Project 1d1

What are the pain points in using LLMs?

Several challenges came up when using LLMs for project planning and use cases, most often related to hallucinations, formatting, and depth. Hallucinations did not occur frequently thanks to careful prompting, but they remained a risk. Every output needed to be read closely to confirm that the logic of the flows held together and that no steps were included that conflicted with the requirements. Formatting also posed problems. Even when the model produced output close to the target structure, the process of copy-pasting into our documents often stripped out elements like lists or indentation, forcing us to make repeated adjustments to keep everything uniform. Depth was the third recurring issue. Some outputs were too shallow and skipped key details, while others went further than we needed, introducing items like order scheduling that were outside the scope of the current phase. Altogether, these issues meant that while the LLMs were productive, their outputs always required an additional layer of review and standardization before being integrated into the project.

Any Surprises? E.g. different conclusions from LLMs?

It was surprising to see that local models, with the right prompting, produced use cases on par with or at times better than proprietary systems like ChatGPT. A key reason for this was that the local models hallucinated less than expected, which meant our edits were more about adjusting the depth of content rather than correcting errors. Another factor was the knowledge base we set up for the local LLM. By grounding it with the requirements PDF, the model could incorporate project-specific details directly into its outputs, which made the results more accurate and relevant to our needs.

What worked best?

Idea generation with LLMs proved valuable during brainstorming, since the models surfaced options and perspectives that might have been missed otherwise. This broadened the range of possibilities considered early on. In addition, use case generation significantly boosted productivity, as editing a draft that was already well-structured required much less effort than writing new cases from the ground up. Together, these uses showed the strength of LLMs in accelerating creative and routine parts of the process.

What worked worst?

Nothing performed outright badly, but zero-shot prompting proved the least effective. With little guidance, the model still produced use cases, yet the results tended to be incomplete and uneven. Sections like preconditions or alternative flows were often

missing, and formatting varied from one case to another. This created extra effort in editing, showing that unstructured prompts led to more work downstream compared to approaches where expectations were clearly laid out.

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

We learned that pre-processing was critical for getting consistent results. Before giving prompts, we often had to set the stage by including small examples and clarifying the level of detail we wanted. This kept the LLM from drifting into irrelevant areas and helped align the depth of the output with our needs. For post-processing, the biggest step was standardization. After the model produced its text, we had to go through and make sure the formatting matched our template and that sections like Preconditions, Main Flow, Subflows, and Alternative Flows were properly labeled. Summarizing the output into a cleaned-up version made it much easier to integrate into the project and also to compare across different prompts. This pre-post process acted like guardrails that kept the LLM focused and its output usable.

Did you find any best/worst prompting strategies?

As expected, careful prompting with examples and specified structures worked well, since the model followed the patterns we set and produced use cases that needed only light editing. Giving it a sample use case or labeling the flows directly cut down on formatting issues and unnecessary detail. In contrast, the worst strategy was vague prompting like “Give me 10 use cases for a food delivery app.” Those outputs were too general, often missing sections like preconditions or alternative flows, and took more time to fix than they were worth.