A Large Language Model-Based Framework for Enhancing Integrated Regression

Jin Choi[1], Sangwoo Noh[2], Seonghee Yim[3], Seonil Brian Choi[4]
([1]jjin.choi; [2]sangwoo.noh; [3]sh.yim; [4]seonilb.choi@samsung.com)

***Abstract* - As the complexity of modern digital designs continues to grow, verification process becomes increasingly resource-intensive. Among the phases of verification process, integrated regression is particularly time-consuming and requires extensive manual effort. In this paper, we identify three major challenges in the integrated regression: test case prioritization optimization, log volume overload and data availability. To address these challenges, we propose leveraging Large Language Model (LLM) to advance the robustness and efficiency of the verification process.**

I. INTRODUCTION

As modern digital designs grow increasingly complex, integrated regression has become a critical phase in the verification process, ensuring system-wide functionality. While integrated regression involves running numerous test cases with limited resources, it presents challenges in managing failures, analyzing results, and prioritizing critical test cases. However, the priority of test cases can change depending on the verification context, making it challenging for verification engineers to manually adjust priorities. Furthermore, analyzing results requires engineers to manually collect error messages from logs and classify them which is time-consuming, especially as the volume of logs increases with system complexity. These challenges can be categorized into three major obstacles: test case prioritization optimization, log volume overload, and data availability.

The priority of test cases is not static and change throughout the verification process. In the early stages, test cases with significant RTL changes will take precedence, as design updates often introduce critical bugs that need immediate attention. While in the later stages, unresolved failures become more critical to ensure reliability. Traditional methodologies rely on heuristics, which can be time-consuming and suboptimal for managing these dynamic priorities.

A significant challenge in integrated regression is dealing with the massive volume of simulation logs. These logs are indispensable for error analysis, but the volume of data can overwhelm engineers, making it difficult to analyze and extract meaningful insights. Additionally, duplicated errors across logs further complicate the process of identifying and resolving issues efficiently.

The complexity of data and documentations in the verification process, along with the complicated environment, presents another challenge. The results and information are often scattered across multiple systems and platforms, making it difficult for engineers to extract and consolidate relevant data efficiently.

To address these challenges, we propose a Large Language Model (LLM)-based framework to streamline and optimize the verification process with dynamic weights adjustment for test case prioritization, log summarization and clustering to manage log volume overload, and Retrieval-Augmented Generation (RAG)-enhanced Agent to improve data availability.

II. RELATED WORKS

Recent studies have explored innovative approaches to improve verification efficiency. For instance, reference [1] introduces a machine learning (ML)-based technique that optimizes simulation regressions by identifying correlations between randomization points and coverage from previous regressions, along with a ranking approach that selects high-coverage seeds. The reference [2] reviews test case prioritization approaches in regression, providing valuable insights into various techniques and their effectiveness in software domain.

In parallel, LLMs have introduced state-of-the-art technologies and are leveraged across a wide range of industries. Reference [3] proposes a tool-augmented LLM methodology to enhance accuracy and maintain up-to-date knowledge. RAG addresses challenges of LLM such as hallucination and outdated information by integrating data from external resources, thereby enriching the knowledge of LLMs. Reference [4] provides in-depth examination of the progression of RAG paradigms, offering a comprehensive understanding of the advancements in RAG. Additionally, reference [5] is cited for its contribution to efficient memory management in LLM serving through the introduction of paged attention, which optimizes memory usage while enhancing model performance.

Building on these advancements, we have a compelling opportunity to explore the application of LLM and RAG within the context of verification regression. By leveraging capabilities of LLM to process complex data and integrate real-time information, we aim to propose a robust framework that enhances verification regression efficiency. This integration represents a novel approach in the ongoing effort to tackle the challenges of verification throughput.

III. PROPOSED METHODS

To streamline and optimize the verification workflow for engineers, we propose a novel framework utilizing LLM to enhance the verification process in three key aspects. First, it dynamically prioritizes test cases based on the verification stage. Second, it addresses the challenges of managing overwhelming logs through summarization and clustering. Finally, it integrates a RAG-enhanced Agent to improve data availability.

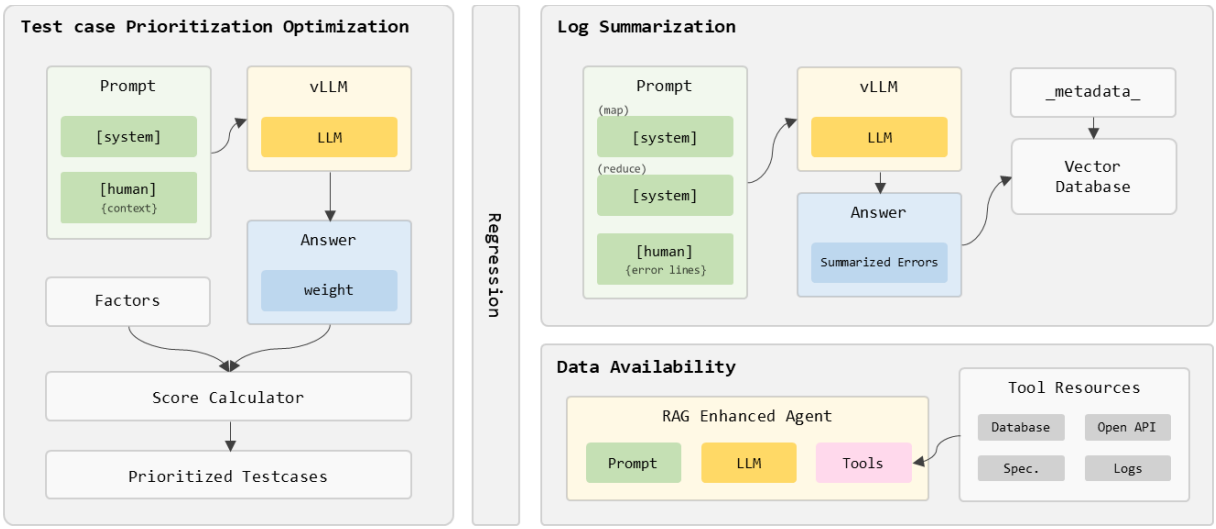

Figure 1. Architecture overview

The proposed framework consists of several components, as illustrated in Fig. 1. Determining the execution priority of tens of thousands of test cases in real-time is challenging due to limited data and the complexity of manual assessment. We optimize testcase prioritization by leveraging LLM alongside regression-related factors. LLM assigns weights of these factors that may vary in importance depending on the contextual information provided in the prompts. Based on these weights, the framework reorders the test cases to ensure that critical test cases are executed first, aligned with the verification context. A detailed explanation is described in Section III-A.

The log summarization process addresses the challenge of efficiently managing large volumes of logs. By using error messages from logs as inputs, LLM generates summarized log data, which are then embedded and stored in a vector database along with relevant metadata. This allows the framework to cluster similar errors for more efficient analysis and group test cases that encounter the same errors. By automating the summarization and categorization of logs, the proposed framework significantly reduces the manual effort required for error analysis. A detailed explanation is provided in Section III-B.

Analyzing regression results often leads to delays in the verification turnaround time. The proposed framework enhances data availability, improving access to a variety of design and verification data resources. To achieve this, the framework utilizes a RAG-enhanced Agent composed of tools bound to LLM to handle the requests of verification engineers. This enables engineers to consolidate fragmented data and streamline manual processes, ultimately improving the overall efficiency of the verification process. A detailed explanation is described in Section III-C.

A. Test Case Prioritization Optimization

The prioritization of test cases is not static; it evolves as RTL design matures. In the early stages of verification, frequent changes to RTL design often introduce critical design bugs. Consequently, rapidly identifying and addressing these bugs becomes essential. As RTL design reaches a certain level of maturity, it becomes crucial to identify verification errors that arise during the execution of test cases. In the later stages of verification, test cases that consistently fail require prioritized attention. However, manually determining the priority of tens of thousands of test cases in these evolving situations is extremely challenging for verification engineers. Furthermore, due to input token limitations, providing LLM with the information of tens of thousands of test cases simultaneously is impractical.

To address this challenge, we propose a novel method that dynamically determines the priority of test cases by leveraging LLM based on a score metric. We identify three primary factors that influence test cases priority in regression: *Change Impact*, *Execution Cost* and *Regression History*. The significance of these factors varies depending on the context of design and verification situation.

Change Impact (CI) reflects the extent of modifications made to RTL and its associated verification issues. It is calculated as the product of RTL changes and the updated issues count. Test cases associated with higher CI values are prioritized, as they are more likely to be impacted by recent design changes.

$$Change\ Impact = \Delta RTL \times Updated\ issues\ count. \quad (1)$$

Execution Cost (EC) estimates time required to run a given test case based on historical data. This factor helps balance the trade-off between importance and execution time of test case. Test cases with a shorter expected runtime might be prioritized while longer test cases are scheduled when sufficient resources are available.

$$Execution\ Cost = 1 / t(Expected\ run\ time\ based\ on\ historical\ data). \quad (2)$$

Regression History (RH) considers the number of failed runs and time remaining in a failed state. Additionally, it applies a discount rate based on the latest status of test cases. This factor highlights test cases with higher likelihood of failure and have historically been problematic, making them prime candidates for early execution.

$$Regression\ History = \frac{N_{fail}}{N_{total}} \times t_{failed} \times Latest\ Status\ Discount\ Rate = \begin{cases} 0.5\ if\ passed \\ 1\ if\ failed \end{cases} \quad (3)$$

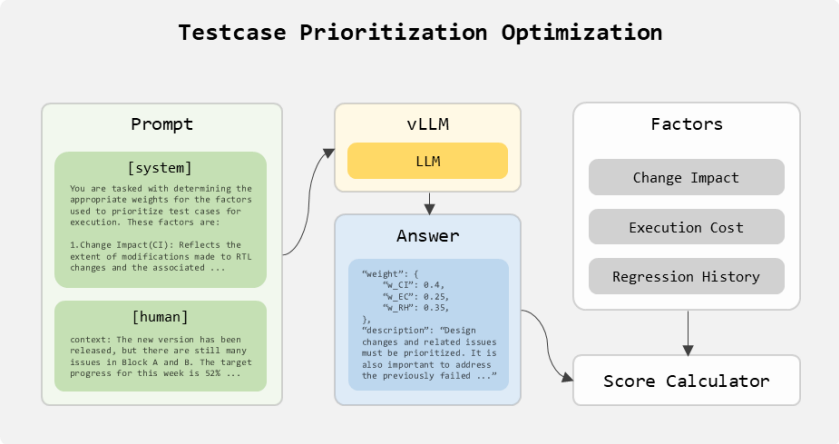

Figure 2. Test case prioritization optimization

Fig. 2 illustrates that LLM is utilized to determine the importance of these factors, which are represented as the weights of the score metric. The system prompt consists of the definition of factors and contextual information from email, release notes, or issue reports is provided as a human prompt to LLM. Whereby LLM adjusts the relative importance of factors based on given contextual information to adaptively optimize the execution order of test cases based on up-to-date information. The output of LLM is formatted in a structured manner to facilitate computation, and the weights are applied to compute a final prioritization score *S*.

$$Score = \omega_{CI} \times Change\ Impact + \omega_{EC} \times Execution\ Cost + \omega_{RH} \times Regression\ History. \quad (4)$$

This score is used to reorder the test cases, enabling an optimized test case prioritization tailored to the context of design and verification situation.

B. Log Summarization

The current process of log analysis presents several challenges. The logs are often large in size, making it difficult for verification engineers to identify relevant information efficiently. Additionally, identical errors are frequently repeated across multiple test cases, further complicating the task of pinpointing the root cause. These redundancies not only increase the volume of data that need to be analyzed but also reduce the overall effectiveness of error detection. To address these challenges, we propose leveraging LLM for log summarization and storing the summarized data in a vector database, as illustrated in Fig. 3.

The summarization process can be approached using two main methods: Map-Reduce and Refined approaches. The Map-Reduce approach involves breaking down errors into smaller chunks and processing each chunk in parallel to generate individual summaries (Map). Once these summaries are created, if the size of total tokens exceeds a maximum token limit, the summaries are repeatedly collapsed until a final summary is generated (Reduce). This final summary integrates the enormous and unmanageable errors into concise and well-organized format. This method is highly efficient for handling large volume of data, as it enables distributed processing for scalability and efficiency, significantly reducing the time required for processing. However, the drawback of this approach is that summaries generated for individual chunks may lack consistency or miss contextual information.

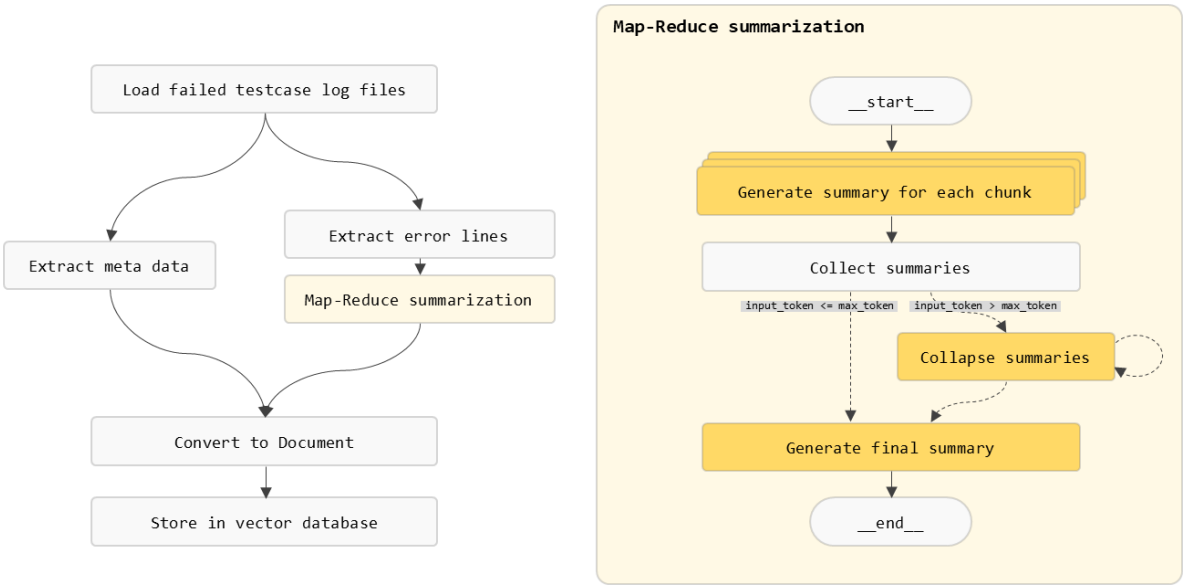

Figure 3. Log summarization

On the other hand, the Refined approach processes the entire data as a single unit. It iteratively summarizes the data by processing the first chunk and combining the summarized content with the subsequent chunks to refine the summary. This approach ensures that context across the entire data is considered, resulting in more precise and coherent summaries. However, it can be slower when dealing with enormous logs, as the entire data must be processed sequentially, which limits scalability.

Among these two approaches for summarization, we opted for the Map-Reduce approach. While Map-Reduce has a potential drawback of lower consistency compared to Refined approach, this issue can be mitigated by adjusting the chunk and overlapping size. This adjustment not only maintains consistency but also allows us to take advantage of the performance benefits of Map-Reduce. These advantages make it possible to deliver critical error information to verification engineers in a timely manner, ensuring that they can respond promptly and effectively to failures during the verification process.

C. Data Availability with Retrieval Augmented Generation (RAG)-enhanced Agent

As verification processes proceed, the system accumulates various types of data, documentation, and issues. These data serve as critical metadata to verification engineers, enhancing both the accuracy and efficiency of the verification process. The engineers utilize this data to carry out repetitive and standardized manual tasks, such as issue bring-ups, bug tracking, test environment setup, and rerunning failed test cases.

However, as design complexity increases, the exponential growth in volume and variety of data presents significant challenges in accessing, retrieving, and effectively utilizing. This inefficiency not only complicates the verification process but also directly contributes to longer verification turnaround times, as verification engineers spend more time on data retrieval and manual tasks rather than focusing on debugging and analyze the result.

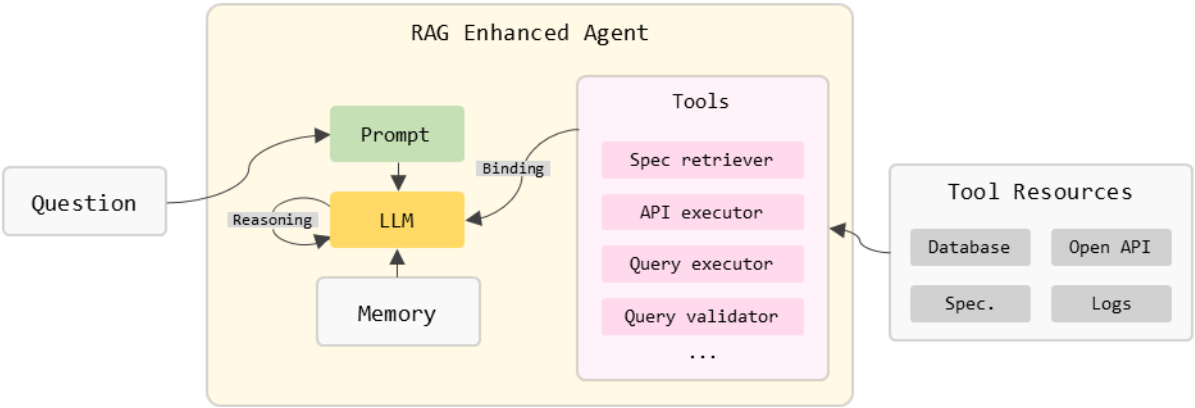

Figure 4. RAG-enhanced Agent

Fig. 4 illustrates the proposed method leveraging a RAG-enhanced Agent to address this challenge. RAG overcomes the limitations of LLM, such as hallucination, outdated information, and untraceable reasoning, by integrating domain-specific data resources that provide up-to-date, relevant information. To further optimize RAG, Agent is integrated into the framework, which is bound to several tools that streamline the verification workflow. The Agent intelligently reasons and selects the appropriate tools including retrievers searching from design specifications and documentations, executors that perform tasks according to the prompts, and validators to verify the accuracy of results. These tools are designed to seamlessly integrate into the Agent, ensuring support for a diverse workflow, including retrieving regression results, troubleshooting, and navigating databases. This flexibility, combined with its reasoning capability, makes the Agent a powerful component for automating repetitive tasks and streamlining the verification process.

IV. RESULTS

The framework we proposed is designed with a focus on delivering prompt responses to verification engineers to enhance their verification processes, while ensuring high efficiency and high-quality results. To achieve this, we opted for Mistral-7B model without fine-tuning due to its robust processing capabilities and utilized vLLM for serving and inference. This approach significantly enhanced performance and reduced memory usage, making the framework well-suited for a real-time application.

A. Test Case Prioritization Optimization

To analyze the contextual information related to design and verification, relevant documents and materials were reviewed. These documents and materials provided updates and details about ongoing issues. For instance, test cases with specific features that were experiencing issues, or requests to add new features to certain blocks were included. Additionally, some materials emphasized the situations where constraints on disk space and licenses required prioritization of certain test cases. Design engineers also provided details on which blocks had been updated and which new features had been added on the test cases.

We analyzed two scenarios; In the first scenario, we assumed that during the early stages of verification, prioritizing test cases related to changes first would help design and verification engineers identify design bugs more efficiently. The framework provided contextual information from the documents and materials through a human prompt to LLM.

Given the context, it seems that the user is prioritizing test cases that have a high potential for failure due to changes in RTL and associated issues. In this case, it would be appropriate to assign higher weights to Change Impact (CI) and Regression History (RH) factors, as they directly address the user's goal.

Execution Cost (EC) should also be considered, but to a lesser extent, as the user is willing to sacrifice some execution time for the sake of finding and addressing potential failures.

Here's a suggested weight assignment for the factors:

```python
weights = {
    'CI': 0.7,  # High weight for Change Impact as it directly addresses the user's goal
    'EC': 0.2,  # Moderate weight for Execution Cost to balance importance and execution time
    'RH': 0.1  # High weight for Regression History as it highlights test cases with a higher likelihood of failure
}
```

Explanation for the weights values:

Figure 5. LLM response of test case prioritization optimization

LLM assigned weights of three factors as follows: $\omega_{CI} = 0.7$, $\omega_{EC} = 0.2$, and $\omega_{RH} = 0.1$, as shown in Fig 5. The values for CI, EC, and RH for each test case were calculated using the equations (1) ~ (3) described in Section III-A and then normalized to derive the final score. As intended, test cases related to blocks with more extensive changes were prioritized and received higher scores. The simulation results indicated that the total time required to complete test cases related to RTL changes was 96.7 hours, representing a 79.1% reduction in time compared to the scenario where no prioritization was considered. As a result, test cases related to RTL changes were executed earlier, allowing design errors to be identified at an earlier stage. The optimization of test case prioritization enables design and verification engineers to detect errors caused by changes more promptly, thereby facilitating faster debugging. This method significantly contributes to enhancing the overall verification efficiency in this scenario.

In the second scenario, on the late stage of verification, the verification engineer's goal was to achieve a 100% pass rate as promptly as possible by executing the remaining failed test cases. The relevant context data of verification were provided as a human prompt to LLM and in this case, the weights derived from LLM for each factor were set to $\omega_{CI} = 0.1$, $\omega_{EC} = 0.3$, and $\omega_{RH} = 0.6$. Since most of the test cases had already passed, they were assigned lower priority based on the discount rate applied to the RH equation (2) in Section III-A. Additionally, due to minimal RTL changes, the CI factor had little impact on the final score. In this scenario, test cases with higher RH and EC values, considering regression history and with shorter run times, were prioritized. The simulation showed that by applying prioritization, the execution of the remaining failed test cases was completed approximately 12.3 hours faster than when no prioritization was applied.

In this case, LLM set the weights to prioritize test cases with shorter execution times in order to improve failure coverage more effectively by identifying failures sooner. Consequently, test cases with longer run times were executed later, and as a result, the overall execution time did not decrease significantly.

The verification process is inherently dynamic and subject to change across various contexts. While we focused on two extreme cases—early-stage verification with substantial RTL changes and late-stage verification with a high pass rate—the results demonstrated that prioritization of test cases helped identify errors earlier and increase failure coverage, aligning with verification engineers' intended goals. The effectiveness of prioritization was particularly evident in these extreme cases, highlighting a clear benefit in terms of time savings and improved verification efficiency.

B. Log Summarization

To analyze the result and identify errors, the manual and time-consuming process of reviewing log, which exceed hundreds of MBs in size, requires significant human effort. Furthermore, handling the large number of test cases, often reaching tens of thousands, adds to the complexity. These lead to delayed issue resolution due to the complexity and volume of logs. To streamline this process, an automated post-processing step was implemented after regression completion to summarize logs and store the summaries in a vector database. We identified that the errors occurring within the logs we used had relatively low correlation with each other, which led us to apply a Map-Reduce approach for summarization.

The summarization process, which involved parsing error messages and performing LLM inference, was executed without prior preprocessing. This decision was based on the observation that preprocessing—such as removing numbers or special characters—often resulted in data distortion. In some cases, it even led to summaries that were less comprehensible to the verification engineers. Notably, file paths and specific error contexts were retained in the summaries, as these details were crucial for engineers to access relevant files or resolve issues effectively. To further optimize the process, a cap of fifteen iterations was applied to the collapse loop. If the summary was not completed within these iterations, the process was halted to prevent excessive computation time, maintaining a balance between completeness and efficiency.

```
# SUMMARY:

1. Multiple error responses occurred during UVM_WRITE operations in the AXI::cdn_ps_amba_passive_if_peri_sve::prd component.
2. Specific errors were found at addresses 0x0203_8fb0, 0x0203_8fc0, 0x0203_8fd0, 0x0203_8fe0, and 0x0203_8ff0.
3. The data written at these addresses was incorrect, causing decoding errors [DECERR].
4. The id for these errors was 0x00, suggesting a potential issue with a specific testcase or scenario.
5. Further investigation is needed to understand the root cause of these errors and to ensure the reliability of the SoC Verification testcase log.
```

Figure 6. Summarized result

The reduce prompt used in the Map-Reduce process, where a format was specified, and LLM returned results in the predefined format as illustrated in Fig. 6. The average time and resources required for each step of this method were as follows: the log size averaged 184.04 MB, the error size was approximately 0.29 KB, and the document size for each summary was 232 bytes. The time required for summarization using LLM inference through vLLM was 5.15 seconds, and storing the summary with metadata in the vector database took 0.009 seconds. The size of data used for analysis decreased by approximately 99.87%, from 184.04 MB to 232 bytes. Moreover, the time to search test cases by errors or metadata in the vector database, which contained a total of 109 documents, was about 0.015 seconds. The retrieval time might vary depending on the number of documents in the vector database, but even so, it still represented a significant reduction compared to the manual process previously used by verification engineers. This process demonstrated high efficiency in terms of time and resource utilization, making it well-suited for handling large volumes of log data.

When searching for errors in the vector database, the human prompts were compared to stored error summaries based on similarity. It was therefore crucial to carefully formulate the prompts. This method was particularly effective when searching with contextual descriptions the context of the errors rather than relying solely on individual keywords or error terms, as it often yielded more relevant results.

C. Data Availability with Retrieval Augmented Generation (RAG)-enhanced Agent

To increase data availability and reduce the overall verification turnaround time, while allowing verification engineers to focus on their core tasks, the framework leveraged a RAG-enhanced Agent. We compared the workflow and time of the conventional versus the proposed method for two common scenarios in the verification process.

First, for searching the information about clock in the design specification document, verification engineers followed the conventional workflow, which required opening all related specification documents and manually searching for the keyword "clock". The process involved sifting through the search results, reviewing multiple entries to find the relevant information until the desired content was located.

In contrast, by using RAG-enhanced Agent, the verification engineers simply provided a detailed prompt to the agent, such as asking for information about clock in the specifications. The agent reasoned and utilized Spec Retriever tool to retrieve the top five most relevant documents from the design specification, offering answers to the user's request. This process, which took an average of just 0.03 seconds, also included clear explanations from LLM and the related documents from vector database, thereby improving the confidence of the response.

The scenario of identifying and re-running test cases that failed due to certain errors is a common situation in the verification process. In the conventional workflow, the verification engineers manually filtered the logs from previously executed test cases to identify error keywords, then re-ran the corresponding failed test cases. This process was not only time-consuming but also prone to inconsistencies and oversight, affecting overall efficiency. With the proposed framework, the agent was tasked with re-running test cases that failed due to specific error keyword. The agent reasoned to utilize Query Executor to generate a database query that was then validated by Query Validator to ensure its accuracy and correctness. Once the query was verified, it retrieved the relevant test cases and sent the result to API Executor, that was designed to re-run the test cases.

A critical aspect of this method was the accurate and precise description of the tools. Proper documentation of the tools was essential for the successful execution of the agent, ensuring that the appropriate tools were selected. Furthermore, careful consideration must be given to the formulation of human prompts within the agent's prompts, as constructing clear and explicit human prompts was vital for the agent's reasoning and tool invocation, minimizing the possibility of invalid reasoning.

V. Conclusion

By leveraging LLM, the framework optimizes test case prioritization with dynamic weight adjustments that adapt to changes in the verification context, reducing reliance on suboptimal heuristics. Moreover, the framework uses LLM for log summarization and clustering to manage the overwhelming volume of logs, making error analysis more streamlined and reducing the manual effort. Additionally, RAG-enhanced Agent is utilized to improve data availability, ensuring that engineers access relevant insights from diverse data sources efficiently.

A. Test Case Prioritization Optimization

For test case prioritization optimization, we conducted simulations of regression results considering three key factors that influence prioritization. The simulations showed that assigning test case reordering based on contextual information leads to more effective prioritization. It is important to note that additional factors beyond the three considered may influence test case prioritization. To accommodate this, the framework is designed to be extensible, allowing it to incorporate new factors and adapt to changes in the values and meanings of existing ones. Currently, the framework focused on factors at the test case level. In the future, it could be extended to incorporate factors at the block or IP level, enabling a hierarchical approach to test case prioritization.

In the proposed method, LLM determined the weights, while the factors affecting prioritization were pre-calculated. As a result, there was no mechanism to reflect verification engineers' preferences of test case prioritization. To address this limitation, adjusting the priority of certain test cases through a variable that is added to the final score calculation or metadata-driven approach could be used to allow verification engineers to specify additional prioritization factor. These approaches would provide greater flexibility and enable verification engineers to influence the prioritization process for specific test cases, offering a more customizable and adaptable solution for test case prioritization in the integrated regression.

B. Log Summarization

This paper presented an efficient approach for log summarization through an automated process that integrated a FAISS vector database. By employing a Map-Reduce approach with parallel and asynchronous processing, the time for summarization can be significantly reduced. Additionally, the use of a verification-specific fine-tuned model, rather than a general pre-trained one, is expected to produce more specialized and accurate summaries tailored to the needs of verification engineers.

While we utilized a FAISS vector database, it does not natively support metadata-based filtering. As a result, data retrieval based on metadata required manually filtering results after retrieving more data than necessary. This process often led to performance degradation as the volume of retrieved data increased, and the results could not be reliably considered metadata-filtered. To address this issue, potential solutions include embedding metadata as a string within the page context of document or storing metadata in a separate database. These enhancements would enable more efficient searches and filtering, improving the quality and efficiency of log summarization and analysis in the verification process

C. Data Availability with Retrieval Augmented Generation (RAG)-enhanced Agent

To enhance data availability and streamline repetitive manual tasks in the verification process, we leveraged RAG-enhanced Agent. In this paper, a single agent was employed to handle all tasks, which revealed a decrease in performance and reliability of responses. For the future work, by dividing responsibilities across multiple agents, the framework would allow each agent to focus on its strengths and expertise, enabling more efficient collaboration and handling of complex tasks efficiently.

Overall, the proposed LLM-based framework demonstrates the potential of LLM to reduce turnaround time and provide a flexible, efficient solution for the verification process, especially integrated regression. By addressing challenges such as test case prioritization optimization, log summarization, and data availability, it streamlines the integrated regression, enabling verification engineers to focus on their core responsibilities and ultimately enhancing the overall efficiency of the verification process.

REFERENCES

[1] Gadde, Deepak Narayan, et al. "Improving Simulation Regression Efficiency using a Machine Learning-based Method in Design Verification." arXiv preprint arXiv:2405.17481 (2024).

[2] Khatibsyarbini, Muhammad, et al. "Test case prioritization approaches in regression testing: A systematic literature review." Information and Software Technology 93 (2018): 74-93.

[3] Li, Minghao, et al. "Api-bank: A comprehensive benchmark for tool-augmented llms." arXiv preprint arXiv:2304.08244 (2023).

[4] Gao, Yunfan, et al. "Retrieval-augmented generation for large language models: A survey." arXiv preprint arXiv:2312.10997 (2023).

[5] Kwon, Woosuk, et al. "Efficient memory management for large language model serving with paged attention." Proceedings of the 29th Symposium on Operating Systems Principles. 2023.