CSC 510 Project 1 - D1

Madison Book, Cynthia Espinoza-Arredondo, Alice Guth, Griffin Pitts

What are the pain points in using LLMs?

The main pain points we encountered with the LLMs involved formatting, being repetitive, and lacking domain-specific context about the restaurant industry (though the team also lacks domain knowledge to this extent). When we did not provide sufficient formatting instructions, the LLMs produced responses that were inconsistent with the required style of the assignment. Similarly, without enough domain context about the food delivery system we were modeling, the LLM often missed functionality requirements or included irrelevant features.

Any surprises? Eg different conclusions from LLMs?

The outputs were generally stronger than we expected. Most of the time, the content of the requirements and use cases made sense and only needed minimal adjustments. The main work on our side was focused more on formatting and verifying details rather than rewriting or fixing major errors. The surprise came from how different LLMs would sometimes emphasize different aspects or leave out core functionality, such as fulfilling a customer's order. This variability meant we often had to cross-check between models, but overall the responses provided a solid foundation with relatively little content-level correction required.

What worked best?

We received the best outputs when we added additional documentation regarding the system we wanted to generate requirements. In this particular case, we used the CSC 326 web pages to use as a reference for the system we wanted to create requirements for. We also received better feedback when we included the food delivery related documentation provided in the course materials.

In addition to providing additional documentation and format instructions, we found that using multiple LLMs and combining their responses ensured that we were not missing any important functionality. For example, when generating requirements for our minimum viable system, some LLMs did not include a Use Case related to the action of fulfilling a customer's order. By using multiple LLMs, we were able to mitigate this effect and ensure that our requirements matched the format and specifications of the food delivery system.

What worked worst?

We received the worst outputs when we forgot to add additional context or formatting instructions to our prompts. Especially if we just asked the LLM to write use cases for a food delivery application, we tended to receive nonsensical responses where the Use Cases didn't make sense and were not in the correct format. Even with some additional context, we often needed multiple rounds of providing context and fixes in order to obtain output in the desired content and format. This may have been due to the fact that most of our prompts were paragraph

or written text based, and it may have been more efficient to preprocess the prompt into another format.

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

To get the best results, it was helpful to be very straightforward with the format and structure of the response we were expecting. In project a1, this looked like uploading examples of use cases in our desired format, ensuring the flows were numbered and organized like we wanted. Then, when prompting the LLMs, we simply had to note that we wanted the output to be in the format of the examples we provided. For every project following a1, we were able to upload the prior assignment submission as our formatting example, as it was already in the style we wanted. Aside from ensuring the spacing and fonts were standardized in the output from the LLMs, if we asked the LLM to give us output in a specific format in our initial prompt, little to no post-processing had to be completed on the output.

Did you find any best/worst prompting strategies?

The best prompting strategies were few-shot prompting and meta prompting. During few-shot prompting, what worked the best was explicitly defining the parameters and what was expected as the output. For example, we clearly defined what a use case was in general, defined its specific parts, and included examples of our desired output which followed the correct formatting conventions. We also clearly defined the parameters of the application we were exploring. This led to the LLM generating the best output, as we got relevant, appropriate, and (often) correctly formatted responses. Meta prompting was also useful. When we found ourselves unsure of how to structure a prompt, we asked the LLM to generate a more specific prompt to help lead us in the right direction. The LLM-generated prompt helped us achieve better output.

The worst prompting strategy was zero-shot prompting. When we provided little to no context of the application, of what use cases are, and what our desired output looked like we often received irrelevant, inappropriate, short, and badly formatted responses. These responses often either did not have much to do with our application or were seemingly random, additional features. Furthermore, this style of prompting often failed to create use cases out of core functionality, such as fulfilling a customer order, which was a big issue. When using this style of prompting, we had to specifically ask the LLM to add use cases of core features. Although few-shot prompting and meta prompting did not yield perfect output, we found them to be substantially better than zero-shot prompting.