Towards Enhancing IR-based Bug Localization Leveraging Texts and Multimedia from Bug Reports

Shamima Yeasmin
Department of Computer Science
University of Saskatchewan
shamima.yeasmin@usask.ca

Chanchal K. Roy
Department of Computer Science
University of Saskatchewan
chanchal.roy@usask.ca

Kevin Schneider
Department of Computer Science
University of Saskatchewan
kevin.schneider@usask.ca

Mohammad Masudur Rahman
Faculty of Computer Science
Dalhousie University
masud.rahman@dal.ca

Kartik Mittal
Department of Computer Science
University of Saskatchewan
wte705@mail.usask.ca

Ryder Hardy
Department of Computer Science
University of Saskatchewan
hkt946@mail.usask.ca

***Abstract*—Software bug reports often miss critical information, delaying their resolution. The last decade has seen a growing trend of combining textual and multimedia information from software artifacts (e.g., bug reports, and programming questions) to support various software engineering tasks (e.g., duplicate bug report detection, and bug reproduction). However, none of the studies performs a fine-grained analysis of the multimedia information attached to bug reports. Hence, it is not clear what their attached images or videos contain or whether they could help identify software bugs or errors automatically. In this paper, we conduct a preliminary study that investigates the presence or prevalence of key elements in 1,469 images attached to 1,000 bug reports and demonstrate their potential to support IR-based bug localization. We have several interesting findings. First, our analysis suggests that the attached images to visual bug reports contain a mix of UI components, programming components, and regular text. Second, our analysis using an LLM (e.g., GPT4o) suggests that it can extract the key elements from the attached images effectively, posing a suitable alternative to human annotators. Finally, our experiments suggest that the multimedia information extracted from the attached images can enhance the performance of a traditional technique for bug localization by improving 34.06% of its search queries.**

***Index Terms*—Visual bug reports, attached images, bug localization, LLM**

I. Introduction

A picture is worth a thousand words. Over the last decade, there has been a growing trend of software developers combining textual and multimedia information to solve their development and maintenance problems [19]. For example, in Stack Overflow Q&A site, users often attach multimedia content (e.g., screenshots) to their posted questions. Similarly, in Bugzilla, software users often attach multimedia content to their issue reports [19]. Recently, GitHub launched a new feature that allows its users to share their multimedia documents (e.g. videos) as a part of the bug reports [6]. When a bug report lacks a succinct description of an encountered bug, the visual items have the potential to complement it with relevant information. In visual bug reports, the attached images or videos may convey various information including UI screenshots, error messages, and programming code. However, to date, visual bug reports have not received much attention. It is also not clear which types of content are present in their attached images or videos and what their prevalence is.

Several existing works leverage multimedia content (e.g., images or videos) to support various software maintenance tasks including duplicate bug report detection [8], bug reproduction [10, 13, 16], and GUI testing [9]. Unfortunately, only a little research has been done to better understand the characteristics of visual bug reports. In recent work, Nayebi [19] investigates the images attached to Stack Overflow questions and Bugzilla reports to determine if they are informative and essential to understand the submitted problems. Nayebi [19] categorizes these images into multiple classes and investigates how the developers perceive the value of these images through a developer study. Recently, Kuramoto et al. [14] conduct a preliminary study to analyze the characteristics of visual bug reports by comparing them with non-visual bug reports. They found that visual bug reports contain fewer text than that of non-visual bug reports, but this does not affect their resolution times. Although the above studies shed some light on the characteristics of visual bug reports, they do not perform any fine-grained analysis to understand the actual contents of the images that are attached to visual bug reports.

There exists a large body of work on bug localization [20, 22, 24, 25, 26, 27, 28, 29] that leverage various textual contents from a bug report (e.g., text, stack traces, commit diffs) to find the corresponding bug. Surprisingly, they often overlook the relevance of the attached multimedia information (e.g., images, videos). A recent work [15] demonstrates how GUI operations from the videos of Mobile apps can benefit the text-retrieval based methods for bug localization. However, there is still a marked lack of research that investigates how the multimedia information (e.g., attached images) from bug reports could help identify the bugs within the source code. Our work is the first that fills this significant gap by extracting and utilizing information from attached images in bug reports and thus improving bug localization performance, which makes our work novel.

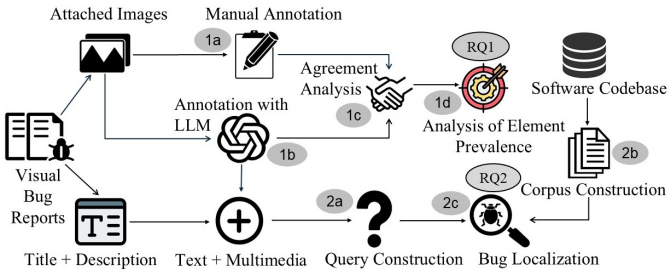

Fig. 1. Schematic diagram of the conducted Study

In this paper, we conduct a preliminary study using 1,000 visual bug reports from 223 projects (a) to better understand the characteristics of their attached images, and (b) to determine whether they are useful for finding software bugs. We ask two important research questions as follows.

- **RQ1: What are the key elements found in the images attached to visual bug reports?** Our analysis suggests that the attached images contain a mix of User Interface (34.5%), programming components (16.2%), and regular text (49.3%). We also found that commercial Large Language Models (LLM) such as *GPT-4o* can extract these elements from the attached images effectively, which has significant implications.
- **RQ2: Does incorporating multimedia information (e.g., attached images) into Information Retrieval-based methods improve their performance in bug localization?** We found that the multimedia information from the attached images can help a traditional technique for bug localization (e.g., *Apache Lucene* [5]) achieve better performance (e.g., ≈21% higher Hit@1 than baseline) by improving 34% of its search queries.

II. Study Methodology

Fig. 1 shows the schematic diagram of our conducted study. First, we manually analyze the bug reports and their attached images, and identify key elements from the images (e.g., UI controls, and programming code). We also repeat identifying key elements using the GPT-4o API [2] to determine the LLM's effectiveness for the task (RQ1). An automatic multimedia information analysis could help us combine text and multimedia information in bug localization. Second, we leverage the extracted information from the attached images in bug localization using Information Retrieval (RQ2). In this section, we discuss the major steps of our study as follows.

A. Dataset Construction

We collect our dataset from an existing benchmark of Kuramoto et al. [14]. The benchmark dataset contains a total of 1,230 videos and 18,760 images attached to 226,286 bug reports that were submitted to 4,173 software projects at GitHub. As our research focuses on images rather than videos, we select the visual bug reports that contain only images. We then examined 1,615 projects from the dataset, analyzed hundreds of visual bug reports, and identified those containing ground truth information (e.g., changed source documents). Through this process, we compiled a final dataset of 1,000 visual bug reports from 223 open-source projects on GitHub, as ground truth information is essential for conducting experiments in bug localization. The entire process required approximately 110 person-hours to complete.

B. Analysis of Bug Reports and their Attached Images

Manual analysis. In our manual analysis, we employ the *snowball approach* [11] to identify unique elements in the images attached to visual bug reports (Step 1a, Fig. 1). In the snowball approach, we start with a few key categories and then gradually add more categories or sub-categories. Our selection of these categories was inspired by the existing literature [14, 19]. For each attached image, we first determine if it matches one of three major categories: *UI components*, *Programming components*, and *Natural language text* (see Table II). If yes, we attempt to find more granular items (e.g., stack traces, Git commands) from the image. Otherwise, we check if it is a novel category. We involve one human annotator for the manual analysis. The annotator was provided with the necessary training and relevant guidelines before the training process. The annotator spent ≈85 hours analyzing 1,000 bug reports and 1,469 attached images.

Automated analysis. Despite clear, explicit guidelines, human annotations could suffer from subjective bias. Involving multiple annotators is a way to mitigate such bias. However, given the extreme costs of annotation, we decided to use another feasible alternative. We annotate the attached images to bug reports using an LLM-based image recognition technique, namely *GPT-4o* [2] (Step 1b, Fig. 1). The Open AI GPT-4o has shown remarkable capabilities in various problem-solving tasks including image annotation. We first construct an appropriate prompt outlining the key categories and sub-categories and instruct GPT-4o to annotate the attached images. (The prompt can be found in the replication package [4]). Then we execute the prompt with the LLM for each of the 1,469 attached images and collect the results containing the annotated key elements.

Agreement analysis between manual and automated annotations. We perform agreement analysis between the two types of annotations above using Cohen's Kappa measure (Step 1c, Fig. 1). Cohen's kappa is a statistical measure to quantify the level of agreement between two raters (e.g., annotators, judges, observers) [3]. Our analysis has reported an *almost perfect* agreement score of 80.49%, indicating that an LLM-based solution (e.g., GPT-4o) can effectively extract key elements from the images attached to bug reports. In our work, the disagreements between the human annotator and GPT-4o were resolved by involving another human annotator. This process took approximately 10 hours to complete.

C. Construction of Queries and Corpus

Once the key elements from the attached images are identified, we apply them to bug localization to determine their benefit. In particular, we incorporate them in an Information Retrieval (IR)-based approach and attempt to improve its

search queries (Step 2, Fig. 1). We construct search queries and corpus for our experiment as follows.

Query construction. We construct two types of search queries for our experiment. First, we capture both the *title* and *description* from each bug report to design our *baseline query* (i.e., T+D). We perform standard natural language preprocessing on these fields, involving stop word or punctuation removal and token splitting. We use a standard list [1] to discard the stop words and punctuation marks. We also use appropriate regular expressions to split the structured tokens (e.g., `getMethodName` is divided into `get`, `Method`, and `name`). Second, to construct the *extended query* (i.e., T+D+I), we additionally extract the tokens from the key elements found in the attached images. First, we use GPT-4o to convert an image containing text into machine-readable text. Then, we apply the same preprocessing steps (e.g., stop word removal, token splitting) to the machine-readable text and combine it with the baseline query to make the extended query (Step 2a, Fig. 1). Given that the incorporation of multimedia information in query construction is automatic, it has important implications.

Corpus construction. We collect all the source code documents from a software project to construct our corpus (Step 2b, Fig. 1). We perform the same standard natural language preprocessing on each source document and index them carefully. From the indices, one can locate the documents containing one or more keywords from a given query, which helps rank the documents according to their relevance. To index and search our corpus, we use the Apache Lucene [5]. The tool has been widely used for indexing and searching in various software engineering tasks including bug localization [12, 17, 18, 21].

D. Ground Truth Selection

We use a standard approach for selecting the ground truth code against each bug report (i.e., query). We go through all the commit messages of a software repository and identify the bug-fixing commits using appropriate regular expressions (e.g., `(B|b)ug\s+\d+|\=\d+| <Repo>-\d+`) [7]. Then we map these commits to the bug IDs of our visual bug reports and extract all the changed source documents from each relevant commit. Such changed documents are then used as the ground truth code for the corresponding bug reports (i.e., queries). The same approach has been widely adopted by the literature [23, 26], which justifies our choice.

E. Retrieval of Buggy Source Documents

Retrieval and ranking of relevant buggy documents were performed based on textual similarity between a given query and each of the documents in the corpus. We employ the Apache Lucene [5], a BM25-based, widely used search engine, to retrieve the buggy source documents according to their relevance to each query (Step 2c, Fig. 1).

F. Performance Metrics

We chose two performance metrics to evaluate the performance of an IR-based technique for bug localization: Hit@K and QE. These metrics have been widely used by the relevant literature [22, 29], which makes them appropriate for our experiment. *Hit@K* calculates the fraction of search queries (i.e., bug reports) for each of which at least one ground truth document is retrieved within the Top-K results. The higher the Hit@K value is, the better the bug localization performance is. On the other hand, *Query Effectiveness (QE)* is a performance metric that returns the rank of the first result that matches the ground truth document within the ranked list. That is, the lower the effectiveness value is, the better a query is.

III. Study Findings

A. Answering RQ1: What are the key elements found in the images attached to visual bug reports?

Key elements found in the attached images: We analyze the images attached to bug reports through manual and automated analysis and determine the presence and prevalence of key elements. We found 8 different types of elements under 3 major categories as follows.

(a) UI Components: The elements from this category contain one or more UI controls that can be acted upon. They may or may not contain any textual labels. We found one key element under this category as follows.

Textual UI. In this case, the attached images contain one or more UI components with textual content or labels.

(b) Programming Components: This category contains program-like elements, which could be either structured or semi-structured. We found five key elements under this category as follows.

Programming code. In many cases, the attached images contain a snapshot of code snippets relevant to the bugs.

Exceptions and errors. The attached images often contain error or exception messages that could be useful to understand the encountered bugs.

Git Commands. Git bash commands and their output were also found in several images attached to visual bug reports.

Python Commands. Python commands and their output were also found in several images attached to visual bug reports.

Other Commands. Several commands such as Unix commands, GPS system commands, and .NET commands were also found in several images attached to visual bug reports.

(c) Natural Language text: Natural language text were also found in many images attached to visual bug reports. They were written in English or other languages.

English text. Most of the attached images are written in the English language.

Other Language text. We also found text in the attached images that were not written in the English language.

Prevalence of key elements. Table II and Fig. 2 show the prevalence of key elements within the images attached to bug reports. From Fig. 2 (a), we see that most of the attached images contain natural language text (49.3%) followed by UI components (34.5%) and programming components (16.2%). Fig. 2 (b) further breaks down these statistics for eight key elements. We see that most of the attached images contain English text (43.7%) followed by textual UI (34.6%). However, we also notice the prevalence of programming code (7.0%), errors,

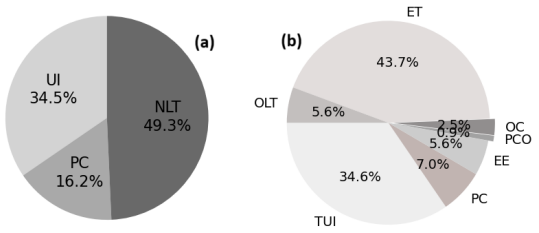

Fig. 2. Prevalence of key elements in the attached images.

TABLE I
IMPACT OF KEY ELEMENTS ON SEARCH QUERY USING QE

Key Elements	Improved	Worsened	Net Improved
Textual UI	184 (31.62%)	177 (30.41%)	1.21%
Programming Code	50 (32.46%)	33 (21.43%)	11.03%
Exceptions and Errors	41 (38.32%)	33 (30.84%)	7.48%
Git Commands	1 (25%)	2 (50%)	-25%
Python Commands	7 (50.00%)	5 (37.71%)	14.29%
Other Commands	23 (38.98%)	15 (25.42%)	13.56%
Enlgish Text	245 (35.40%)	205 (29.62%)	5.78%
Other Language Text	23 (22.77%)	27 (26.73%)	-3.96%

TABLE II
KEY ELEMENTS IN THE ATTACHED IMAGES

Category	Key Item	#Images
UI Components	Textual UI	1032
Programming Components	Programming code	209
	Exceptions and errors	167
	Git commands	4
	Python commands	27
	Other commands	76
Natural Language text	English text	1305
	Other Language text	168

and exceptions (5.6%), which can complement the textual information from bug reports. Python and other commands were found in ≈3% of the attached images. It should be noted that the same attached image contains key elements from multiple categories, indicating the diverse nature of the attached images to visual bug reports.

> **Summary of RQ1:** Visual bug reports contain 8 key elements from 3 major categories - UI components, programming components, and natural language text. However, **16.2%** of the attached images contain program-like elements (e.g., code snippets, errors, exceptions, stack traces), which could complement the text from bug reports.

B. Answering RQ2: Does incorporating multimedia information (e.g., attached images) into Information Retrieval methods improve their performance in bug localization?

We select 773 visual bug reports from 175 projects based on their ground truth availability at GitHub. Two types of queries – (a) baseline query (T + D), and (b) extended query (T + D + I) – were captured from each bug report and were executed against the IR-based method [5] to collect the results.

From Table III, we see that the extended query (T+D+I) performs better than the baseline query (T+D) consistently across

TABLE III
IMPACT OF ATTACHED IMAGES ON BUG LOCALIZATION

Group	Hit@1	Hit@5	Hit@10
T+D	13.87%	37.35%	47.67%
T+D+I	16.89%	38.78%	49.59%

T = Title, **D** = Description, and **I** = Text from Image

TABLE IV
IMPACT OF ATTACHED IMAGES ON SEARCH QUERIES

QE	Improved	Preserved	Worsened	Net Improved
UIC	184 (31.62%)	221 (37.97%)	177(30.41%)	1.21%
PC	122 (41.64%)	83 (28.32%)	88 (30.03%)	11.61%
All	250 (34.06%)	269 (36.65)%	215 (29.29)%	4.77%

UIC = UI Components, **PC** = Programming Components

three Hit@K measures. In particular, it achieves 21.77%, 3.28%, and 4.03% higher performance in Hit@1, Hit@5 and Hit@10 respectively, which are promising. Table I shows how the key elements from the attached images improve the search queries during bug localization. We see that the *Programming Code* and *Python Commands* significantly improve the baseline queries, delivering 11%–14% net improvement in queries. On the other hand, textual UI or English text improve the queries marginally. A few components, especially non-English text introduce noise and lead to worse queries than the baseline. From Table IV, we see that 122 of the extended queries perform better than the baseline queries when all key elements from *Programming Components* are considered. Besides, 36.65% of the extended queries retain their performance when all 8 key elements are used. Thus, the information extracted from the attached images has the potential to improve the search queries and thus can benefit existing IR-based methods for bug localization.

> **Summary of RQ2:** Multimedia information from visual bug reports has the potential to support existing methods for bug localization. Key elements from the attached images (e.g., Programming Components) can complement the text-based baseline queries, delivering **11%–14%** net improvement in query effectiveness.

IV. CONCLUSION AND FUTURE WORK

A few studies attempted to understand the characteristics of the images attached to bug reports, but they did not perform any fine-grained analysis. In this paper, we conduct an empirical study with 1,000 bug reports and 1,469 attached images and answer two important research questions. Our findings suggest that the attached images contain eight key elements under three major categories. Most elements contain natural language text or UI, but 16% contain programming components. We also found that the extracted information from the attached images can improve the chance of localizing software bugs when applied to an existing technique. Thus, our preliminary findings have significant implications for tackling the challenge of software bug localization. Future work can exhaustively investigate the roles and benefits of multimedia information (e.g., attached images) in bug localization.

REFERENCES

[1] Stop Words List. https://code.google.com/p/stop-words/.

[2] OpenAI: GPT-4o, A multimodal large language model created by OpenAI. https://platform.openai.com/docs/guides/vision.

[3] Cohen's kappa, A Statistical Measure. https://datatab.net/tutorial/cohens-kappa.

[4] Replication package, Bug Localization Leveraging Texts and Multimedia from Bug Reports. https://bit.ly/3ZiPnNz.

[5] Lucene, Java Library. https://lucene.apache.org/core/.

[6] Github blog, Video uploads now available across GitHub. https://github.blog/2021-05-13-video-uploads-available-github/.

[7] Adrian Bachmann and Abraham Bernstein. Software —Process Data Quality and Characteristics: A Historical View on Open and Closed Source Projects. In *Proc. IWPSE and Evol Workshops*, page 119–128, 2009.

[8] Nathan Cooper, Carlos Bernal-Cárdenas, Oscar Chaparro, Kevin Moran, and Denys Poshyvanyk. It Takes Two to Tango: Combining Visual and Textual Information for Detecting Duplicate Video-Based Bug Reports. *CoRR*, abs/2101.09194, 2021.

[9] Juha Eskonen, Julen Kahles, and Joel Reijonen. Automating GUI Testing with Image-Based Deep Reinforcement Learning. In *Proc. ACSOS*, pages 160–167, 2020.

[10] Sidong Feng and Chunyang Chen. Gifdroid: Automated Replay of Visual Bug Reports for android apps. In *Proc. ICSE*, page 1045–1057, 2022.

[11] Jayati Gulati and Muhammad Abulaish. A Novel Snowball-Chain Approach for Detecting Community Structures in Social Graphs. In *Proc. SSCI*, pages 2462–2469, 2019.

[12] Sonia Haiduc, Gabriele Bavota, Andrian Marcus, Rocco Oliveto, Andrea De Lucia, and Tim Menzies. Automatic Query Reformulations for Text Retrieval in Software Engineering. In *Proc. ICSE*, pages 842–851, 2013.

[13] Yanran Kou, Hohyeon Jeong, and Eunseok Lee. Image-based Bug Oracle Automation for Bug Report Reproduction Using Wt Detection. In *Proc. SEAI*, pages 43–47, 2021.

[14] Hiroki Kuramoto, Masanari Kondo, Yutaro Kashiwa, Yuta Ishimoto, Kaze Shindo, Yasutaka Kamei, and Naoyasu Ubayashi. Do Visual Issue Reports Help Developers Fix Bugs?: - A Preliminary Study of Using Videos and Images to Report Issues on Github -. In *Proc. ICPC*, pages 511–515, 2022.

[15] Junayed Mahmud, Nadeeshan De Silva, Safwat Ali Khan, Seyed Hooman Mostafavi, S M Hasan Mansur, Oscar Chaparro, Andrian (Andi) Marcus, and Kevin Moran. On using gui interaction data to improve text retrieval-based bug localization. In *Proc. ICSE*, ICSE '24, 2024.

[16] Kevin Moran, Mario Linares-Vásquez, Carlos Bernal-Cárdenas, Christopher Vendome, and Denys Poshyvanyk. Automatically Discovering, Reporting and Reproducing Android Application Crashes. In *Proc. ICS)*, pages 33–44.

[17] Laura Moreno, John Joseph Treadway, Andrian Marcus, and Wuwei Shen. On the Use of Stack Traces to Improve Text Retrieval-Based Bug Localization. In *Proc. ICSME*, pages 151–160, 2014.

[18] Laura Moreno, Gabriele Bavota, Sonia Haiduc, Massimiliano Di Penta, Rocco Oliveto, Barbara Russo, and Andrian Marcus. Query-Based Configuration of Text Retrieval Solutions for Software Engineering Tasks. In *Proc. FSE*, page 567–578, 2015.

[19] Maleknaz Nayebi. Eye of the Mind: Image Processing for Social Coding. In *Proc. ICSE : New Ideas and Emerging Results*, page 49–52, 2020.

[20] Brent D. Nichols. Augmented Bug Localization Using Past Bug Information. In *Proceedings of the 48th Annual Southeast Regional Conference*, 2010.

[21] Mohammad Masudur Rahman and Chanchal K. Roy. Strict: Information Retrieval Based Search Term Identification for Concept Location. In *Proc. SANER*, pages 79–90, 2017.

[22] Ripon Saha, Matthew Lease, Sarfraz Khurshid, and Dewayne Perry. Improving bug localization using structured information retrieval. pages 345–355, 2013.

[23] Ripon K. Saha, Julia Lawall, Sarfraz Khurshid, and Dewayne E. Perry. On the Effectiveness of Information Retrieval Based Bug Localization for C Programs. pages 161–170, 2014.

[24] Wang Shaowei and Lo David. Amalgam+: Composing rich information sources for accurate bug localization. *J. Softw. Evol. Process.*, pages 921–942.

[25] Bunyamin Sisman and Avinash C. Kak. Incorporating version histories in information retrieval based bug localization. In *Proc. MSR*, pages 50–59, 2012.

[26] Shaowei Wang and David Lo. Version History, Similar Report, and Structure: Putting Them Together for Improved Bug Localization. In *Proc. ICPC*, page 53–63, 2014.

[27] Ming Wen, Rongxin Wu, and Shing-Chi Cheung. Locus: Locating Bugs from Software Changes. In *Proc. ASE*, page 262–273, 2016.

[28] Chu-Pan Wong, Yingfei Xiong, Hongyu Zhang, Dan Hao, Lu Zhang, and Hong Mei. Boosting Bug-Report-Oriented Fault Localization with Segmentation and Stack-Trace Analysis. In *Proc. ICSME*, pages 181–190, 2014.

[29] Jian Zhou, Hongyu Zhang, and David Lo. Where should the bugs be fixed? more accurate information retrieval-based bug localization based on bug reports. In *Proc. ICSE*, pages 14–24, 2012.