Project 1 D1: Reflection Document

Janelle Correia, Aadya Maurya, Jacob Philips, Aarya Rajoju, Galav Sharma
Monday, September 22nd, 2025
CSC 510

What are the pain points in using LLMs?

One of the pain points we experienced was the long wait time while the LLM generated the use cases. If the response is not what we want, we would have to wait for the generation again.

Another pain point was LLMs forgetting earlier instructions and giving wrong responses. Sometimes, the use cases were too generalized and not distinctive enough from other use cases. Especially after the first 20 use cases.

Any surprises?

It was somewhat surprising that the LLMs were able to parse through the long list of URLs we needed to feed it (although, they were unable to fully access/read each article, which was expected). We expected the LLM to stop mid-search due to the list being too long, but it spent quite a lot of time going through our resources and then eventually provided us with relevant responses. Another component of our LLM usage that we found interesting was how true-to-text the results were. We know that LLM's often hallucinate even when basing their responses off of provided data, so we were expecting to see more hallucinated results than we did, but we were able to link many of the LLMs results to the provided sources, which was a pleasant surprise.

Lastly, when we were whittling down our Use Cases to our 10 MVP cases, we expected the LLMs to inject some hidden bias into the results, like favoring Use Cases that resonate more heavily with mainstream users while neglecting minority cases. However, we found the results to be quite balanced, as they provided obvious cases that cover the most important/popular requirements of a food delivery app, but formatted them in a way that generalizes the user experience and feels universally beneficial (non-specific to a particular kind of user), as opposed to the more niche cases that benefit the more vocal, enthusiastic population of users.

What worked best?

The LLMs produced much more meaningful results when more detailed and structured information was given to the model, along with a specific, detailed prompt outlining what is needed and how it must be done. Iterating on this output led to better quality of detailed and well-structured use cases. On the other hand, vague and very broad inputs led to more generic responses, which required longer conversations to narrow down the focus and get the expected results.

What worked worst?

Having longer conversations with the LLM wasn't helpful, as it would often start forgetting important details and stray from the main focus. It would also start to give repetitive responses or contradict itself.

Additionally, giving the LLM all necessary information at once/in the first prompt wasn't particularly helpful, as the LLM would get overwhelmed and stop after a few steps in the prompt to deliver information that clearly didn't encompass the full scope of the problem (as it would become clear that several steps were overlooked). Breaking the problem down into a fixed, digestible number of steps (not too long to avoid the LLM forgetting earlier steps or making contradictions) in a cohesive manner seemed to be the most effective method of prompting to produce meaningful results.

What pre-post processing was useful for structuring the import prompts, then summarizing the output?

Some useful techniques for dealing with importing all of the use cases and summarizing them include "for-looping" the AI prompts - which essentially include taking a prompt for each use case of the 30+ and having the AI summarize each one in a sentence or two. Within these "for-looped" prompts, it's also useful to have the AI output whether it thought the use case should be included in the final list of use cases given the assignment instructions to reduce the number of use cases. You could then use that determination as to whether each one should be included or not as a preliminary eliminator for use cases if you have too many. If you don't have enough, not including the part where you ask the AI if the use case should be included or not should be skipped. Overall, this is a good way to get the AI to think about each one individually and shorten context windows.

Another technique that builds off of the last one is taking the summaries of each "for-loop" and using the AI to compare them against each other in various ranking schemes. Since the previous method only looks at each individual one, this method is useful for determining based on comparisons to each one, and will generally result in better output as compared to asking individually. This still avoids context window issues, which is a best practice in prompting.

Lastly, asking the AI to summarize its own prompts assists greatly in summarization.

Did you find any best/worst prompting strategies?

Best Prompting Strategies:

- *Be hyper-specific and structured*
  - Provide the LLMs with detailed and well-organized information from the start, explicitly outlining the required output format, the tone and the specific steps to take.
  - This method leaves very little room for ambiguity and avoids the LLM's tendency to produce vague or generalized responses, forcing it to focus on the precise details you provide.
- *Iterative Refinement*
  - Treat the first output as a first draft, not a final product, and then use subsequent prompts to give targeted feedback, asking the LLM to critique and improve its own work based on feedback.
  - This approach uses a series of short, focused interactions to steer the model toward the desired outcome, progressively layering on constraints and improving the quality with each step.
- *Task Decomposition*
  - This was one of the most influential strategies for handling complex requirements without overwhelming the model. Instead of giving one massive instruction, we broke it into a logical sequence of smaller, independent tasks.

Worst Prompting Strategies:

- *Single-Shot*
  - Giving the LLM all the information in one massive initial prompt, and expecting a perfect final result from a single prompt should be avoided.
- *Extended Conversational chains*
  - All LLMs have a context window, and engaging in long back-and-forth conversations resulted in the LLM forgetting earlier instructions, which was highly counterproductive.
  - Due to this limited context window, it would stray away from the main focus, contradict itself, and give repetitive responses, wasting a lot of time and effort.