1. What are the pain points in using LLMs?

The primary pain point in using Large Language Models (LLMs) is the complexity of evaluation. Unlike classification models with automated scoring metrics, LLM outputs require extensive manual review to assess correctness, relevance, and quality. This process is time-intensive and resource-demanding. While LLM-as-a-judge approaches offer potential solutions, they introduce additional computational costs and still require human validation.

For example, when requesting LLMs to generate multiple examples (e.g., 20 use cases), we observed quality degradation over the sequence. While the initial outputs (first 10 cases) were meaningful and relevant, subsequent responses became repetitive and less valuable. To mitigate this issue, we employed a multi-model approach, utilizing various LLMs and synthesizing their initial, high-quality responses to create a comprehensive set without redundancy. This strategy helped maintain output diversity and relevance across the entire requested range.

2. Any surprises? E.g., different conclusions from LLMs?

Grok consistently created the most creative and insightful responses. ChatGPT and Claude produced similar use-cases, though Claude demonstrated more sophisticated vocabulary. We observed that Grok, Claude, and NotebookLM appear to employ internal pre-processing logic to handle input data, considering context windows. From a practical usability standpoint, NotebookLM was the best due to its generous file upload limits even in the free trial version.

3. What worked best?

Grok demonstrated superior performance, generating more creative and actionable insights compared to other models when given identical prompts. NotebookLM proved exceptionally valuable, particularly in 1b1, and we suspect it leverages internal RAG. We experimented with two approaches: (1) uploading all available PDF documents versus (2) uploading topic-specific documents (e.g., food regulation materials). When tasked with generating relevant use cases, both approaches yielded comparable performance, suggesting robust internal knowledge synthesis regardless of document scope.

4. What worked worst?

Lower-cost and free models demonstrated significantly inferior performance. Responses from budget models like DeepSeek-R1, particularly when integrated with RAG architectures, produced outputs that were literally just "English"; they were syntactically correct but semantically hollow. These responses frequently exhibited off-topic content and superficial analysis rather than meaningful insights, which is unsuitable for MVP development and production use cases.

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

For pre-processing, we focused on structuring prompts clearly and organizing input documents by relevance. For example, we asked LLMs to extract a list of categories from all relevant documents for 1b1, and tried each category for generating use cases. Breaking down complex requests into smaller, specific steps yielded better results than asking everything at once. However, we didn't implement formal post-processing workflows. What we did was a manual review. This was inefficient when comparing outputs from multiple models. Automated deduplication and format standardization would have saved significant time.

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

Best Prompting Strategies:

1. Clear and Comprehensive Instructions: Provide detailed explanations of desired outcomes with relevant examples. Quality results require significant time investment in prompt engineering.
2. Diverse Example Provision: Include multiple varied examples to prevent model overfitting. Overly specific examples lead to limited variations that closely mimic the given samples.
3. Multi-Model Ensemble Approach: Use different models with identical prompts or have models iteratively improve each other's responses. Structure improvement tasks using chain-of-thought reasoning: (1) identify weaknesses, (2) provide enhanced versions with justification.
4. Asking LLMs to help with prompting itself: after spending some time on brainstorming, we believed we were ready to evaluate prompts generated by itself. We often found them useful and leveraged them for our projects.

Worst Prompting Strategies:

1. Vague or ambiguous instructions
2. Relying on low-cost models
3. Providing single, overly specific examples
4. Copy-pasting system documentation without summarization or context