PROJ1D1 (G24) - WolfCafe System

Sailesh Sridhar (ssridh29)
Adam Myers (ajmyers5)
Swetha Manivasagam (smaniva4)
Akash Ramanarayanan (aramana3)

What are the pain points in using LLMs?

There are few points we would like to share.
1. **Context drift:** The LLM in general, forgets the context in the long run in terms of the chat conversation, requiring me to constantly re-establish context. (Not often though!!)
2. **Hallucinations:** It would sometimes generate details that sounded correct but were factually wrong for our specific project (e.g., suggesting a feature that contradicts an established stakeholder need).
3. **Formatting:** When teammates worked on 30 use cases each and then we sat together and brainstormed, the formatting was slightly different for each of us. The prompt plays a major role, but even with the same prompt, it was different.
4. **Verbosity:** The default output gave us a lot of words which we had to cut down by manual editing of the content. But this comes down to the style of prompting as well.

Any surprises? Eg different conclusions from LLMs?

For each part, every team member asked the LLM for use cases, and then we compared the results and chose our favorites. This became a surprise as oftentimes, members would have very different results. For example, when asking the LLM what the best 10 out of the 30 use-cases were, each member in our team was given slightly different answers. One member was given a result including an allergen use-case while another was given a use case on a managerial dashboard for adding users. None of us had the same resulting “best” use-cases.
We were also very surprised that the LLM would often go off topic with Use-cases. For example, when asking the LLM to give us 30 use-cases, one team member was given 10 use-cases that were heavily focused on the process of delivery. This was especially surprising because their chat history was very similar to that of the other team members.

What worked best?

When creating the use cases, the LLMs effectively consolidated information from both online sources and our spreadsheet of reference links, which served as a knowledge base for deeper analysis. By applying a RAG (Retrieval-Augmented Generation) approach, we were able to identify gaps in our use cases and enrich them with additional details. Having access to such a large and structured knowledge base gave us a broader understanding of the overall system, while also incorporating perspectives from different stakeholders. Moreover, using multiple LLMs ensured that the outputs were not biased toward a single viewpoint, and instead provided more comprehensive, well-rounded insights. Implementation of the RAG also saved us time from going through each of the links and information provided.

What worked worst?

- When the same prompt was given twice, LLMs generated different responses each time and thus reproducibility was low.
- When the prompts are not too detailed, LLMs diverge and give irrelevant use cases.
- The use cases were far-fetched (Example: fraud prevention, marketing, etc).
- Use cases were sometimes overlapping and too similar.
- Use case responses were sometimes too detailed and sometimes very abstract. We had to prompt more questions to get the similar levels of detailing.
- Alternative flow and sub-flows were too peculiar.
- Asking LLMs to prioritise top 10 use cases from 20 other use cases was not very useful. We had to pick each of the best 10 use cases based on relevancy and intuition.

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

When prompting the LLM, the most useful way that we found to structure the prompt was to start by providing the LLM with as much information on the project as possible. Most chat histories that we made, started with providing the problem statement, project requirements, and a summary of what our expectations were for an answer. Then, when asking the question that we wanted an answer for, we would be very verbose to ensure that none of the results would be misinterpreted. Lastly, since use-case formatting was oftentimes inconsistent, we found that prompting the LLM with our preferred output was helpful since the resulting use-cases would be more consistent.

Did you find any best/worst prompting strategies?

Best Strategies:

1. **Few Shot Prompting:** Giving out examples of what we expect as an output really helps.
2. **Role Playing:** Give the LLM a role. For Example, “You are an admin” or ”Act as a customer”. These would make the LLM think like that person and give out responses.
3. **Iterative Interactions:** Ask follow up questions and give your opinion. This would improve the quality of responses for the recurring queries.
4. **Reducing Hallucinations through Knowledge-Base Restriction:** By telling the LLM to refer to only a specific set of knowledge base helped reduce hallucinations and irrelevant information for the responses generated.

Worst Strategies:

1. **Open Ended Prompts:** Giving incomplete prompts/ not providing the full context would surely affect the final response.
2. **Context Dumping:** Dumping a lot of context and expecting the LLM to give out a good response is not plausible.
3. **Assumption of Implicit knowledge:** Expecting the LLM to remember details from a previous, separate conversation or to understand project-specific acronyms and jargon without defining them. Each prompt should be as self-contained as possible to ensure accuracy.