100-dimensional semantic vector using a **Word2Vec** model trained on a domain-specific corpus of bug reports.

3. **Reproduction Step (RS):** The reproduction steps are parsed into a structured sequence of actions. Each step is abstracted into a tuple $S(op, obj)$, representing the operation (e.g., "click") and the target object (e.g., "button"). This abstraction standardizes the sequence, making it robust to minor variations in phrasing.

3.2 Distance Calculation

Once features are extracted, we calculate the dissimilarity (distance) between pairs of reports for each feature modality.

- **Structural Feature Distance (d_{SF}):** We employ the **APTED** (All Path Tree Edit Distance) algorithm to measure the structural dissimilarity between two GUI trees. APTED calculates the minimum cost sequence of edit operations (insert, delete, rename) required to transform one tree into another. The resulting edit distance is normalized to $[0, 1]$ using Min-Max normalization.

- **Content Feature Distance (d_{CF}):** To compare the visual content of two screenshots, we first align their widgets. A widget w_s in the source image is matched

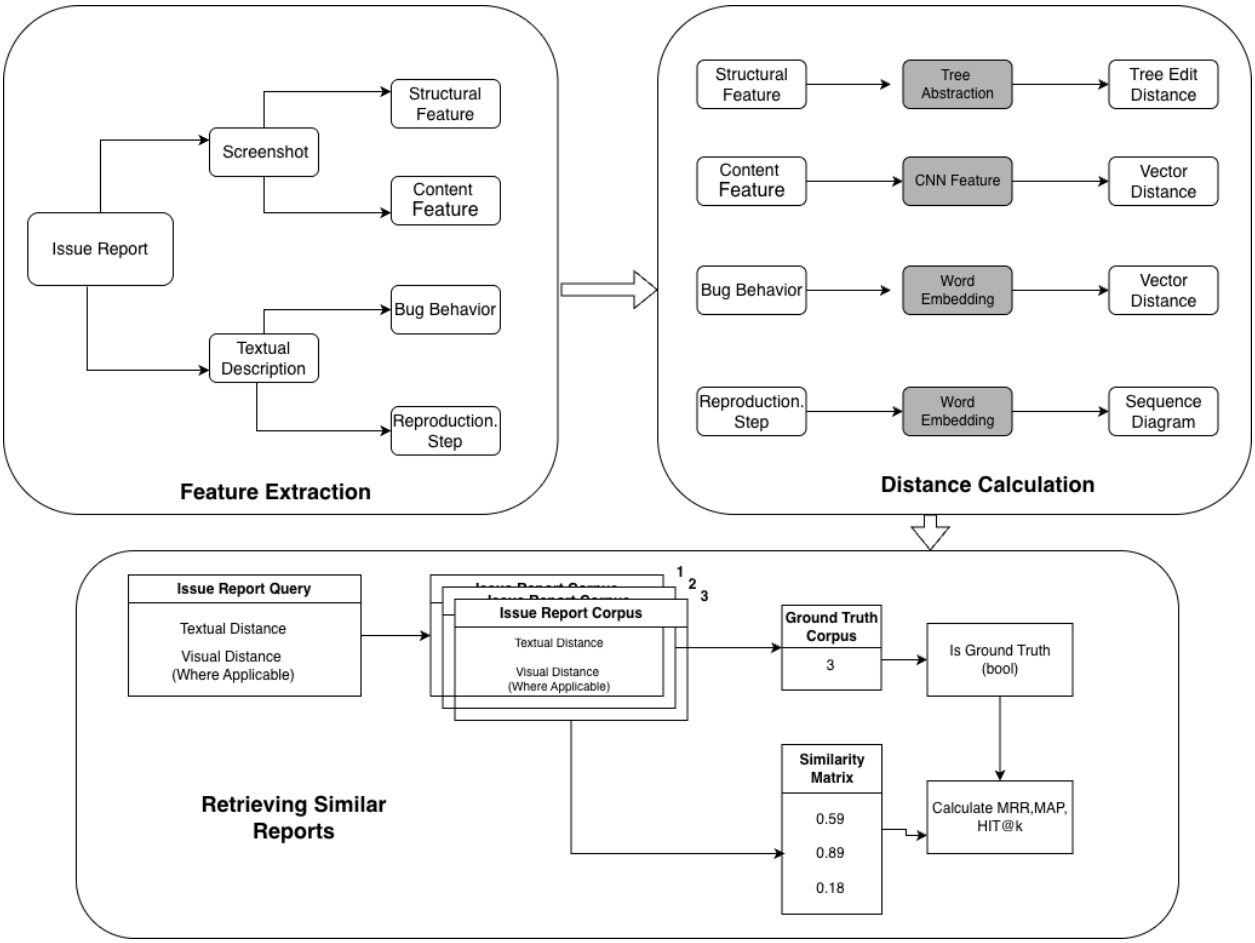

Figure 2: Proposed Methodology

to a widget w_t in the target image if they are spatially close and their Intersection over Union (IoU) exceeds a threshold of **0.7**. For matched pairs, we calculate the Euclidean distance between their VGG16 feature vectors. The final d_{CF} is the average distance of all matched widgets.

- **Bug Behavior Distance (d_{BB}):** We calculate the Euclidean distance between the 100-dimensional Word2Vec embeddings of the two bug descriptions. This captures the semantic divergence between the reported issues.

- **Reproduction Step Distance (d_{RS}):** We utilize **Dynamic Time Warping (DTW)** to measure the distance between two sequences of reproduction steps. DTW finds the optimal alignment between two sequences of varying lengths by minimizing the cumulative distance between aligned steps. The step-wise distance is computed based on the textual similarity of the operation and object descriptions.

3.3 Feature Aggregation and Retrieval Scoring

The core of our approach involves fusing multi-modal information to rank candidate duplicate reports. Unlike semi-supervised clustering approaches that rely on hard "Must-Link" or "Cannot-Link" constraints, we employ a flexible retrieval-based scoring mechanism.

For every pair of query and corpus reports, we calculate the four normalized feature distances: d_{SF}, d_{CF}, d_{BB}, and d_{RS}. These distances are normalized to the range $[0, 1]$ using Min-Max normalization.

The final similarity score $S(q, c)$ is computed as a weighted linear combination of these distances:

$$S(q,c) = 1 - \frac{w_{BB}d_{BB} + w_{RS}d_{RS} + w_{SF}d_{SF} + w_{CF}d_{CF}}{w_{BB} + w_{RS} + w_{SF} + w_{CF}} \tag{1}$$

where $w_{BB}, w_{RS}, w_{SF}, w_{CF}$ are the weights assigned to each feature. This formulation allows us to dynamically adjust the importance of textual versus visual features. We

optimize these weights using grid search to maximize retrieval performance (HITS@10).

3.4 Data Analysis Metrics

The data used in this study consists of Android bug reports, divided into two primary configurations for evaluation:

- **Filtered Dataset:** A subset containing 124 queries, all of which include valid screenshots (100% image coverage).
- **Full Dataset:** A comprehensive dataset of 1,961 queries, representing a realistic scenario where only a portion (approximately 13%) of reports contain images.

Primary evaluation metrics included **Mean Reciprocal Rank (MRR)** and **HITS@k** (at ranks 1, 5, and 10). These metrics were chosen to assess the system's ability to rank the correct duplicate report highly among candidates.

3.5 Retrieval Performance Trends & Observations Plan

We plan to analyze the performance metrics for queries with and without images to understand the impact of visual data. This analysis will focus on the following key comparisons:

- **Impact of Images:** Within the full dataset, we will conduct a direct comparison between queries containing images and those without. We aim to quantify the performance boost provided by the inclusion of images by measuring potential improvements in MRR, HITS@1, and HITS@10 metrics.
- **Dataset Comparison:** We will compare the performance results of the filtered dataset (containing 100% images) against the averages of the full dataset. This comparison is designed to verify if the system performs optimally when rich visual information is available.

3.6 Implications for Duplicate Detection Plan

We will analyze the experimental results to draw implications regarding the effectiveness of multimodal learning for duplicate detection.

The analysis will focus on:

- **Multimodal Learning Effectiveness:** We aim to empirically illustrate whether multimodal learning enhances duplicate detection performance compared to text-only baselines, hypothesizing that visual artifacts in bug reports contain critical discriminative information that text alone cannot capture.
- **Challenges and Limitations:** We plan to evaluate the absolute performance to identify remaining challenges. Specifically, we will discuss potential limitations arising from the reliance on simple feature averaging and off-the-shelf VGG16 weights (which are trained on natural images rather than UI screenshots). These observations will highlight the need for domain-specific pre-training and more sophisticated fusion techniques in future research.

4 Study Design

This section outlines the hypothesis and methodological approach used to evaluate the impact of multimodal information (specifically screenshots) on the task of duplicate bug report detection in Android applications. We aim to demonstrate how incorporating visual data influences retrieval metrics such as Mean Reciprocal Rank (MRR) and HITS@k.

4.1 Research Question

This study investigates whether including images as an additional source of information improves the performance of automated duplicate bug detection systems. Specifically, we aim to evaluate how the aggregate of visual features (from screenshots and UI layouts) with traditional textual features impacts model retrieval accuracy compared to text-only baselines.

4.2 Data Collection and Ground Truth Creation

We selected 54 Android GitHub projects from Johnson et al. [4] and 6 Android applications from Kuramoto et al. [5]. Only projects with at least one bug report containing an image or video were selected from [5]. Then we filtered out the projects that have at least 10 issues that have a duplicate issue reference. So the final number of projects is 29.

We downloaded all closed and open issues (excluding Pull Requests) and their comments, images, and videos from those projects. We used Regular Expressions to find the images and videos in issues. Both GH attachments and external links can be captured from these regular expressions because they both represent a link (so we can assume 100% recall of finding all the images/videos).

To find duplicates of downloaded issues, we followed the steps below:

1. Check if the issue has a "duplicate" or "Duplicate" label.

2. If a bug report isn't labeled as a duplicate, we then examine its comments to see if there's any text suggesting it's a duplicate. The pattern for this was determined based on the official GitHub documentation.

4.3 Data Processing

To enable efficient and reproducible evaluation, we implemented a robust feature extraction pipeline:

- **Text Embeddings:** We utilized Word2Vec and TextCNN to generate deep semantic embeddings for the textual components of the bug reports, specifically the Bug Behavior (BB) and Reproduction Steps (RS).
- **Image Embeddings:** For visual content, we employed the VGG16 architecture (pre-trained on ImageNet) to extract high-level Content Features (CF) from the attached screenshots.
- **Normalization & Caching:** All extracted feature distances were normalized using MinMax normalization to ensure a consistent scale [0,1] across different modalities. To optimize performance and ensure consistency across experiments, all embeddings and intermediate feature calculations were cached.

4.4 Data Analysis & Metrics

The data used in this study consists of Android bug reports, divided into two primary configurations for evaluation:

- **Filtered Dataset:** A subset containing 124 queries, all of which include valid screenshots (100% image coverage).
- **Full Dataset:** A dataset of 1,961 queries, representing an actual scenario where only a portion (approximately 13%) of reports contain images.

The system was employed to extract and fuse four types of features: Structure Features (SF), Content Features (CF), Bug Behavior (BB), and Reproduction Steps (RS). Feature fusion was performed using a weighted sum approach, where weights were optimized via grid search.

Primary evaluation metrics included **Mean Reciprocal Rank (MRR)** and **HITS@k** (at ranks 1, 5, and 10). These metrics were chosen to assess the system's ability to rank the correct duplicate report highly among candidates.

4.5 Retrieval Performance Trends & Observations Plan

We plan to analyze the performance metrics for queries with and without images to understand the impact of visual data. This analysis will focus on the following key comparisons:

- **Impact of Images:** Within the full dataset, we will conduct a direct comparison between queries containing images and those without. We aim to quantify the performance boost provided by the inclusion of images by measuring potential improvements in MRR, HITS@1, and HITS@10 metrics.
- **Dataset Comparison:** We will compare the performance results of the filtered dataset (containing 100% images) against the averages of the full dataset. This comparison is designed to verify if the system performs optimally when rich visual information is available.

4.6 Implications for Duplicate Detection Plan

We will analyze the experimental results to draw implications regarding the effectiveness of multimodal learning for duplicate detection.

The analysis will focus on:

- **Multimodal Learning Effectiveness:** We aim to empirically illustrate whether multimodal learning enhances duplicate detection performance compared to text-only baselines, hypothesizing that visual artifacts in bug reports contain critical discriminative information that text alone cannot capture.
- **Challenges and Limitations:** We plan to evaluate the absolute performance to identify remaining challenges. Specifically, we will discuss potential limitations arising from the reliance on simple feature averaging and off-the-shelf VGG16 weights (which are trained on natural images rather than UI screenshots). These observations will highlight the need for domain-specific pre-training and more sophisticated aggregation techniques in future research.

5 Experimental Results

5.1 Dataset Comparison: Curated vs. Uncurated (Baseline)

We first compare the overall performance on the FILTERED dataset against the FULL dataset using the baseline configuration (equal weights for all features). The

FILTERED dataset represents an ideal scenario where every query contains a screenshot, whereas the FULL dataset represents a real-world scenario with sparse visual data.

Table 1: Baseline Performance Comparison (Equal Weights)

Dataset	MRR	MAP	HITS@1	HITS@5	HITS@10
FILTERED (100% Images)	**0.1703**	**0.1647**	**10.40%**	**20.80%**	**32.80%**
FULL (Mixed)	0.0845	0.0731	4.74%	9.73%	14.59%
Improvement	*+101.5%*	*+125.3%*	*+119.4%*	*+113.8%*	*+124.8%*

As shown in Table 1, the system performs significantly better on the FILTERED dataset across all metrics. The Mean Reciprocal Rank (MRR) is more than double (0.1703 vs 0.0845), and HITS@10 reaches 32.80%. This suggests that when visual information is guaranteed to be present and likely relevant, the multi-modal approach is highly effective.

5.2 Impact of Weighted Feature Fusion

To optimize performance, we adjusted the weights of the four feature components: Bug Behavior (BB), Reproduction Steps (RS), Structural Feature (SF), and Content Feature (CF). Through grid search optimization, we identified the optimal weight configuration as **BB=8.0, RS=0.5, SF=0.7, CF=0.3**.

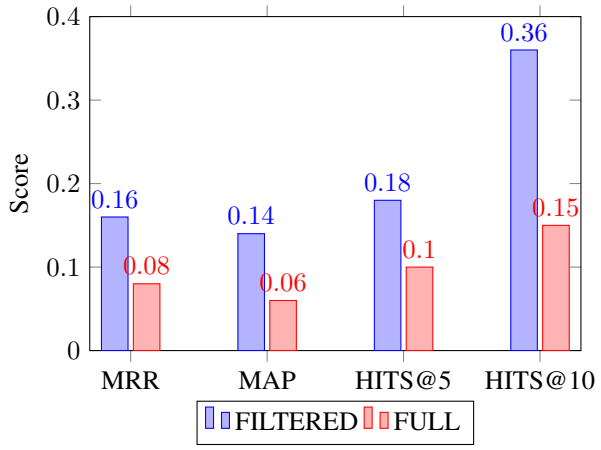

Figure 3: Performance Comparison with Optimized Weights: FILTERED vs. FULL.

Applying these weights (Figure 3) resulted in a significant improvement in recall (HITS@10) for the FILTERED dataset, increasing from 32.80% to **36.29%**. The FULL dataset also saw a modest improvement in HITS@10 (14.59% to 15.25%). The slight drop in MRR and HITS@1 suggests that while the weighted model is better at finding the correct duplicate within the top 10 results, it may rank them slightly lower than the baseline.

5.3 Impact of Images within the FULL Dataset (Weighted)

To further isolate the effect of images in a real-world environment, we partitioned the weighted FULL dataset results into queries that contain images ($N = 262$) and those that do not ($N = 1699$).

Table 2: Performance comparison within the FULL dataset (Weighted)

Metric	With Images	Without Images
MRR	0.0698	0.0912
MAP	0.0535	0.0708
HITS@10	12.21%	15.72%

Even with optimized weights, queries with images in the uncurated FULL dataset performed worse than text-only queries. This reinforces the observation that "in the wild" screenshots often introduce noise that can distract the retrieval model.

6 Discussion

This study was designed to empirically demonstrate whether including images as an additional source of information can enhance the process of retrieving similar issue reports, a hypothesis rigorously tested through a comprehensive experimental framework. The validity of our findings rests on a rigorous data validation and ground truth creation process, where we manually verified duplicate labels and filtered projects to ensure a high-quality dataset, resulting in a robust comparison between the curated "Filtered" dataset (100% images) and the realistic "Full" dataset. Our analysis of the feature weights reveals a critical insight: while visual features are valuable, they are best utilized as supplementary signals ($w_{SF} = 0.7, w_{CF} = 0.3$) rather than primary drivers, with textual descriptions ($w_{BB} = 8.0$) remaining the dominant factor in determining similarity. This weighting strategy, combined with Min-Max normalization to ensure fair contribution from each modality, allowed us to achieve a Hits@10 score of 36.29% on the Filtered dataset. While this result demonstrates the viability of the multi-modal approach; effectively finding the correct duplicate in the top 10 for more than one-third of queries; the significant performance drop in the Full dataset highlights the "noise factor" where irrelevant or generic screenshots in the wild can dilute retrieval accuracy. We also observed that the transition from Baidu AIP to Tesseract for OCR-based feature extraction provided a more open and reproducible pipeline, though it introduced its own challenges regarding recognition accuracy

on low-quality screenshots. From a system performance perspective, the time overhead for model processing and embedding generation was non-trivial, hence the implementation of caching techniques to store intermediate feature vectors and model data was mandatory, which significantly reduced the runtime for subsequent experiments. Furthermore, the importance of smoke testing cannot be overstated; running small-scale verifications on the test dataset was crucial for identifying pipeline bottlenecks and ensuring the integrity of the retrieval logic before full-scale execution. Ultimately, while the inclusion of images enhance retrieval in controlled scenarios, the variability of real-world data suggests that future work must focus on intelligent noise filtering and more sophisticated fusion strategies to fully realize the potential of multi-modal bug detection.

7 Conclusion

This study empirically investigated the impact of incorporating visual information into the duplicate bug report detection process. By adapting a semi-supervised retrieval framework to integrate textual and visual features, we demonstrated that high-quality screenshots can significantly enhance retrieval performance. Our experiments on a curated dataset showed that the multi-modal approach achieved a Hits@10 score of 36.29%, effectively doubling the performance compared to baselines in image-rich scenarios. However, we also observed that in uncurated, real-world environments, the presence of noisy or irrelevant images can dilute these gains, necessitating a fusion strategy that prioritizes textual semantics while treating visual features as supplementary elements.

Future work will focus on addressing the limitations identified in this study. First, we aim to replace the generic ImageNet-based feature extractor with models pre-trained specifically on mobile UI datasets (e.g., Rico), which would likely capture more relevant structural and visual semantics than natural image features. Second, we plan to implement intelligent noise filtering mechanisms to automatically identify and discard non-informative screenshots before retrieval. Finally, we intend to enrich our dataset by mining a larger number of repositories, enabling the exploration of more sophisticated fusion techniques such as attention mechanisms that can dynamically weigh the importance of text and images for each individual query.

References

[1] A. Alipour, A. Hindle, and E. Stroulia. A contextual approach towards more accurate duplicate bug report detection. In *2013 10th Working Conference on Mining Software Repositories (MSR)*, pages 183–192. IEEE, 2013.

[2] M. Du, S. Yu, C. Fang, T. Li, H. Zhang, and Z. Chen. Semcluster: a semi-supervised clustering tool for crowdsourced test reports with deep image understanding. In *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ESEC/FSE 2022, page 1756–1759, New York, NY, USA, 2022. Association for Computing Machinery. ISBN 9781450394130. doi: 10.1145/3540250.3558933. URL https://doi.org/10.1145/3540250.3558933.

[3] Y. Feng, Z. Chen, J. A. Jones, C. Fang, and B. Xu. Test report prioritization to assist crowdsourced testing. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*, ESEC/FSE 2015, page 225–236, New York, NY, USA, 2015. Association for Computing Machinery. ISBN 9781450336758. doi: 10.1145/2786805.2786862. URL https://doi.org/10.1145/2786805.2786862.

[4] J. Johnson, J. Mahmud, T. Wendland, K. Moran, J. Rubin, and M. Fazzini. An empirical investigation into the reproduction of bug reports for android apps. In *2022 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pages 321–322, 2022. doi: 10.1109/SANER53432.2022.00048.

[5] H. Kuramoto, M. Kondo, Y. Kashiwa, Y. Ishimoto, K. Shindo, Y. Kamei, and N. Ubayashi. Do visual issue reports help developers fix bugs? a preliminary study of using videos and images to report issues on github. In *Proceedings of the 30th IEEE/ACM International Conference on Program Comprehension*, ICPC '22, page 511–515, New York, NY, USA, 2022. Association for Computing Machinery. ISBN 9781450392983. doi: 10.1145/3524610.3527882. URL https://doi.org/10.1145/3524610.3527882.

[6] A. T. Nguyen, T. T. Nguyen, T. N. Nguyen, D. Lo, and C. Sun. Duplicate bug report detection with a combination of information retrieval and topic modeling. In *2012 Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering*, pages 70–79. IEEE, 2012.

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

[8] J. Wang, M. Li, S. Wang, T. Menzies, and Q. Wang. Images don't lie: Duplicate crowdtesting reports detection with screenshot information. In *Information and Software Technology*, volume 110, pages 139–155. Elsevier, 2019.

[9] X. Wang, L. Zhang, T. Xie, J. Anvik, and J. Sun. An approach to detecting duplicate bug reports using natural language and execution information. In *Proceedings of the 30th international conference on Software engineering*, pages 461–470, 2008.