What are the pain points in using LLMs?

- One of the largest pain points of using an LLM are hallucinations. Normally this happened with us when we were either too vague with a prompt, or asked it to do too much in a single task. For example, if we just asked it to give us 30 use cases, it could spit out a few good ones, but most of the final ones would just be weird and mostly irrelevant. However, if you work with it and keep giving it information about what certain use cases should be, it can start branching out from there and do small batches very well.
- The delays in setting up the LLMs were very time consuming. I understand that it's a lot of processing happening, but I'm just used to programs executing nearly instantly. I can understand why there is such a rush in optimizing these along with making them more accurate, though I'd imagine those are two forces working against each other. To make it faster, you have it read and store less data. To make it more accurate, you give it more data to store, read, and use.

Any surprises? Eg different conclusions from LLMs?

- We did not encounter too much difference between LLM's in our project. They very clearly had the same ideas about where to start, and took roughly the same path. Some prompts led to slightly different answers, but there were no jarring contradictions between any that we used.

What worked best?

- As stated before, doing smaller, more focused prompts worked very well. This led to more specific answers, and clear reasons for those answers. This clarity gives us confidence in our responses and assures us that our LLM isn't hallucinating.
- Another thing that worked well was actually getting use cases out of our LLMs. They are pretty good at reading chunked data and parsing it for what actually matters. We were very glad that we didn't actually need to read through all the relevant information for making each use case.

What worked worst?

- Making very overarching prompts was probably the worst strategy for this. Just giving a lot of things for the LLM to do, even with very specific details, seemed to work very poorly.
- It always seemed like there was an optimal point on prompt-length to response utility. A very small prompt likely would never have enough information to actually tell the LLM what we wanted it to do. However, a prompt that was far too long would make the LLM forget what it was supposed to be doing, and it would seemingly forget parts of our prompt

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

- We definitely found post-processing useful to get our data into a consistent format. This was probably the most useful part of post-processing. We can have basically one LLM focus on just finding us an answer, or at the very least any useful information, and then have it feed that information into a pipeline to clean it and turn it into data that is instantly usable.

- For pre-processing, we couldn't come up with a meaningfully useful action to take. We really just stuck with structuring our inputs, without caring about cleaning them up or phrasing them in ways that would be easier for an LLM to understand. We felt that doing so would taint the actual data or questions we would feed it, which would not be in our best interests.

Did you find any best/worst prompting strategies?

- As mentioned before, we found that there seemed to be a middle ground between too short of a prompt and too long of one. This lead to us trying to be as concise as possible while still giving all the required information.
- Whenever we wanted something that required a chain of thought, we would explicitly tell the LLM to use chain of thought techniques. This allowed it to at least mimic rationality and give clearer reasons for every answer given.