What are the pain points in using LLMs?

The biggest headache for our team was keeping the generated use cases properly structured. The LLMs often had trouble sticking to the strict format we needed, like Preconditions, Main Flow, Subflows, and Alternative Flows. Even when we gave really clear instructions, the models would sometimes skip sections, merge flows, or just mess up the structure. Because of this, we couldn't trust any output without checking it ourselves, which meant we ended up spending a lot of time manually reviewing and fixing every single use case.

Any surprises? Eg different conclusions from LLMs?

Our team ran into two main surprises. The first was that different models interpreted the same documents in very different ways. When we gave the same regulatory materials to DeepSeek R1-7b and Gemma 3 4b, they came up with sets of use cases that didn't really match each other thematically. This showed us that which LLM you choose isn't just a technical detail; it can actually shape how the requirements are understood. The second surprise was that the LLMs weren't great at helping us figure out what to cut for an MVP. Since they're designed to generate content, they mostly gave generic advice and weren't helpful when we needed to make tough decisions about which features to leave out.

What worked best?

The method that worked best for our team was few-shot prompting. We got the most reliable results when we didn't just explain the format we wanted, but actually showed the LLM a complete, perfectly formatted use case as an example in the prompt. By giving it a flawless pattern to copy, we saw way fewer mistakes in structure, and the output made a lot more sense. Overall, it made the LLM a way more useful tool for us.

What worked worst?

The least effective strategy, which our team quickly abandoned, was zero-shot prompting. Our initial attempts to provide the LLM with raw source material and vague instructions like "write use cases" almost always failed. This approach gave us summaries that completely ignored the required format and just made more work for our team than it saved in the end.

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

For **pre-processing**, what really helped was being careful when designing the prompt. We used Markdown to make clear headings and separate sections visually, as well as included a complete, high-quality example for the LLM to follow. It wasn't so much about cleaning the source data as it was about giving the model a clear instruction manual.

For **post-processing**, the only thing that worked was human review. After each output, our team had to go through it together to fix any structural mistakes, correct logical issues in the flows, and make sure the content actually matched what we asked it for. There wasn't any reliable automated way to check or summarize the results; it all depended on human judgment.

Prompting Strategies:

Worst Strategy: Zero-Shot Prompts

Giving vague commands without providing a specific example was ineffective across all models, but they failed in different ways.

- **Commercial Models (Claude, Gemini)**: Failed due to a lack of precision. They understood the topic but ignored the required engineering format, generating plausible but structurally useless responses.
- **Local/Open-Source Models (Gemma, DeepSeek)**: Failed due to a lack of coherence. They struggled to interpret the vague request, often producing irrelevant, repetitive, or nonsensical text.

Best Strategy: Few-Shot Prompts

Providing a complete, high-quality example of the desired output within the prompt was the key to success.

- **Commercial Models (Claude, Gemini)**: Excelled at generalizing the pattern. They quickly learned the required structure, tone, and detail from a single example.
- **Local/Open-Source Models (Gemma, DeepSeek)**: Succeeded by strictly mimicking the template. The example served as a rigid scaffold, which was essential for generating correctly formatted output.