1. **What are the pain points in using LLMs?**

- The biggest pain point when using LLM's for something like requirements engineering is the context window. From our experience, Claude especially struggles to consider past conversation history when the conversations get especially long and complicated. When trying to use Claude to assist in generating use cases, it would sometimes generate use cases that contradicted the workflows of older use cases. In order to work around this, we would need to provide all of our current use cases and have a RAG-like system. Chat GPT. Prompting was also crucial in avoiding this. We would need to explicitly state in the prompt to ensure that the newly generated use cases do not contradict the older ones and each other.
- Continuity of work: While I can obtain outputs with high quality within a single continuous chat, once I resume the task sometime later, I sometimes find that some of the constraints I had previously added are no longer remembered.
- Difficulty of cross-LLMs collaboration: When using multiple LLMs to complete the same task, it is pretty hard to share the multi-turn interaction history and intermediate results across them without losing information.
- Context awareness for LLM chat systems: Daily tasks often involve multiple software applications. For example, completing a homework assignment requires information from various sources, such as textbooks on Github, a draft in Google Docs, and code from an IDE. Copying and pasting content from each application into an LLM chat system and then arranging the prompts is time-consuming and inefficient.

2. **Any surprises? Eg different conclusions from LLMs?**

- Sometimes LLMs can really open up my mind. They turn out to be a truly amazing and inspiring brainstorming tool. This is especially useful in the planning phase of development. For example, an LLM could provide useful suggestions for use cases that are outside of my current knowledge framework.
- A notable finding is the reliance on user expertise. While LLMs possess vast knowledge, when I'm unfamiliar with a concept, I struggle to articulate my intent or needs precisely. Consequently, the LLM often produces generic or vague content because it lacks the specific direction required for a tailored answer.

3. **What worked best?**

   Prompt engineering was crucial in successful usage of LLM's for requirements engineering. One explicit example is the few-shot prompting. When creating new use cases from blank, few-shot prompting helps LLM to formulate the use cases and result in answers with suitable length and granularity. On the other hand, we would provide all the older use cases as context and tell it to ensure that the new use cases worked seamlessly with the older ones. Doing so worked well every time and it helped to efficiently come up with new use cases or revise old ones. This also worked very well in skimming all existing use cases to see which can be discarded for the MVP. This made our jobs much easier considering how many use cases there were by the end of it.

4. What worked worst?

- Generating a complex table while maintaining a well-controlled format.
- While a carefully crafted prompt is provided, LLMs sometimes struggle to generate the expected answer on the first try, which makes multiple refinements necessary.

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

- Pre-processing:
    - Turning trunk of natural language into structured format, eg. json file
    - Use LLM to rewrite and structure the prompt
- Post-processing:
    - Validity and consistency checks to avoid conflicts with previously generated content.

6. Did you find any best/worst prompting strategies?

Good prompting strategies:

- Finely structured prompts
- Explicit instructions
- Guide LLMs to reason
- Use XML tags to structure different types of context
- Few-shot prompting
- Showing chain of thoughts
- Asking LLMs outputs text and tables instead of images (Due to the poor performance of Multi-modal LLMs)

Worst prompting strategies:

- Simple and implicit instructions, such as “refine the use cases”, would lead to uncontrollable results
- Lack of defined output formatting constraints
- Inconsistent examples. Should ensure all few-shot examples consistent and also align with the requirement in description
- Broad, unspecialized queries yield generic results