CSC 510 Project 1d1

Tiva Rocco, Rujuta Palimkar, Aayushi Masurekar, Anusha Upadhyay

Group 5

What are the pain points in using LLMs?

One of the biggest challenges in using LLMs for requirements engineering is maintaining consistency of style and format. Without strict templates, the model often defaults to informal narratives or bullet points, requiring repeated prompting to achieve a professional, structured use case format. Another pain point is the issue of redundancy and overlap. LLMs sometimes generate requirements that appear distinct on the surface but, upon closer review, essentially repeat the same functionality in slightly different terms. This highlights the need for human validation or automated semantic checks. Context management also presents difficulties: when multiple documents are uploaded, the LLM may surface requirements at different levels of detail, mixing strategic goals with low-level technical specifications. Moreover, there is a persistent verification gap — while outputs are often well-structured and plausible, there is no guarantee that they accurately align with true stakeholder needs or system constraints. Finally, traceability remains weak. Although the LLM can generate detailed requirements, linking them back to source documents, regulations, or stakeholder inputs is not automatic and requires additional effort to maintain compliance and auditability.

Any surprises? Eg different conclusions from LLMs?

A surprising aspect of working with LLMs was observing their non-deterministic nature. When asked the same question in different sessions, the LLM sometimes emphasized different aspects of the same input documents — for example, focusing on nutritional policy in one response but highlighting tax compliance in another. This inconsistency demonstrated both the creative flexibility and unpredictability of the model. Another surprise was the level of innovation in the generated outputs. The LLM occasionally proposed requirements not explicitly mentioned in the documents, such as context-aware AI meal recommendations. These suggestions were often plausible and aligned with the project's goals, but they also required careful evaluation to separate genuinely valuable insights from speculative ideas. Finally, I found that LLMs are naturally stronger at amplification (expanding a short prompt into a detailed requirement) than condensation (summarizing long requirements into concise statements). Summarization often required more precise prompting to avoid oversimplification, whereas expansion flowed more naturally, sometimes even adding unexpected but relevant detail.

What worked best?

The more structured the input was the better the LLM response was going to be. Utilizing an iterative approach, asking the LLM to refine and reformat each response through structured prompts also proved to work well. We think this is because it allowed the model to amplify ideas we already framed, so the output remained relevant and coherent. Another discovered "best practice" was brainstorming with the LLM. Thinking out loud, throwing ideas out there. The models don't necessarily need perfectly structured thoughts and sentences to provide supplemental ideas or help push your thought process forward. Once working with the LLM through the brainstorming and refining stages (practicing careful prompting) it has

somewhat of a grasp on what your goals are along with your content. Knowing this, the models can be helpful when considering alternative flows, edge cases, or new ideas from a different perspective.

What worked worst?

Some strategies that didn't make the cut were zero-shot, underspecified, or generally overly open-ended prompts. This caused models to produce generic scenarios or miss subtle details and constraints regarding the WolfCafe design structure. Prompting in this way requires multiple follow-ups and refinements. Another factor to consider is that when trying to maintain consistency across iterations, details introduced in earlier iterations may be overwritten or reworded inconsistently in the next. Of course tasks requiring stricter logic like exact database schemas were more error-prone than high level design.

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

For preprocessing, the use of RAG was very effective. It helped filter and prepare relevant content from research papers and food standards before giving it to the model. This made sure the use cases were based on the right domain knowledge instead of being created in isolation. Iterative prompt refinement was also necessary. The inputs were rewritten to make them clearer, the scope was defined, and the level of detail or uniqueness was specified. Several strategies were used to structure the prompts effectively, including using bullet points, numbered lists, or templates, defining clear constraints (for example, "generate exactly 30 use cases" or "ensure no overlaps"), and framing the model as an expert (such as "Act as a software engineer specializing in food delivery systems"). Key terms like "MVP," "stakeholder bias," and "food compliance" were also defined in advance to maintain consistency. These preprocessing steps helped create clear and reliable inputs, improving the quality of outputs.

For post-processing, the focus was on improving both the accuracy and usability of the outputs. The responses were checked carefully to identify errors, hallucinated or irrelevant points, and repeated items. Overlapping use cases were merged through deduplication, and long answers were summarized to make them concise and actionable. The outputs were also reorganized and reformatted into structured formats such as lists, tables, or requirement-style statements to make them easier to use. Additionally, the use cases were prioritized based on feasibility, stakeholder value, and MVP requirements, and terminology, scope, and style were cross-checked to ensure consistency. These steps allowed the initial 30 use cases to be narrowed down into 10 unique and practical MVP use cases.

Overall, preprocessing through retrieval, structured prompt design, constraint definition, and role specification, combined with post-processing through error checking, deduplication, summarization, reformatting, prioritization, and cross-checking, greatly improved the reliability, clarity, and usefulness of the outputs. These steps ensured that the final results were accurate, practical, and aligned with the scope of the MVP.

Did you find any best/worst prompting strategies?

Best strategies

- Frame with role, goal, constraints, and output format; keep one objective per prompt and limit scope.
- Add just enough context and few-shot examples to demonstrate structure and edge cases; keep examples consistent.
- Ask for structured outputs (bullets/JSON/table) plus success criteria; require a short self-check or verification pass.
- Decompose multi-step tasks into short turns; plan first (step-back), then execute and revise.
- For factual tasks, anchor with sources or retrieval and request citations/evidence in the output.
- For complex reasoning, use sample-and-check or self-consistency, then finalize; avoid exposing long internal reasoning by default.

Worst strategies

- Vague or multi-goal prompts without constraints, format, or audience.
- Overly long context or conflicting examples that cause drift.
- Negative-only guidance (“don’t do X”) without positive targets.
- Forcing verbose step-by-step reasoning on simple tasks.
- Asking for broad answers without evidence requirements.