ECE 322 Lab Assignment 5

Large Language Models (LLMs) for Software Testing

Overview

The objective of this lab is to explore how Large Language Models (LLMs) can be integrated into the software testing workflow, and to understand their advantages and disadvantages in this context. While there are many possible directions, this lab focuses on *black-box test case generation* and *unit test case generation*.

We will revisit the software from Lab 2 and Lab 3. This time, we will challenge LLMs to generate test cases and compare their outputs against manually constructed ones.

LLMs in a Nutshell:

An LLM is a type of neural network model that specializes in natural language related tasks [1]. It is trained to predict the next word (token) for the given context.

Although the term suggests a '*large*' model, there is no universally accepted definition of how large an LLM must be. Generally, LLMs we talk about nowadays have the following characteristics [1], [2]:

1. Are based on the Transformer architecture
2. Are trained on massive text datasets
3. Contain billions of optimized parameters

More importantly, we have observed with the scaling of the model size, these models demonstrate often exhibit *emergent abilities*, which are special capabilities that are not observed in smaller models, such as reasoning and following complex user instructions.

There is ongoing debate about whether these emergent abilities are genuine or simply the result of massive data exposure [3]. For our purposes in software testing, this distinction is less important.

However, it is crucial to remember that even the most advanced LLMs are next-token predictors. This means they may generate plausible but boilerplate or inaccurate outputs that don't fully fit the intended task. Some recognized limitations of LLMs include [1], [4], [5], [6]:

1. Hallucination: LLMs may produce inaccurate or fabricated information.
2. Maximum context window: Context window means the length of the dialog the LLM can process. With software input, it is likely that the context window is occupied, we might end up with very limited number of interactions with LLM agent.
3. LLMs are stateless: LLMs do not remember. The memory feature we see is really an external software engineering approach to store some key information about previous interactions as system prompt. The model itself does not member anything once the interaction is done.

Prompt Engineering:

The quality of prompt is critical to obtain a satisfiable result from LLMs, this has given birth to the field of prompt engineering.

In general, the more detailed instruction you provide to the LLMs, the more relevant result you will get. However, over-detailed prompts can reduce the benefit of emergent capabilities of LLMs. Therefore, our prompt needs to consider the balance between the automation of LLMs and the quality of output.

Best practices in prompt engineering emphasize on the rule-play and context, refer to Google's documentation as a good starting point (https://cloud.google.com/discover/what-is-prompt-engineering#use-cases-and-examples-of-prompt-engineering).

For example, a typical barebone prompt looks like:

> *Attached is my program, design some black box test cases for it.*

Whereas a structured prompt would be:

> *Assume you are an expert in software testing, provided is a drone program. Using EPC strategy, identify extreme points for each*

variable and construct test cases based on that.

The key to prompt engineering is to be creative and try different approaches. You can also refer to some best practices you find from other software testers using LLMs in their workflow or even interact with LLMs to let them help you design.

Introduction (No marks)

Please specify the name and version of the LLM(s) you used. If you used multiple versions or models, please list all of them.

Task 1 (30)

Revisit the drone program from lab 2 task 1. Design your prompt similar to the barebone sample prompt, ask the LLM of you choose to: (1) generate test cases.

In your report, show the prompt you used to obtain the results, and list all generated test cases in a table similar to what we have done so far. Observe the generated test cases: Does it give correct test cases? Does generate test cases following some strategies? Does it give the correct number of test cases? How they are compared to yours.

Task 2 (30)

Revisit the remote-controlled car program from lab 2 task 2. Design your prompt follows the structured prompt, ask the LLM of you choose to: (1) apply the EPC strategy and develop a set of test cases; (2) apply the weak nx1 strategy and develop a set of test cases

In your report, show the prompt you used to obtain the results, and list all generated test cases in a table similar to what we have done so far. Observe the generated test cases: Does it give correct test cases? Does it give the correct number of test cases? How they are compared to yours.

Task 3 (30)

Revisit the bisection program from lab 3 task 1. Design your prompt follows the structured prompt, ask the LLM of you choose to generate unit test cases that comply with the requirement for task 1.

In your report, show the prompt you used to obtain the results, and list all generated test cases in a table similar to what we have done so far. Observe the generated test cases: Does it give runnable test cases? Any modifications you need to make generated unit tests working? Does it generate tests that comply with our requirements? How they are compared to yours, especially if the coverage report is similar.

Discussion (10)

Comment on the effectiveness of LLMs for software testing. Describe the challenges and issues you encountered. Does three limitations we discussed in LLMs in a nutshell section applies to software testing usages? How prompts affect the output of LLMs? Any extra notes you would like to share?

Your report should include

1. Three generated test case tables. For tasks 1 and 2, you are not required to actually try out these test cases.
2. Prompts you used to get this output.
3. A discussion of your results.

References

[1] S. Minaee *et al.*, “Large Language Models: A Survey,” Mar. 23, 2025, *arXiv*: arXiv:2402.06196. doi: 10.48550/arXiv.2402.06196.

[2] A. Vaswani *et al.*, “Attention Is All You Need,” Aug. 01, 2023, *arXiv*: arXiv:1706.03762. doi: 10.48550/arXiv.1706.03762.

[3] R. Schaeffer, B. Miranda, and S. Koyejo, “Are Emergent Abilities of Large Language Models a Mirage?,” May 23, 2023, *arXiv*: arXiv:2304.15004. doi: 10.48550/arXiv.2304.15004.

[4] L. Chen and G. Varoquaux, “What is the Role of Small Models in the LLM Era: A Survey,” Apr. 15, 2025, *arXiv*: arXiv:2409.06857. doi: 10.48550/arXiv.2409.06857.

[5] A. T. Kalai, O. Nachum, S. S. Vempala, and E. Zhang, “Why Language Models Hallucinate,” Sept. 04, 2025, *arXiv*: arXiv:2509.04664. doi: 10.48550/arXiv.2509.04664.

[6] R. Massenon, I. Gambo, J. A. Khan, C. Agbonkhese, and A. Alwadain, “”My AI is Lying to Me”: User-reported LLM hallucinations in AI mobile apps reviews,” *Scientific Reports*, vol. 15, no. 1, pp. 1–15, Aug. 2025, doi: 10.1038/s41598-025-15416-8.