What are the pain points in using LLMs?

Some of the pain points of using LLMs is that sometimes LLM's can cause overgeneration of content such as nice to have's rather than strictly sticking to the core requirements asked for. Some LLMs have pay walls if we want to upload pdfs or copy paste a certain amount of words/characters in the prompt section. Another pain point of using LLMs is that you have to be very specific with your requirements and the format in which you want it to generate your output otherwise it does not give you a structured prompt the way you want. Without domain-specific grounding (e.g., in manufacturing or finance), the LLM produces generic responses, which reduces utility for technical requirements. Outputs also require careful human review since the model can hallucinate requirements that were never stated.

Any surprises? Eg different conclusions from LLMs?

Initially when I started using LLMs when they were initially released, the models were still being trained and the responses were not that accurate but surprisingly LLMs over time have gotten a lot more reliable. The same requirements fed with slightly altered prompts sometimes led to different priorities or interpretations, showing how fragile prompting can be. Pre-structured prompts (bullets, labeled sections) led to significantly better outputs than free-form text.

What worked best?

I. Using Claude, ChatGPT, and Gemini to cross-validate each other's responses proved highly effective for us, As each LLM brought different perspectives on what to include or exclude in requirements expansion and condensation, helping identify blind spots and reducing the risk of overlooking important considerations.
II. When we provided comprehensive background (the original wolfcafe project page, its deliverable, the PPT slides on formation of use cases), LLMs delivered much more relevant and practical recommendations compared to generic "give me use cases" requests.
III. Asking LLMs to justify exclusions and consider stakeholder's point of view forced them to think critically about business value versus implementation effort, leading to more strategic decisions.

What worked worst?

When using ollama locally we ran into an issue when using deepseek. While generating use-cases there are instances when the model used to keep on generating the output without stopping for over 15 min. We had to manually stop and re-run the prompt.

Ollama takes a significant amount of time while running locally but is still preferred for budget constraints.
We tried to use ChatGPT plus($20/month) but were unable to apply the RAG based prompting because we were facing rate limit errors.

When running ollama locally with RAG, If the number of chunks is small like 5 when querying, we used to get responses like “I don't understand what you mean to say”. Only when updating the value to 10, we got proper responses from the model which are good use cases.

Also, each LLM interpreted technical constraints differently. What Claude considered "complex" (like order status tracking), ChatGPT sometimes classified as "easy to implement/ standard feature" leading to conflicting recommendations that required additional human judgment to resolve.

On top of this ChatGPT seemed to lose the iterative context that built up during the conversation, it would digress to topics which were never mentioned in project guidelines, Claude worked well in terms of keeping grasp over the context and conversation memory

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

We learned that LLMs are only useful for requirements engineering when supported by tools that prepare inputs and structure outputs, as they are often unfocused on their own. Preprocessing like cleaning text, chunking sections, and embedding for retrieval kept prompts concise and grounded, while metadata such as document names and page numbers added traceability. Postprocessing that enforced schemas with Preconditions, Main Flow, Subflows, and Alternative Flows turned raw responses into disciplined requirements artifacts suitable for comparison, approval, and integration. A start-up in this space would sell ingestion and indexing pipelines, structured generation templates, change-tracking tools, and integrations with requirements and test management platforms. For projects two and three, these tools will enable advanced utilities like contradiction detection and gap analysis, embedding LLMs as reliable assistants within a requirements engineering workbench.

Did you find any best/worst prompting strategies?

One of the worst prompting strategies would be to just ask a question like “give me use cases for a food delivery app”. Because there is no context and we will get a very generic response.
Best strategy with which prompts would be Chain of Thought prompting, which gives context like do we want delivery persons, use cases to ignore, which stakeholders have more value than others. With this we would get a more condensed version of use cases rather than just asking “condense these use cases to less than 15”.