Leveraging Large Language Model for Debugging Production Failures

Taufiq Daryanto
taufiqhd@vt.edu
Virginia Tech
Blacksburg, Virginia, USA

ABSTRACT

Debugging production failures in modern software is a challenging and time-consuming task for developers [4]. Prior research has shown that logs can assist in debugging [4], and Large Language Models (LLMs) with relevant context can be helpful [9]. However, there is limited research on systematically using LLMs to aid in debugging by incorporating logs. To address this gap, we introduce a system that utilizes LLMs with retrieval-augmented generation (RAG) [7] based on the log and codebase to help users identify possible causes and potential fixes for issues. In this study, we also conducted a pilot study to gather software engineers' opinions on this system. The pilot study serves as a preliminary investigation to enhance the tool for further research.

ACM Reference Format:
Taufiq Daryanto. 2024. Leveraging Large Language Model for Debugging Production Failures. In *Proceedings of Hokies in Advanced Topics in Software Engineering (Hatse 2024).* ACM, New York, NY, USA, 6 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

Debugging production failures can be challenging, especially considering the complexity of modern software [4]. Developers often spend hours and days trying to identify potential causes by examining the logs [4]. This is a significant concern because production failures can be expensive [6].

It has been shown that logs can assist developers in debugging production issues [4]. Also, related research has indicated that Large Language Model (LLM) with relevant context can be useful for debugging purposes [9]. However, there has been a lack of research that systematically evaluates how we can utilize LLM to aid developers in debugging production issues by incorporating logs as contextual information.

To address the research gap, we introduce a system that helps debug production failures using LLMs with retrieval-augmented generation (RAG) [7] based on the log and codebase to help users identify possible causes and potential fixes. In this study, we also conducted a pilot study to gather software engineers' opinions on this system. The pilot study serves as a preliminary investigation to enhance the tool for further research.

The video demo of the tool can be seen in https://youtu.be/uVpE5UtEcNI. The source code can be found in https://github.com/taufiqhusada/prod_debugger.

Hatse 2024, Blacksburg, Virginia

ACM ISBN 978-1-4503-XXXX-X/18/06
https://doi.org/XXXXXXX.XXXXXXX

2 BACKGROUND

2.1 Debugging Production Issue

There have been several studies on debugging production issues. One prior study by Chen et al. shows that logs can help debug production failures [4]. Their study developed a tool called LogMap, which leverages static analysis techniques to map each log message to its corresponding logging line in the source code, aiding developers in debugging production issues based on the log. Despite the benefits of their tool in debugging by mapping the log to the source code, developers still need to investigate the logs manually. Manual log investigation can be time-consuming.

Another study conducted by Alaboudi et al. explores how people typically engage in debugging activities and observes some challenges in debugging [3]. One challenge they mentioned in debugging is that people often have difficulty aggregating and understanding multiple pieces of information from various sources. This suggests a need for a better debugging tool that can integrate various pieces of information to aid in debugging.

Therefore, we aim to bridge these research gaps by developing a debugging tool that can automatically investigate the log and integrate various pieces of information, including the log and the codebase, to aid debugging.

2.2 Large Language Model for Debugging

Several works have used Large Language Models to assist in the debugging process. Viet et al. employed an LLM to fix buggy code by creating a pipeline that uses the LLM for code summarization, generation, and modification [5]. In more recent research, Singh et al. introduced PANDA, a system designed to debug performance issues in databases using LLM agents. It incorporates Retrieval Augmented Generation (RAG) with multi-modal data to add more context to the LLM to enhance the debugging process [9].

Inspired by the work of Singh et al. in using LLM with RAG to debug performance issues in databases, we are building a system to debug production issues by leveraging LLM with RAG using both logs and the codebase.

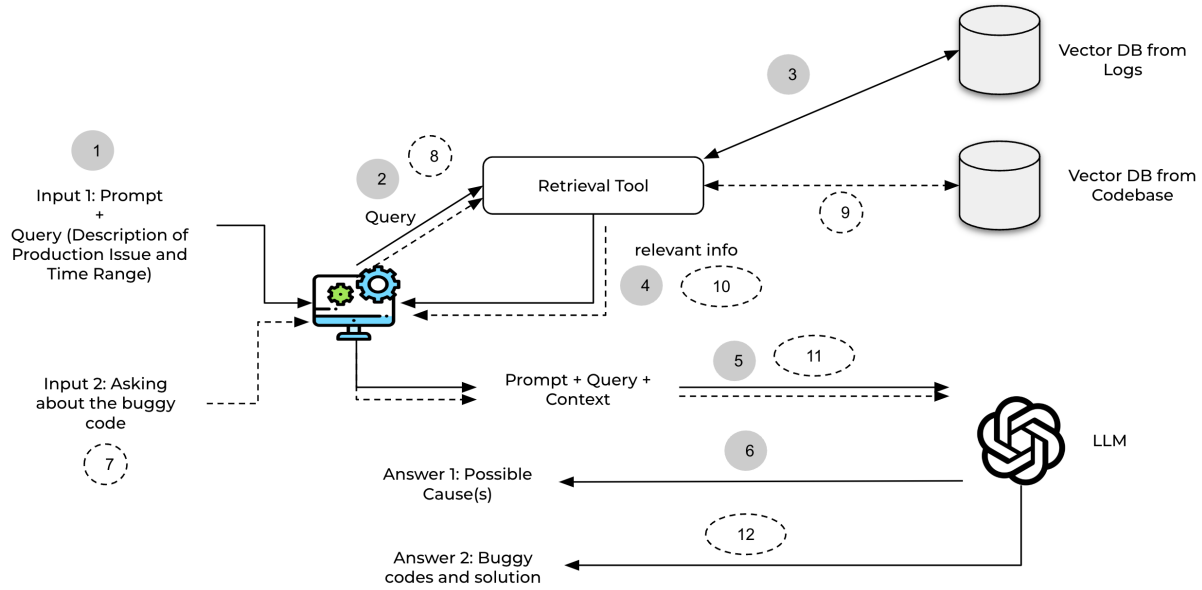

Figure 1: System Architecture

3 SYSTEM DESCRIPTION AND IMPLEMENTATION

This system leverages a large language model with retrieval-augmented generation (RAG) based on the log and codebase to help users identify possible causes and potential fixes for issues. It is implemented as a Chatbot web application that can interactively take user queries and aid in debugging. Using this system, the user does not have to go through the logs manually and a large codebase to identify the bug.

The system operates as follows (see Figure 1): First, the user inputs a query, which includes an issue description. Next, the system utilizes the Retrieval Augmented Generation (RAG) approach [7] to locate relevant logs based on the query. After obtaining the relevant logs, they are used as additional context when querying the Large Language Model (LLM) for possible causes of the production failure. Subsequently, the LLM returns potential causes based on the issue description and the relevant logs. Users can then explore these potential causes and ask about the buggy code. Following this, the tool retrieves relevant sections of code based on the relevant logs and the chosen potential cause. Once the relevant code sections are obtained, they are incorporated into the prompt posed to the LLM, which then provides explanations regarding which part of the code is likely problematic and offers guidance on resolving the issue.

3.1 Log Fetcher and Integration

This component is designed to fetch the recent logs from the log platform. For this study, we decided to use Graylog [1] as our log platform since it is open source. We integrated a sample buggy app with the log platform. Then, for this debugging system, we fetch the logs using the Graylog REST API [2] by specifying the time range and metadata we want to retrieve, including the message ID. We then save these recent logs as a JSON file. This file can be used for the next step, which is retrieval-augmented generation.

3.2 Retrieval Augmented Generation using Log

The purpose of retrieval-augmented generation (RAG) using logs is to enable users to ask the large language model about the potential cause of an issue based on the log (see Figure 2). Incorporating log querying into the LLM is important in this case to remove hallucination from the LLM. Since the entire log is too much to feed into the LLM at once, that's why we need to only grab the relevant logs using this RAG technique.

First, we convert the logs into vector embeddings using *text-embedding-ada-002* and store them in vector databases. For simplification purposes, we choose to use an in-memory vector store as the vector database, even though there is a trade-off of filling up the server memory. After we convert the logs into a vector store, we also convert the user query into vector embeddings. Then, we perform a similarity search to retrieve the top-K relevant logs based on the query. When retrieving the relevant logs, we also retrieve the metadata including the *message id*. After we retrieve relevant logs, we forward the query to the LLM together with the relevant log to get the possible causes of the issue with the relevant log.

To make the user interaction seamless, we also integrate this system with the log platform by providing the URL of the relevant

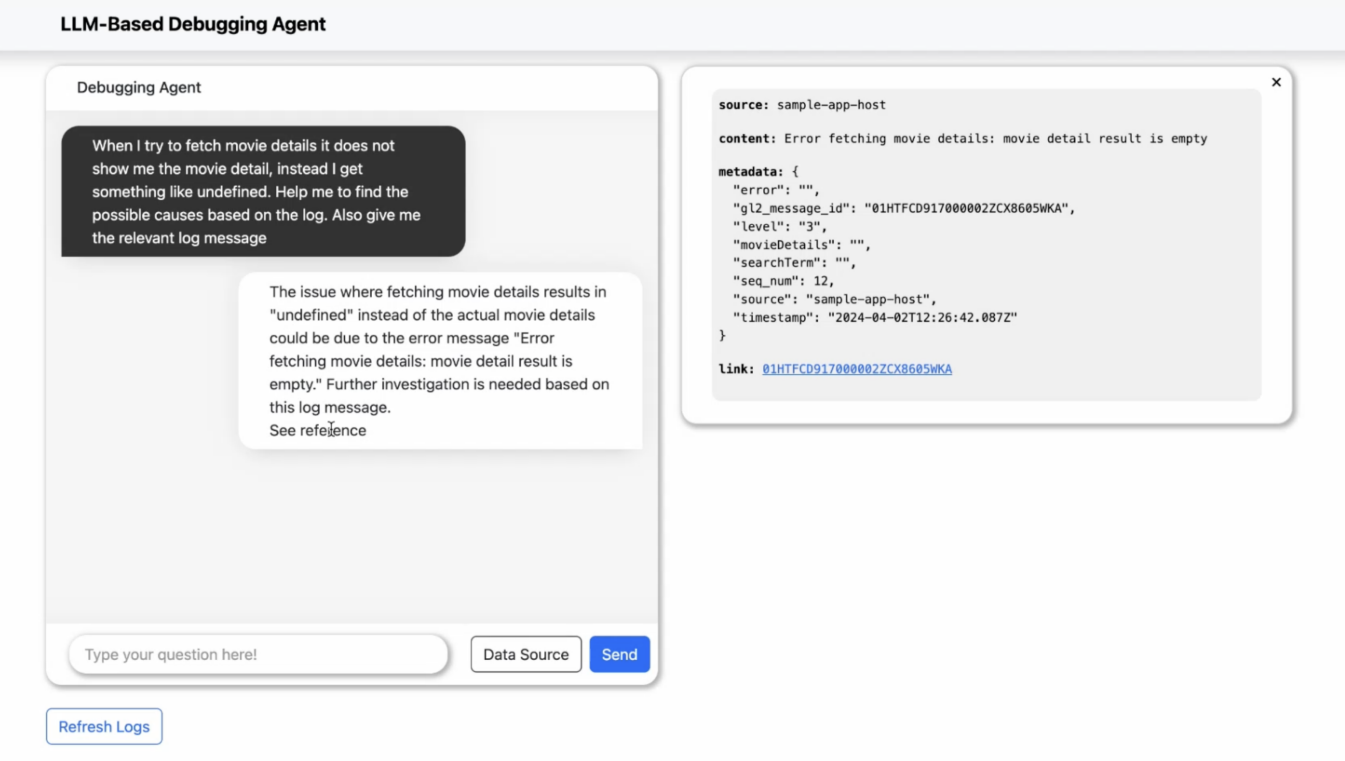

Figure 2: User Interface when User Asks about The Possible Issue based on The Log

log message. This is by adding the *message id* that we get during the RAG process as part of the URL parameter of the log platform. When the user clicks the URL, the user will be directed to the log platform, specifically showing the relevant log (see Figure 3).

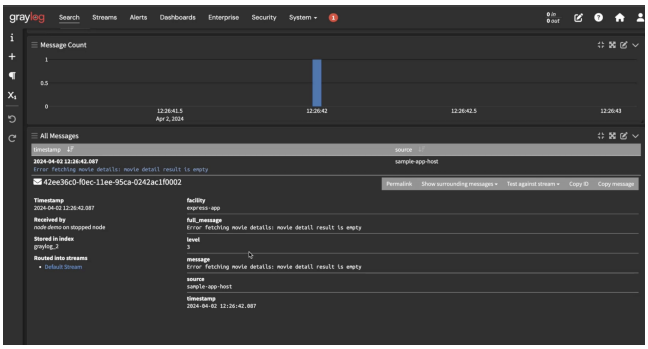

Figure 3: Integration with Log Platform

3.3 Retrieval Augmented Generation using Code Base

After the user gets the relevant possible cause from the LLM based on the log, they still need to identify which part of the code caused the issue and how to fix it. To accomplish this, the system utilizes retrieval-augmented generation using the codebase (see Figure 4). The process is similar to the previous steps, but for this stage, we use the codebase instead of the logs. We parse the code into chunks, convert it into embeddings, and store it in an in-memory vector store. Then, we run a similarity search between the user query and the code vector store. In this step, we also include the chat history as part of the query since the chat history contains the relevant logs. By including the chat history containing the relevant logs, we aim to retrieve more relevant codes. Lastly, we forward the query to the LLM to obtain the part of the code that caused the issue and the potential solution for fixing it.

3.4 UI Components and Implementation

The user interface of this debugging system is presented as a chatbot within a web-based application. This design facilitates interactive communication with the user during debugging. Furthermore, if the language model (LLM) provides an incorrect solution, the user can indicate that the solution is wrong and request to explore other possible solutions. Thus, this system essentially employs a "human-in-the-loop" approach to assist in the debugging process.

The front-end part of the application is implemented using the VueJS framework with HTML, CSS, Bootstrap, and TypeScript. The back-end is implemented in Flask with Python. We also use the LangChain library and OpenAI API for our LLM with RAG.

4 SAMPLE USE CASE

To illustrate the system's capability, we created a sample use case scenario. In this scenario, we prepared an app in NodeJS based on an

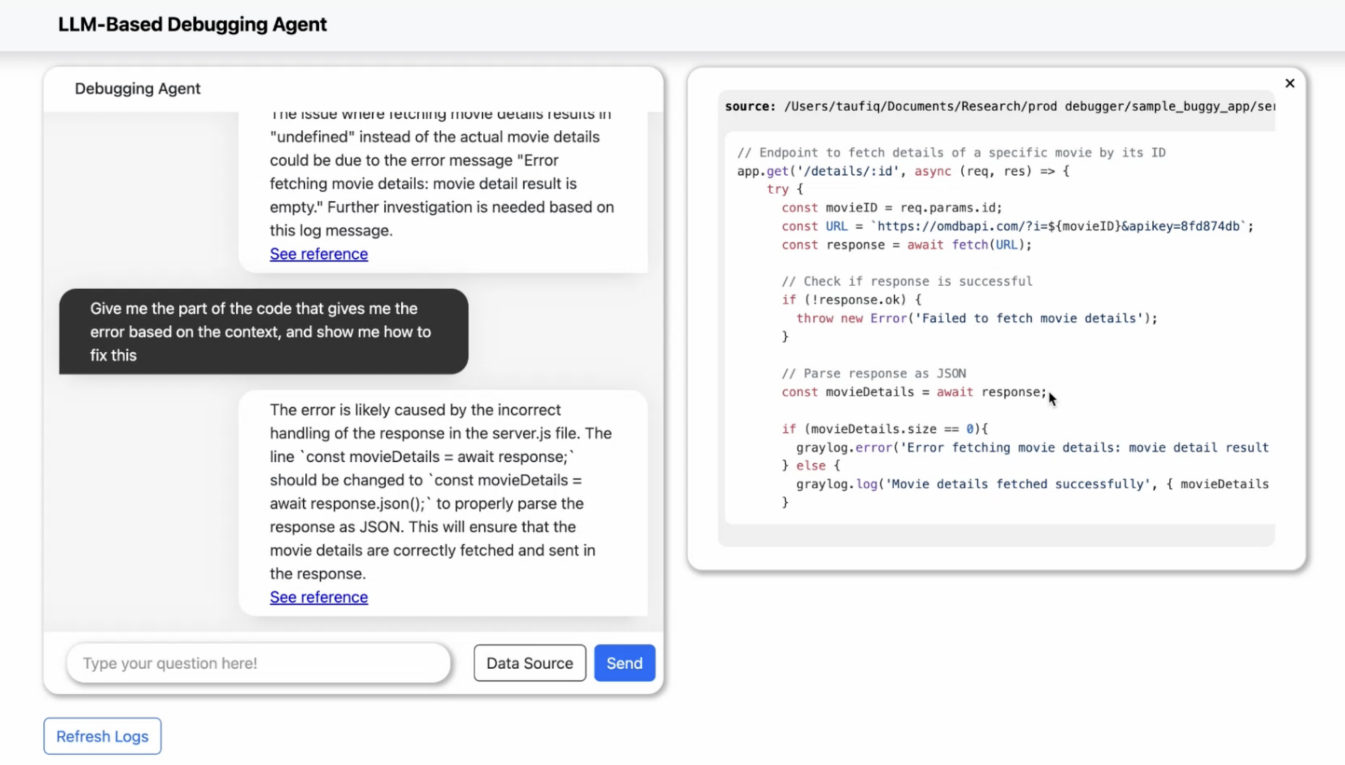

Figure 4: User Interface when User Ask about The Buggy Code

open-source web-based Movie Search application. We introduced a bug in this app by making the movie search result undefined (see Figure 5). Then, we added some logging and connected this app to the log platform.

We then use the debugging system to debug this issue. First, we type in a query by describing the issue (see Figure 2):

> *"When I try to fetch movie details it does not show me the movie detail, instead I get undefined. Help me to find the possible causes based on the log. Also give me the relevant log message"*

Then, the system outputs the relevant log message that causes the issue. After that, we type another query asking the part of the code that causes the issue and the potential solutions:

> *"give me the part of the code that gives me an error based on the context, and show me how to fix this."*

Then, as we can see in Figure 3, the system provides the part of the code that causes the issue and suggests potential solutions. In this case, the debugging system points out that the issue lies in the incorrect parsing of the response, leading to an undefined result

While we cannot conclusively determine that this system is able to effectively debug any kind of issue since we are only testing it on one use case, this simple use case illustrates that the system can identify the cause of the bug based on the log and provide the part of the code that causes the issue along with potential fixes.

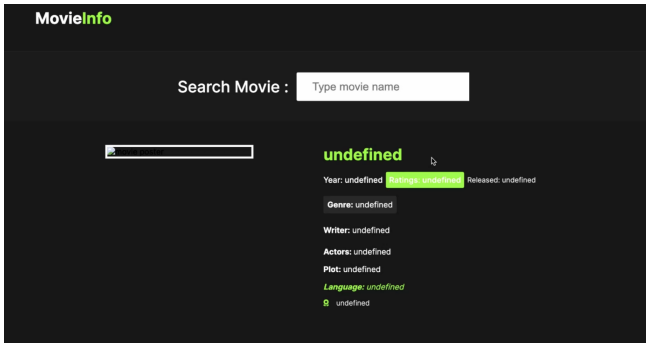

Figure 5: Sample Buggy App

5 PILOT USER STUDY

To understand people's perception on this tool, we conducted a pilot user study to gather user opinions. Specifically we are interested in exploring about:

(1) **RQ1:** What are some benefits of this LLM-based debugging system?
(2) **RQ2:** What are some challenges of this LLM-based debugging system?
(3) **RQ3:** What are some future improvements for this LLM-based debugging system?

We plan to use this pilot user study as a basic understanding to improve the tool for further research.

5.1 Procedure

We created a survey to ask people with experience in debugging software engineering issues about their opinions on this tool. The survey included a video demo demonstrating sample interactions when using this tool for debugging. Screenshots from Figures 2 to 5 were captured from this video demo. The survey questionnaire contains several questions, including their thoughts on the system, a comparison between debugging using this tool and their usual debugging process, and suggestions for future improvements to the system. Additionally, we included background questions asking about their typical challenges when debugging issues and their usual debugging methods.

5.2 Participants

In total, there are 7 respondents, including 4 Virginia Tech graduate students and 3 professional software engineers. All of them have experience as software engineering interns or full-time professionals and have experience in debugging.

ID	Profession	Years of Experience in Software Engineering
P1	CS Grad Student at VT	3.5
P2	CS Grad Student at VT	0.5
P3	CS Grad Student at VT	1
P4	Full-time Software Engineer	1.5
P5	CS Grad Student at VT	1
P6	Full-time Software Engineer	1.5
P7	Full-time Software Engineer	3

Table 1: Respondent Demographics

5.3 Data Collection and Data Analysis

We collected data from the participants' answers to the survey. Those data were related to how participants perceive the LLM-based debugging tool, some of their background challenges when debugging, and opinions about future improvements to the system.

To analyze the data, we used an open coding approach [8] with thematic analyses. The open coding process involved reading the survey responses and identifying emerging patterns. This allowed us to develop an initial set of codes that captured the main themes in the data. We then grouped related codes together into broader thematic categories through an iterative process of reviewing the codes and their relationships.

6 FINDINGS

6.1 Perceived Benefits of LLM-based debugging system

Most respondents perceive this system can help save time when debugging production issues. Participants perceive that this system can potentially identify the root causes of the issue based on the log and pinpoint the part of the code that causes the bug, along with potential solutions.

> *"I think this could be an efficient solution to find the cause for production bugs and ultimately fix them. This can save developers time and efforts." - P5*

Time efficiency is important since production failures can be expensive [6]. The time efficiency that such a system could potentially bring is also related to the fact that the system is integrated with the log platform

> *"I like how it's integrated with the log system and it helps us finding potential bugs easier and faster, developer don't have to dig the sand of logs too deep" - P7*

As P7 mentioned, since the system is integrated with the log platform, developers do not have to investigate the log manually, thus saving time and effort. Moreover, investigating the log manually is also hard for some developers.

> *"It is a bit challenging to identify the issue from the logs. In case if there is a lot of data in the logs then it becomes difficult to actually find the logs that report the error in them." - P1*

6.2 Perceived Challenges of LLM-based debugging system

6.2.1 Logs are not always reliable. Since this system identifies potential issues from the log, we also have to consider that logs are not always reliable, as mentioned by P7.

> *"[...] however not all logs are correct and not all bugs can be found by logs so it's definitely not a perfect solution" - P7*

6.2.2 Not everyone can give the right prompt. To obtain the relevant logs, users need to describe the issue specifically. The system utilizes semantic search based on embeddings to retrieve the relevant logs based on the prompt. If the user cannot accurately describe the issue using the right words, the system cannot provide the relevant logs, thus resulting in an incorrect diagnosis.

> *"I also think there should be some suggestions on how to phrase questions to the bot when you open in, in case users do not get the answers they want due to a lack of specificity." - P2*

As mentioned by P2, the user prompt could lack specificity, preventing the user from obtaining the correct solutions.

6.3 Suggestions for Future Improvement

6.3.1 Integrating with other monitoring tools can be useful. When developers debug production issues, they commonly find logs useful, but they also investigate other factors. Production issues can be caused by resource allocation-related issues, such as database latency or unexpectedly high user traffic. To investigate such issues, developers may look into monitoring platforms such as Grafana to observe metrics like the number of requests and latency.

> *"I think it would be good if the system can also detect other metrics outside of logs such as CPU of a database, certain custom metrics, etc." - P7*

As P7 mentioned, the future direction of this debugging system involves integrating with other monitoring tools beyond logs to obtain additional metrics that can potentially aid in debugging.

7 DISCUSSION

7.1 Benefit and Challenges in Using LLM to Debug Production Issue

The previous study shows developers often use logs for debugging [4]. However, manually finding the problem from a large amount of logs is difficult. Therefore, as stated in the findings section 5.3.1, using a large language model with RAG to debug production issues based on the log can potentially be useful and time-efficient.

However, several challenges exist, such as the fact that logs are not always reliable (based on finding 5.3.2), and this kind of tool heavily relies on logs for debugging. Even though we can argue that this only happens occasionally since the previous study shows that developers can debug production issues based on the log most of the time [4]. On the other hand, another challenge lies in the user's ability to give the right prompt by describing the issue specifically (based on finding 5.3.3). When the user does not provide the right prompt, the tool cannot retrieve the relevant logs, thus giving a wrong diagnosis.

Another thing we need to consider is that sometimes developers use other monitoring tools to debug production issues (Based on finding 5.3.4). Since this tool only debugs the issue based on the log and the code, issues related to resource allocation may not be able to be diagnosed by this tool. Therefore, future work can focus on integrating this tool with other monitoring platforms.

7.2 Implication and Future Work

Even though this study does not provide a comprehensive evaluation of how effective LLM can be used for debugging production issues, it can serve as a foundation and design consideration for future work in this field. This study also describes each component in detail, which helps other researchers replicate the tool and build upon it for future research.

Future work can focus more on conducting a more comprehensive evaluation of this tool using real-world production issues. This will help to understand better how effective LLM can be in debugging production issues. Additionally, besides the features implemented in this tool, future work can add more functionality, such as integrating with monitoring platforms other than logs.

8 LIMITATION

This study does not provide a comprehensive evaluation of how effective LLM can be for debugging production issues. Hence, we cannot conclude that this system is effective for debugging production issues. However, this study can serve as a foundation and provide design considerations for future work in this field.

9 CONCLUSION

In this study, we developed a web-based chatbot application that leverages LLM with RAG to aid in debugging production issues by incorporating logs and the codebase. Based on the pilot study we conducted, we identified that this type of system can potentially be useful, although there are some challenges that we need to address.

This project is related to the class theme, "Software Engineering is a human activity," as debugging is inherently a human activity. Developing a tool that utilizes Large Language Models to assist developers in debugging production issues will help the software engineering process, particularly in the maintenance phase, as outlined in the Software Development Life Cycle.

REFERENCES

[1] [n. d.]. Graylog. https://graylog.org/.

[2] [n. d.]. Graylog REST API. https://go2docs.graylog.org/5-0/setting_up_graylog/rest_api.html.

[3] Abdulaziz Alaboudi and Thomas D LaToza. 2021. An exploratory study of debugging episodes. *arXiv preprint arXiv:2105.02162* (2021).

[4] An Ran Chen. 2019. An empirical study on leveraging logs for debugging production failures. In *2019 IEEE/ACM 41st International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*. IEEE, 126–128.

[5] Tung Do Viet and Konstantin Markov. 2023. Using Large Language Models for Bug Localization and Fixing. In *2023 12th International Conference on Awareness Science and Technology (iCAST)*. IEEE, 192–197.

[6] Herb Krasner. 2021. The cost of poor software quality in the US: A 2020 report. *Proc. Consortium Inf. Softw. QualityTM (CISQTM)* (2021), 1–46.

[7] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. 2020. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in Neural Information Processing Systems* 33 (2020), 9459–9474.

[8] Johnny Saldaña. 2021. The coding manual for qualitative researchers. (2021).

[9] Vikramank Singh, Kapil Eknath Vaidya, Vinayshekhar Bannihatti Kumar, Sopan Khosla, Murali Narayanaswamy, Rashmi Gangadharaiah, and Tim Kraska. 2024. Panda: Performance debugging for databases using LLM agents. (2024).