Evaluating LLM Performance on General Program Analysis

Monoshi Kumar Roy, Lucas Ericson, Pratik Maitra, Md Anwar Hossain Zahid
Iowa State University

ABSTRACT

Program analysis is a challenging task due to several factors. The inherent variability in programs, stemming from the diverse problems they solve and the methods used, adds to this complexity. Additionally, the scale of operations these programs handle, ranging from small to large, further complicates the analysis process. This study evaluates the effectiveness of Large Language Models (LLMs) in such tasks, focusing on control flow graphs, pointer analysis, cycle dependency, and data-flow analysis. We collected data from several open-source websites, including Geeks for Geeks, Common Weakness Enumeration (CWE), and numerous other sites. We ran these data on open-source and closed-source LLMs like Wizard-Coder, GPT-3.5, Bard, Code Llama, etc. The evaluation included applying and analyzing the impact of prompt engineering techniques to refine LLM responses. Our findings identify the strengths and weaknesses of different LLMs in program analysis, highlighting the potential of prompt engineering to improve their performance and adaptation. This research contributes to a deeper understanding of LLMs' capabilities in automated program analysis, paving the way for further integration and advancements in this critical field.

KEYWORDS

Large Language Models, Program Analysis

ACM Reference Format:
Monoshi Kumar Roy, Lucas Ericson, Pratik Maitra, Md Anwar Hossain Zahid. . Evaluating LLM Performance on General Program Analysis. In *Proceedings of* . ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Program analysis, the systematic examination of computer programs for properties such as correctness, robustness, safety, and liveness, has been a cornerstone of software engineering. Traditionally, this field relied on specialized tools and formal methods designed for specific tasks. However, the landscape has undergone a seismic shift with the advent of Large Language Models (LLMs), particularly transformer-based models like WizardCoder, StarCoder, CodeLlama, Falcon, GPT 3.5/4, Claude, and Bard. This paper delves into the intersection of LLMs and program analysis, investigating these models' potential to revolutionize how we approach and execute critical software analyses. Beyond their natural language processing capabilities, LLMs exhibit a unique knack for understanding and generating code-like structures, opening up unprecedented possibilities in the realm of program understanding.

The motivation for exploring LLMs in program analysis stems from their recent successes in various natural language tasks and the tantalizing prospect of extending their capabilities to programming languages. The allure lies in their ability to process natural language descriptions of programs, potentially democratizing program analysis by making it accessible to non-expert users. Traditional program analysis tools, while powerful, often face challenges in handling the complexity and scale of modern software projects. LLMs, with their inherent scalability and adaptability, present a compelling alternative. This paper aims to explore the current state of the art in utilizing LLMs for program analysis and identify challenges and propose potential solutions for a more seamless integration of LLMs into existing analysis workflows.

In this research project, we aim to achieve several key objectives. We comprehensively evaluate various LLM chatbots, including GPT-3.5, GPT-4, Claude, and Bard, in performing essential program analysis tasks. This includes creating accurate control flow graphs, checking for cyclic dependencies, analyzing pointers, and performing dataflow analysis. We apply prompt engineering techniques, such as Zero-Shot prompting, Chain of Thought (CoT), and Self Consistency, to enhance LLM responses. The study will explore how different prompts influence LLM behavior and adaptability in the context of program analysis tasks. After applying prompt engineering techniques, re-evaluate LLMs to understand how they adapt their responses with each successive prompt. This iterative process provides insights into the learning and adaptation mechanisms of LLMs. Thus, our effort will contribute to a comprehensive understanding of the capabilities and limitations of current LLMs in the challenging domain of program analysis.

We believe, our study sets out to explore the transformational potential of LLMs in program analysis by combining knowledge from the most recent research, resolving issues, and creating creative ideas to improve LLM efficacy in this crucial area.

2 PROBLEM STATEMENT

Our goal is to evaluate and apply prompt engineering techniques on both closed-source and open-source LLMs for carrying out various program analysis tasks and rank the performance of the LLMs. Our focus is to leverage the capacity of LLMs t execute a diverse set of program analysis tasks like control flow graph construction, cyclic dependency detection, pointer analysis, and dataflow analysis. Our ultimate aim is to experiment and evaluate the performance of the LLMs in analyzing critical program analysis tasks and to have insights into their effectiveness. This study may contribute to the understanding of the capability of LLM models to reason about computer programs and can lead to enhancing the capability of LLM models in real-world software engineering scenarios.

, ,
© Association for Computing Machinery.
ACM ISBN 978-x-xxxx-xxxx-x/YY/MM.
https://doi.org/10.1145/nnnnnnn.nnnnnnn

3 APPROACHES OR ALGORITHMS

This section details the primary objectives and methodologies employed in our research to evaluate the performance of Large Language Models (LLMs) in program analysis tasks.

3.1 Approach

The study is structured around the following key objectives:

(1) **Evaluation**: Our goal is to conduct a comprehensive evaluation of various LLM chatbots, such as GPT-3.5, GPT-4, Claude, and Bard, in performing essential program analysis tasks. These tasks include creating accurate control flow graphs, checking for cyclic dependencies in code, analyzing pointers, and performing dataflow analysis. We will use a dataset of problems for each task that have been manually analyzed as a benchmark to evaluate and rank the LLMs based on their performance.
(2) **Prompt Engineering**: Prompt engineering [5] is the process of structuring text that can be interpreted and understood by a generative AI model. They can vary in specificity , context and linguistics or organization. Some of the in-vogue prompt engineering techniques include Zero-Shot prompting [3], Chain of Thought(CoT) [6] and Self Consistency [4].
We intend to apply prompt engineering techniques to these LLMs and observe the variations in their responses. The study will limit the number of prompts to four additional prompts following the initial one. This constraint focuses on the efficiency and adaptability of the LLMs to refined prompts.
(3) **Re-evaluation**: We plan to re-evaluate the LLMs after applying prompt engineering. This re-evaluation assesses how the LLMs adapt their responses to each prompt, providing insights into their learning and adaptation mechanisms.

Each of these objectives contributes to a comprehensive understanding of the capabilities and limitations of current LLMs in the context of program analysis.

3.2 Research Questions

As part of our overall evaluation, we aim to address the following research questions to gain a deeper understanding of the performance and adaptability of LLMs in program analysis tasks.

3.2.1 *Overall Performance Evaluation.*

- How effectively do LLMs perform essential program analysis tasks such as control flow graph construction, cyclic dependency detection, pointer analysis, and dataflow analysis?
- Can the performance of different LLMs be objectively ranked based on their accuracy in solving manually analyzed problems for each program analysis task?

3.2.2 *Prompt Engineering Impact.*

- How does applying prompt engineering techniques affect the responses of LLMs to program analysis tasks?

3.2.3 *Prompt Sensitivity and Adaptability.*

- To what extent are LLMs sensitive to variations in prompt structure when performing program analysis tasks?
- How do LLMs adapt their responses to program analysis prompts after introducing additional prompts through prompt engineering?

4 EXPERIMENTAL SETUP

We outline our experimental methodology for evaluating the performance of Large Language Models (LLMs) in program analysis tasks.

4.1 Creating the Dataset

4.1.1 Collecting code: We couldn't find any suitable dataset to start with for our task, so we curated a diverse set of datasets to address our intended program analysis tasks. We gather different programming examples from public platforms like GeekforGeeks for flow analysis and Cyclic Dependency analysis. We carefully collected only those programs that don't have any flow analysis tasks or cyclic information available online. Otherwise, there is always a threat that LLMs know about the program information as these models are trained on a large chunk of data. We collected samples from authoritative datasets like Common Weakness Enumeration (CWE) for pointer analysis. This meticulous selection of samples provides the opportunity to a comprehensive analysis of the LLM models and also gives us the foundation to evaluate our proposed prompt engineering techniques. We have 30 sample examples in our pointer analysis dataset and 15 codes in our second dataset which was used for other three tasks.

4.1.2 Establish Ground Truth. We made two datasets for our task, one dataset for the pointer analysis task, and one more dataset for the other three remaining tasks. The reason behind making two datasets is that, from CWE, we got the ground truth label for the pointer analysis dataset and this dataset contains C programs emphasizing pointer analysis. Hence, we wanted to introduce diversity in our dataset, and for dataflow analysis, control flow analysis and cyclic dependency, we used the same dataset but different from the pointer dataset. We had to manually label the ground truth for this dataset, so we took two approaches. One, we used the tools that we learned in our 513 classes (For example, LLVM IR representation) and we also gathered a consensus among the four of us to choose the appropriate ground truth.

4.2 Prompt Design

- **Prompt Design**: To design effective prompts for our tasks, we studied the current prompt engineering techniques widely used and experimented with by the researchers. We found that some techniques like zero-shot prompting (with little/no context), few-shot prompting(with instance response in the context), Chain-of-Thought or CoT (With thought processes behind the few-shot instances), Self-Consistency (Improved CoT with multiple thought processes). We have experimented with different prompts with varying specificity and context with zero-shot. Zero-shot is a basic and simple prompting technique. Its performance is also quite good compared to other prompting techniques [2].
- **Example Prompts**: In our prompt, we used one set of (Three different prompts) for the pointer analysis task, and for the

other three tasks, the same prompting was used, with different wordings. For pointer analysis tasks, we have used three different prompts. Here Program denotes our test codes.

(1) **System Prompt**: "You are a super smart Program Analyst".
User Prompt: "Analyze the pointers in the following code. `Program`"
(2) **System Prompt**: "You are a super smart Program Analyst".
User Prompt: "Are there any potential issues with the following pointer use? `Program`"
(3) **System Prompt**: "You are a program analyst specializing in security vulnerabilities related to memory, buffer, and pointer misuse."
User Prompt: "The following C / C++ code may or may not contain misuses such as memory, buffer, or pointer issues, and is enclosed in triple backticks. Analyze the following code `Program`"

Similarly, we have used three different prompts for the other three program analysis tasks (Cyclic Dependency, Data-Flow Analysis, and Control-Flow Analysis). However, we used prompts with the same structure with specificity differences for these three tasks. For example, we used the prompt for Cyclic Dependency and the same prompt for Data-Flow Analysis; instead of asking about Cyclic Dependency, we gave a query about Data-Flow Analysis.

(1) **System Prompt**: "You are a Program Analyst. Users will provide you with a program, and you will analyze its dataflow".
User Prompt: "Please share the dataflow information of a program by enclosing the code in triple backticks: `Program`"
(2) **System Prompt**: "You are a Program Analyst. Users will provide you with a program, and you will analyze its dataflow".
User Prompt: "Please analyze and provide details on the dataflow in a Computer program.
-Definition: Dataflow information includes understanding how data is manipulated and flows through different program parts.
Now, based on the given information, Please share the dataflow details of a program by enclosing the code in triple backticks: `Program`"
(3) **System Prompt**: "You are a Program Analyst. Users will provide you with a program, and you will analyze its dataflow."
User Prompt: "Please analyze and provide details on the dataflow in a program.
-Definition: Dataflow information includes understanding how data is manipulated and flows through different program parts.
-Example Scenario: Consider a program that sorts an array of integers. Analyze the dataflow to identify how the elements are manipulated during the sorting process.
Now, based on the given information, please share the dataflow details by enclosing the code in triple backticks: `Program`"

4.3 Evaluation

- **Sanitize Datasets**: Before feeding our test codes to the LLM models, it was crucial to sanitize the dataset. This process was done by validating and cleansing the codes to ensure the integrity of the data samples. This process ensured the LLM models got reliable data samples so that we got meaningful and accurate responses from the models.
- **Baseline Prompt Use**: For the baseline prompt, we have used our Prompt 1 template.
- **Comparison with Ground Truth**: With baseline prompts, we run the LLM models with our dataset and reported the performance based on baseline prompt.

4.4 Reevaluation with Prompt Engineering

- **Application of Techniques**: One of our research inquiries centered around the impact of prompt engineering on program analysis tasks. To address this question, we enhanced the specificity and context in our Prompt 2 and Prompt 3 templates, guiding the language model towards the intended tasks.
- **Comparative Analysis**: Following the execution of all models with three distinct prompts for four program analysis tasks, a manual analysis of the results was undertaken. Given the nature of these Language Model Models (LLMs) as text generation models, they tend to generate extensive information related to the provided prompt and code examples. The results were systematically examined to identify the most effective LLM models for program analysis tasks, focusing on overall accuracy metrics.

5 RESULTS AND DISCUSSION

Here, for all the cases, we looked into the responses of the first 10 samples of the datasets as we did it manually. The results of the Control Flow Graph (CFG), Cyclic Dependency (CD), Pointer Analysis (PA), and Dataflow Analysis (DA) are presented in Table 1, Table 2, Table 3, and Table 4. From the tables, we can see that when we improve our prompts with more specific instruction and context (Prompt 2 and Prompt 3), we are not getting better results in the case of CFG and CD. The same trend was seen for commercial models like GPT 3.5/4 or Bard. However, there is improvement in the case of PD and DFA tasks for the open-source models (WizardCoder, StarCoder, Falcon, etc). For example, in the case of pointer analysis (Table 3), StarCoder failed to generate any correct response with Prompt 1 (0% overall accuracy). But, with prompt 2 and prompt 3, we were getting significant improvement (from 0% to 50-60%). However, we are surprised to see the results of Falcon, as it is one of the largest models in terms of parameter numbers(40 billion). So, we expected much better results than other open-source models, as the second biggest model was CodeLlama with 34 billion parameters. We didn't get any correct results using the Falcon model, making it the worst-performing model in our experiment. Table 5 summarizes the results across all the models and it indicates that commercial

LLM models like (GPT, Bard, and Claude) are performing better than open-source LLMs for program analysis tasks. For our curiosity, we tried to do another analysis on GPT 4's image analysis capability for the control flow graph. We also present in Figure 1 the well-defined Control Flow Graphs(CFG), which GPT-4 constructed using DALL.E from our prompts.

Table 1: CFG Prompt Response Accuracy

LLM	Prompt1_acc	Prompt2_acc	Prompt3_acc
GPT-4	60%	60%	60%
GPT-3.5	40%	35%	40%
Bard	25%	25%	25%
Claude	50%	45%	45%
WizardCoder	35%	35%	35%
StarCoder	35%	35%	35%
LlamaCoder	45%	45%	45%
Falcon	0%	0%	0%

Table 2: Cyclic Dependency Prompt Response Accuracy

LLM	Prompt1_acc	Prompt2_acc	Prompt3_acc
GPT-4	100%	100%	100%
GPT-3.5	60%	100%	100%
Bard	60%	80%	100%
Claude	100%	100%	100%
WizardCoder	60%	40%	40%
StarCoder	40%	0%	40%
CodeLlama	60%	60%	60%
Falcon	0%	0%	0%

Table 3: Pointer Analysis Prompt Response Accuracy

LLM	Prompt1_acc	Prompt2_acc	Prompt3_acc
GPT-4	100%	90%	100%
GPT-3.5	75%	80%	95%
Bard	45%	90%	75%
Claude	90%	90%	85%
WizardCoder	0%	60%	20%
StarCoder	0%	60%	50%
CodeLlama	0%	60%	50%
Falcon	0%	0%	0%

Table 4: Dataflow Analysis Prompt Response Accuracy

LLM	Prompt1_acc	Prompt2_acc	Prompt3_acc
GPT-4	90%	100%	100%
GPT-3.5	50%	55%	70%
Bard	40%	30%	45%
Claude	85%	80%	90%
WizardCoder	90%	100%	80%
StarCoder	20%	70%	80%
CodeLlama	50%	50%	100%
Falcon	0%	0%	0%

Table 5: Ranking of the overall Accuracy of LLMs in Program Analysis Tasks

LLM	CFG	CD	Pointer	Dataflow	Overall
GPT-4	60.00%	100.00%	96.67%	96.67%	88.33%
Claude	46.67%	100.00%	88.33%	85.00%	80.00%
GPT-3.5	38.33%	86.67%	83.33%	58.33%	66.67%
Bard	25.00%	80.00%	70.00%	38.33%	53.33%
CodeLlama	45.00%	60.00%	36.67%	66.67%	52.08%
WizardCoder	35.00%	46.67%	26.67%	90.00%	49.58%
StarCoder	35.00%	26.67%	36.67%	56.67%	38.75%
Falcon	0.00%	0.00%	0.00%	0.00%	0.00%

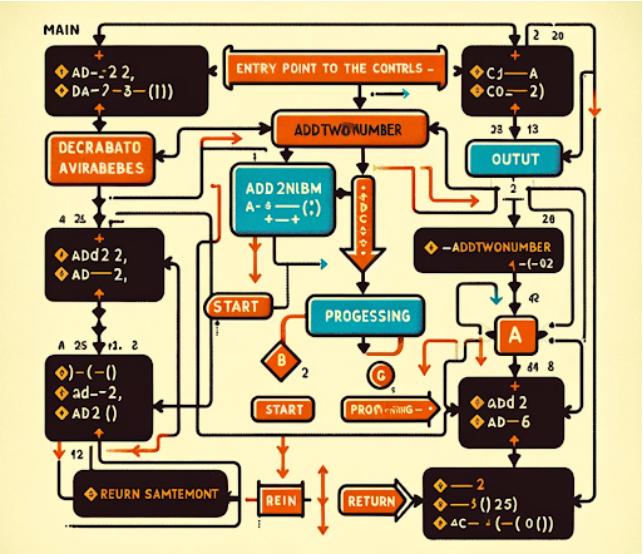

Figure 1: ChatGPT-4 CFG for adding two numbers.

6 RELATED WORK

The integration of LLMs in program analysis is a dynamic and evolving field. Recent studies, such as Chen[1] et al. (2022) exploring the understanding of Mixture of Experts in deep learning, and Kojima[2] et al. (2023) demonstrating that Large Language Models are Zero-Shot Reasoners, have set the stage for this exploration. Other works, including Wan[3] et al. (2023) on self-adaptive prompting and Wang[4] et al. (2023) on self-consistency in chain-of-thought reasoning, have illuminated various aspects of LLM behavior. Other relevant work in prompt engineering and their application to such tasks include the paper on prompt engineering techniques [5] and Chain of thought [6]. These studies collectively showcase the potential of LLMs in program analysis tasks, ranging from control flow analysis to pointer analysis. However, a

comprehensive synthesis of these findings is essential to discern overarching patterns, identify gaps in current knowledge, and lay the foundation for further advancements.

7 CONCLUSION AND FUTURE WORK

7.1 Research Insights Gained

Regarding our initial research questions, we realized that:

- Our experiments revealed that the reasoning capability of LLMs, even for simple C/C++ programs, is still far from perfection. This highlights a significant gap in their current ability to fully understand and analyze code.
- The results indicate that GPT-4 and Claude are outperforming other models in program analysis tasks. This suggests a higher level of sophistication and capability in these models.
- Prompt engineering was observed to significantly alter the responses of the models. This finding underscores the importance of how questions and tasks are presented to LLMs.
- In the case of top-performing closed-source models like GPT-4 or Claude, there was hardly any variation in performance with different prompts. This may suggest a robustness in their underlying architectures or training.
- Notably, some improvement in the performance of LLMs, especially open-source models, was observed when prompts were modified. This indicates a potential area of exploration in enhancing the effectiveness of LLMs through prompt optimization.

These insights contribute to a deeper understanding of the current capabilities and limitations of LLMs in the realm of program analysis. We can summarize and conclude the insights as:

7.2 Discussion about the results

- **Effectiveness of Prompt Engineering**: We observed that prompts incorporating additional context generally yielded better results for open source models. However, an overly specific prompt tends to lead models to "invent" unnecessary answers.
- **Comparison of Closed vs. Open Source Models**: Closed-source models consistently outperformed their open-source counterparts.
- **Top Performers**: Among all models, GPT-4 and Claude exhibited the best performance, with CodeLlama leading among open-source LLMs.
- **Possible Reasons for Superior Performance**: The remarkable performance of GPT-4 and Claude might be attributed to the implementation of a "mixture of experts" [1] paradigm. Another contributing factor could be the inclusion of our dataset in their training data, considering the popularity of sources like GeeksforGeeks among data crawlers.

We do believe this project has a potential for future research work and we would like to participate in such a research study.

8 ACKNOWLEDGEMENTS

The entire group would like to thank Professor Wei Li and Ashwin Joshy for their invaluable guidance and feedback on our research.

REFERENCES

[1] Zixiang Chen, Yihe Deng, Yue Wu, Quanquan Gu, and Yuanzhi Li. 2022. Towards Understanding Mixture of Experts in Deep Learning. arXiv:2208.02813 [cs.LG]
[2] Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. 2023. Large Language Models are Zero-Shot Reasoners. arXiv:2205.11916 [cs.CL]
[3] Xingchen Wan, Ruoxi Sun, Hanjun Dai, Sercan Arik, and Tomas Pfister. 2023. Better Zero-Shot Reasoning with Self-Adaptive Prompting. In *Findings of the Association for Computational Linguistics: ACL 2023*, Anna Rogers, Jordan Boyd-Graber, and Naoaki Okazaki (Eds.). Association for Computational Linguistics, Toronto, Canada, 3493–3514. https://doi.org/10.18653/v1/2023.findings-acl.216
[4] Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. 2023. Self-Consistency Improves Chain of Thought Reasoning in Language Models. arXiv:2203.11171 [cs.CL]
[5] Yike Wang, Shangbin Feng, Heng Wang, Weijia Shi, Vidhisha Balachandran, Tianxing He, and Yulia Tsvetkov. 2023. Resolving Knowledge Conflicts in Large Language Models. arXiv:2310.00935 [cs.CL]
[6] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le, and Denny Zhou. 2023. Chain-of-Thought Prompting Elicits Reasoning in Large Language Models. arXiv:2201.11903 [cs.CL]