Project: 1d1

Project Group 6 - Section 1

Project Members:

Name	Email
Khush Ketan Patel	kpatel53@ncsu.edu
Mia Glenn	mglenn2@ncsu.edu
Richa Jha	rjha3@ncsu.edu
Ishwarya Gandamsetty	igandam@ncsu.edu

1. What are the pain points in using LLMs?

LLMs can only remember a limited amount of information at once, making it hard to keep track of all requirements when working on big projects. We often had to remind the LLM about previous work or break down large requirement sets into smaller pieces, losing some of the big picture that ‘s important for requirements work.

Unlike regular requirements tools, LLM conversations don’t automatically save different versions or track changes. It’s hard to see how a requirement changed over time or remember why we made certain choices without writing down all the prompts and responses by hand.

2. Any surprises? Eg different conclusions from LLMs?

Claude was able to stay focus on being accurate when asked about topics not covered in source material (like “car insurance use cases”). Claude correctly said it had no information rather than making up answers. This commitment to accuracy was unexpected and proved very valuable for requirements work where being precise is critical. On the other hand, ChatGPT often provided a vague detailing when it lacked information about the query.

Claude’s ability to consistently follow complex formatting instructions was remarkable. When provided with detailed use case templates, Claude followed them exactly, reliably producing outputs with proper Preconditions, Main Flow,

and Alternative Flows sections. This consistency was more reliable than expected and significantly reduced cleanup work afterward.

3. What worked best?

One of the successful approach was using few-shot prompts with carefully made templates. As documented in our prompt history, providing 2-3 complete examples of properly formatted use cases let to consistently high-quality outputs. The template approach with clear sections (Preconditions, Main Flow, Subflows, Alternative Flows) worked very well.

The “do it yourself” local LLM with RAG proved highly effective for domain-specific requirements. By providing background documents about food safety regulations, FDA codes, and business processes, we could generate requirements that were both technically accurate and appropriate for the domain.

The most productive sessions involved step-by-step improvement where we would:

1. Generate initial requirements with simple prompts
2. Improve with few-shot examples
3. Add domain context for accuracy
4. Review and improve specific sections

4. What worked worst?

Simple, basic prompts consistently produced generic, low-quality requirements that lacked domain knowledge and proper structure. These outputs required significant manual revision and often missed critical alternative flows or edge cases.

Trying to generate all 10 or more use cases in a single prompt led to repetitive, shallow requirements with decreasing quality toward the end of the output. Breaking the work into smaller, focused prompts produced much better results.

Additionally, when prompts were left too open-ended, ChatGPT sometimes filled gaps with hallucinated assumptions about the domain.

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

the output?

Pre-Processing:

- Collecting relevant domain documents and examples before prompting
- Converting those documents into relevant chunks(vectors) and storing them in vector database.
- Fetching relevant chunks based on the query and creating clear structural templates with obvious section heads.
- Defining specific formatting requirements, length limits, and quality criteria.
- Establishing consistent terminology and technical language.

Post-Processing:

- Converting LLM outputs to consistent templates across different tools.
- Checking outputs against domain requirements and business rules.
- Organizing generated requirements into clear requirements documents.
- Maintaining links between prompts, outputs, and final requirements.

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

Best prompting strategies:

- Claude’s ability to follow complex structural templates consistently exceeded expectations.
- Claude’s tendency to admit when it didn’t know something rather than guess proved surprisingly valuable for requirement work, ensuring reliability even when domain knowledge was incomplete.
- Claude showed excellent ability to take feedback and improve outputs through back-and-forth conversation, making it ideal for requirements improvement workflows.
- When provided with domain documents through RAG implementation, Claude effectively combined external knowledge with prompt instructions to produce contextually appropriate requirements.
- ChatGPT responded well to iterative refinement, producing more well defined requirements maintaining consistency with the provided external documentation.

- ChatGPT excelled at condensing long discussions into concise summaries without losing heavy context, making it easy to pass on the conversation to different windows.

Worst prompting strategies:

- Claude’s accuracy focus led to overly cautious outputs that missed creative or innovative requirement possibilities.
- In some cases, Claude produced very detailed but repetitive content when simpler, more concise requirements would have been more useful.
- Broad prompts often led to hallucinated assumptions about the domain, introducing new information which required additional validation.