Project 1d1 – Problem Reflection

CSC510: Software Engineering – Fall 2025

Zhaonan Meng, Rahmy Salman, Travis Thompson, John Stack

Let's begin with a discussion of the six reflection points shown in the assignment, and end with a preliminary design for a collaboration-based, LLM-powered RE documentation engine.

One of the largest pain points in using LLMs was the challenge of information traceability. Each LLM has its own unique way of formatting its sources from the web, with some (ahem CLAUDE) more or less refusing to cite its sources in line and hardly citing at all. The inline citations provided by ChatGPT were great, but the output verbosity was typically significantly less than the other models. Aside from verbosity, the two main models we investigated (Claude and ChatGPT) provided very similar use cases and stakeholder analysis. We were hoping to play the output of each model off of the other, so this was an unpleasant surprise for us. In an ideal scenario, many models could be made to collaborate and critique one another in the requirements engineering, but that does not seem to be feasible.

The question of "what worked best" is a deeply optimistic one: the easiest policy worked very well. We simply threw everything into the LLM context. Project spec, links from the Google sheet, PDFs from online, all went into the context. We didn't hit any major problems with this technique, but it certainly would benefit from RAG. The worst technique we found was "cross-prompting", or providing multiple models the same prompt. As we mentioned earlier, the models tended to come to the same conclusions. They were similar to the point of redundancy.

For input preprocessing, we were hoping to provide a single link to the Google sheet containing all the web documents to parse through, but the model was not able to access the Google sheet link. Instead, we had to copy and paste the links from the sheet to the model. This

is a pain point we could improve upon. For output postprocessing, we found that the model either had very ugly output formatting or inconsistent output formatting across prompts. You could either take the ugly output it gave you and make it pretty yourself, or you could try prompting it to give you prettier bullet points and lists and sections and so on. You would rather have RE specific software that formats, counts, and maybe sorts use cases and other RE elements.

LLMs have become extremely adept at following instructions. That being said, they are only going to perform the work that you direct them to do. Therefore, if we left out context or provided only a vague description of the problem that we were trying to solve, the LLM would provide an unsatisfactory answer..

Now, let's discuss collaboration-based, LLM-powered RE documentation engine. Here's the story: a business wants to start a new project and needs the RE doc. They gather representative stakeholders. Each representative is given an LLM instance as their agent ("stakeholder agent") to which they will enumerate all the requirements, desires, hopes, dreams that they have for the project. This can be in the form of natural language or as complete documents in pdf form or found on the web. Stakeholder agents need to be capable of RAG, they should come with suggestions for their particular stakeholder. Think of stakeholder agents as an advocate for their stakeholder. Then, all stakeholder agents condense their conversations with the stakeholders into relevant portions and pass those documents to a final LLM agent referred to as the orchestrator. This orchestrator takes those documents and outputs the final RE document.

At this point in the process, each stakeholder can review the final document with their agent and provide feedback. If a stakeholder or their agent expresses a need for reevaluating the final document, they are free to submit a new conversation to the orchestrator for reconsideration.