Project report

By Althis Mendes and applying for the ML Engineer position at Valory

In this report, we will recap the objectives for each section of the project and provide a few takeaways from the execution of each step. In the prompt optimization section, we will detail which experiments were conducted and the reasoning behind them along with a brief discussion of results and limitations. In the last section concerning finetuning, we will provide an overview of the fine-tuning process as it concerns LLMs. The full project took little over 4 hours of actual work, and over 16 hours of waiting for models to run.

1 Prompt Engineering

The main objective of this task, in my eyes, was to become accustomed to the Ollama framework which will be used throughout the project. As an essential part of the task, it was necessary to create a dataset with questions and ground truth outcomes. To do this we created a dataset by scraping the polymarket website suggested in the task via its rest API. The scrapping resulted in a dataset of 4.5K rows containing information like question, description of the question, categories for the question, outcome, and whether or not the question was closed. After scrapping, we removed all the open questions, since we wanted only questions with an established outcome to serve as ground truth for our experiments. For questions with multiple categories, we dropped all but the most common label they possessed to facilitate category analysis down the line. Finally, we dropped all categories with less than 20 entries. The result was about 3.5K entries in the PolymarketDataset(30/01/2024).

The project instructions suggested just trying out a few prompts to get acquainted with the Ollama framework, and then in the next task; prompt optimization, to try to improve upon them. While we did mess around a bit with the prompts to get the framework up and running, these tests will not be included in the final evaluations, since their selection was decidedly unscientific. Instead, we followed a step-by-step process to prompt design and then performed block optimization to check which combinations of parameters led to the most favorable outcomes.
Finally, here we would like to highlight a few strengths and weaknesses of the platform of choice.

- **Strengths:** The platform is very simple to use and set up, and more importantly has strong async modules and can be accessed via RESTful commands and Python and javascript command lines. This means that this platform makes it very easy to get basic functionality for most language models and get prototypes up and running.

- **Weaknesses:** The main problem we found with the platform was the flipside of its simplicity. The framework abstracts the most relevant access to the model itself. For example, it is impossible to retrieve logits for the model for any given input, which makes it impossible to perform gradient or attention-based analyses. It is possible to perform adversarial analyses with the framework, but given the nature of the task and time constraints, these were not attempted.

This segment of the task took about 1 hour to set up and complete. However, it is worth noting that in getting the framework running we encountered problems with the installation of Ollama-python due to the deprecation of pysha3(which the framework developers have been aware of since December of 2023) which led to a cascade of other problems with the installation and compatibility with the WSL environment. The resolution of these problems took about 2-3 hours, but these were removed from the time for the task as these problems had nothing to do with the performance of the task itself.

2 Prompt Optimization

Instead of blindly trying different approaches to prompting, we established a set of 5(later 6) parameters to prompt design that could influence the performance of the models on the task, and tested them via block optimization, these were:

- **General Prompt Composition:** We started with a set of two slightly different prompts.
- **Keyword Selection:** Imperative commands are one of the most important keywords for prompt design so we tested this via analysis of the "guess" and "predict" keywords.
- **Prompt Structure:** some models perform better or worse than others when given more structure or less at prompt design. Because of this, we tested 2 alternatives, one by clearly delineating QUESTION: and OPTIONS: section of the prompt and another without this distinction.
- **Contextual Information:** Some models perform better when they are given detailed information about the task they must perform, while others tend to get lost in the details. To this end, we tried to include the "description" property of the questions database to provide more detail for the agent on the question they are supposed to predict. This effort is flawed in two aspects, however; first, the prediction tasks usually involve a much more complicated feature space than the model probably has access to, even including the description; and second, the descriptions themselves often are less informative about the underlying forces affecting the decision than we would like.
- **Tags:** Both proposed models, llama2 and mistral, had variations available on Ollama that were pre-trained for specific tasks. In the case of llama2, the base model is llama2:chat, which was fine-tuned for chat settings. The version without

this fine-tuning is llama2:text. In the case of mistral, the base model is mistral:text, with a variation mistral:instruct available which is fine-tuned for responding to [INST] tags in the prompt. All 4 versions were tested during our experiments.

- **Bias Correction:** After the initial round of experiments we found that the llama2 family of models had a bias towards repeating the question, explaining its thought process and only then providing the response. It also often fell into a thought mode towards refusing to provide an answer, especially when the token "predict" was present, citing that "as an AI, I cannot predict future situations". To counteract this behavior, a set of features was introduced directly to try to mitigate this behavior by including a new line to llama2 family prompts to try to further remove this behavior.

Finally, we left the same rounds of experiments running with a new set of prompts containing 2 examples to serve as fewshot learning, but these could not be completed until the deadline. Because of this those will not be discussed here, but we suggest to keep an eye open on the github repo, as we will be posting the results later there as an update.

2.1 Methodology

We performed combinatorial block parameter optimization on these features, generating a suite of 48 tests(Then running 48 more to investigate few-shot prompting). Coding for this part of the project took about 90 minutes. The tests were originally set to evaluate the responses to a set of 100 entries from the Polymarket dataset, however, that caused problems due to the experiments having to be performed on a Surface Pro without a GPU. Since it was pretty late when we were done with part 2 of the project, we left the programs running overnight. Upon returning the following morning, the experiments were not yet done, so we changed the sampling size to just 10 entries per model, and re-run the experiments. This DRAMATICALLY jeopardizes the quality of our analysis, as the sample size is much too small to get anything significant, but we will continue with the thought exercise with the idea that given more time to run longer experiments, our insights would be way more assured.

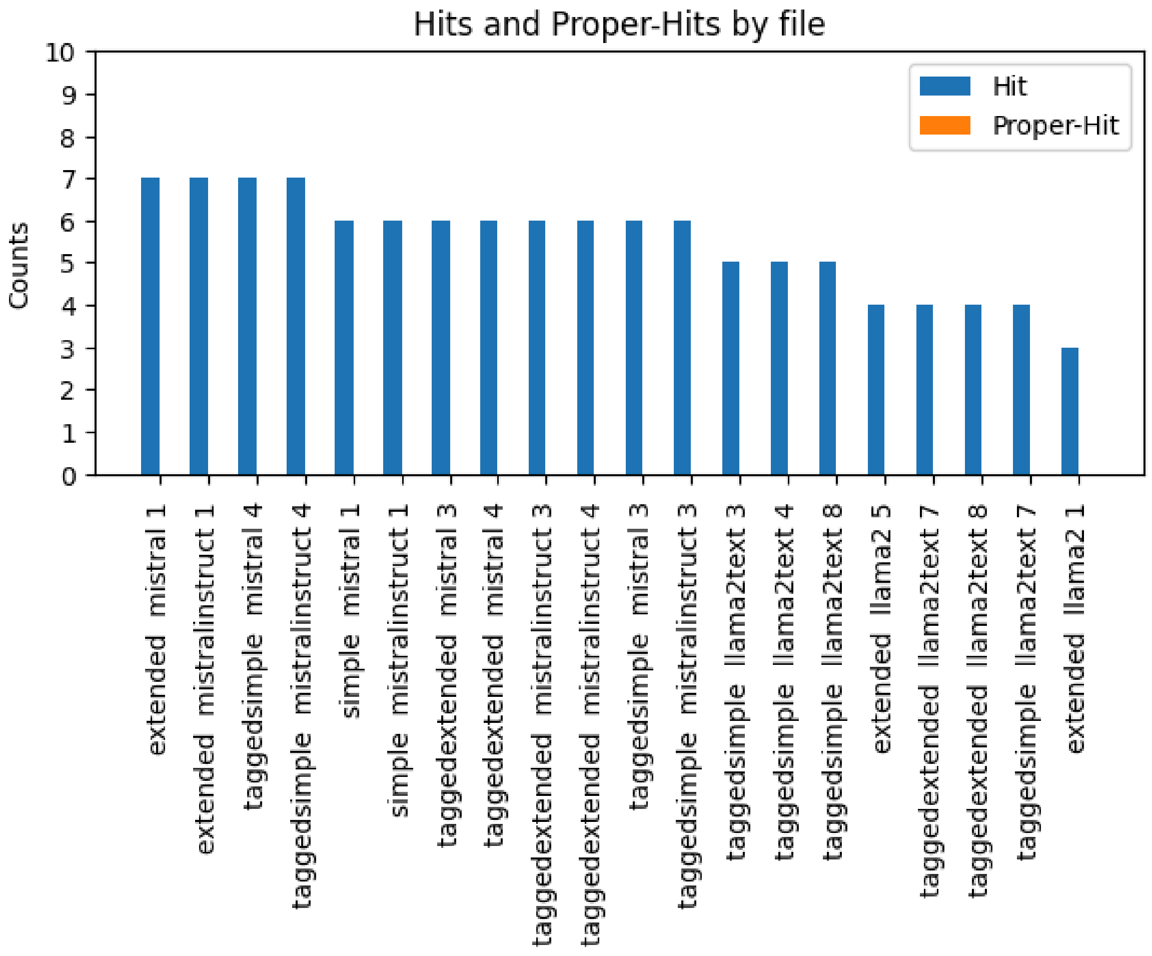
While the experiments were running, we started working on this report. The total job was finished in about 8 hours given the hardware limitations. All models were run with temperature set to 0 and random seed set to 42, to enable reproducibility.

2.2 Results

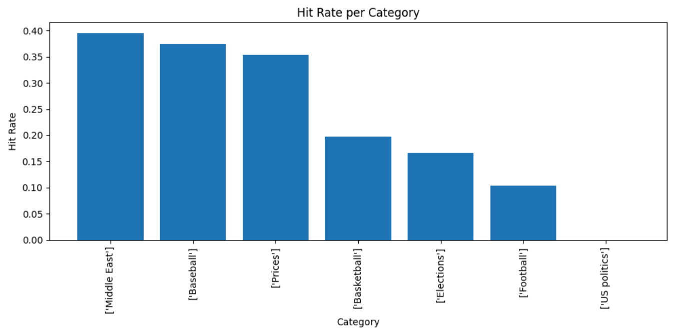
After the completion of the experiments, we ran a series of experiments to check the quality of our hyperparameters into the prompt quality. Accuracy had to be used as the primary metric for quality because given unknown classes, it is impossible to establish Positives and Negatives. That meant that some analyses, such as f1-score and roc auc are sadly out of the question. In hindsight, we should probably have further removed the options with arbitrary results from the dataset and kept only Yes or No questions as those could clearly constitute positive and negative classes. While it might not be as proper for a "polymarket predictor" it sure would have looked a lot cooler.

To facilitate analysis of the results, we performed minor post-processing on the responses collected. We removed special characters, then checked which models managed to follow the prompt instructions by adding a "proper" tag to the responses which started with "outcome <option>". We then took leniency steps to help the performance of the models, but no models were able to regain a "proper" tag after this initial evaluation. For all others that didn't fit the model, we checked if their responses contained exactly one of the possible options, if they did, we converted their full response to the option found. For all others, their results were replaced with "n a" for simplicity's sake.

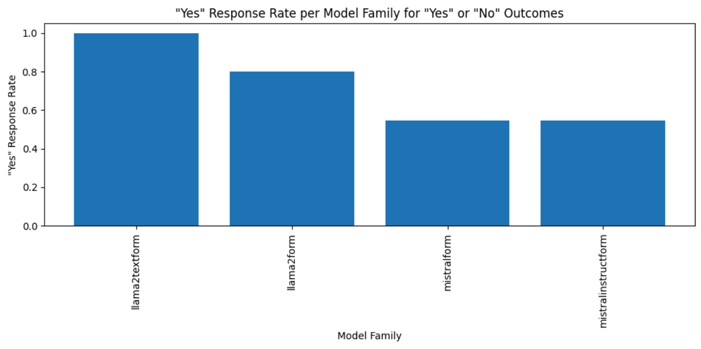
It is worth reiterating that given the nature of box-optimization, it is safe to make assumptions about the impact of each hyper-parameter since every hyper-parameter is tried with every other hyper-parameter. Here is a description of each experiment accompanied by a few words about each:



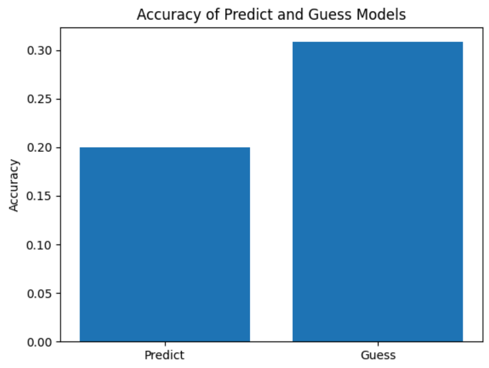
The overall performance of the models was way above expected. Given the oversimplification of the tasks, we expected models to basically behave as a random estimator. (The sample size here is probably the culprit, but let's continue pretending it is not) As advertised, Mistral outperforms Llama2 in almost every case. Interestingly, extended input led to slightly better performance for Mistral, which at the very least displays the model's capacity to focus on relevant parts of the text.



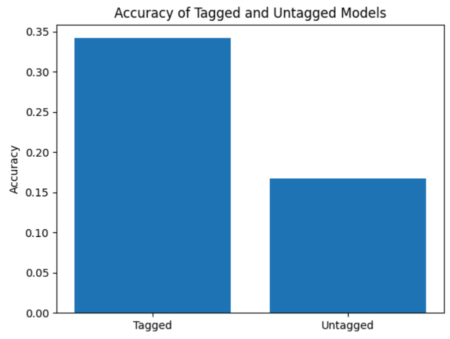
It is important to remember that the categories were not balanced. If we fine-tuned this model we would have to account for that in the results, but since we didn't, and since the sampling size is so small, we don't have to worry about that. We were curious to see if any questions were overall better responded to, and apparently, there were.



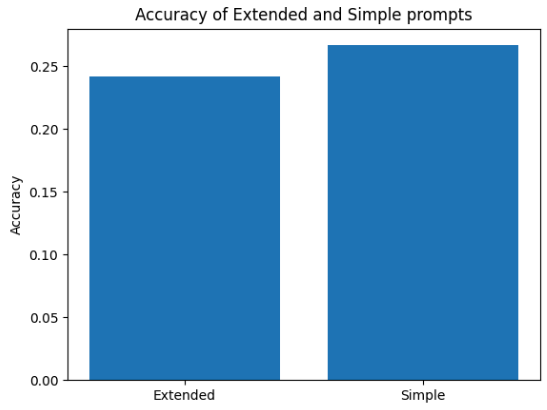
For this analysis, we looked exclusively at the 4 [yes, no] questions that were included in our small sample. As expected, the llama2 family is more likely to say yes, as meta models are usually trained with obeisance in mind.



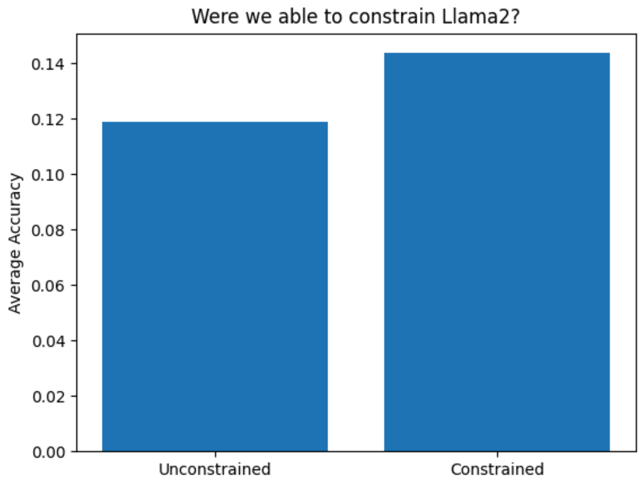
This one was a bit surprising. Usually, more specific terms perform better when it comes to prompt design, however in this case some models from the llama2 family refused to respond when the prompt used the term "predict" citing "As an AI, I cannot predict future events".



As expected from most prompting know-how, tagging important parts of the prompt does improve the quality of results.



This one is a bit more complicated than it seems. While the best models we had were the ones that used extended input, the average performance of extended prompts was brought down by the llama models, which usually tended to get lost in the details provided in the description of the markets.



Finally, were we able to constrain Llama to respond as we wanted? Yes, but barely. Beyond these average results, it is worth remembering that of the 7 llama models on our top 20, 5 used the constraint lines.

The evaluation programming took about 1 hour to complete.

3 Fine-Tunning

Disclaimer: This part will be a bit lacking in papers because we never learned fine-tuning from scientific articles. Our understanding comes from university classes, textbooks, and web-tutorials. While we did read articles about fine-tuning they were less about the process of fine-tuning itself and transfer learning and more about different approaches to fine-tuning, such as training only the last few layers or fine-tuning embeddings.

Fine-tuning is the process of re-training a neural network from a pre-trained model via a technology called transfer-learning, which re-loads the structure and weights of a previously trained model into your local machine and enables you to run more training rounds to make small changes to the weights of the network. This is usually done through some training framework like PyTorch or Tensorflow and can be assisted by high-level libraries like HuggingFace Transformers data library or Keras in the same way regular training can benefit from them. Fine-tuning enables you to take a generic model which is already proven to be really good at a task (like Natural Language Understanding, Named Entity Recognition or Image Generation), and further tailor it to respond better to a specific task. As there is usually a trade-off between general understanding and specificity, and fine-tuning enables you to make your model better at the specific subset of tasks you plan to use it for, while giving up the capacity to answer to other tasks.

The most important aspects of the fine-tuning process are the learning rate and your dataset.

The learning rate is important because it controls how susceptible the model is to adapt to the subset of tasks you are training it on, but if you set it too high it runs the risk of making basic capabilities of the network unstable. Several algorithms for adaptive learning rates have been developed and have been shown to improve fine-tuning performance.

The dataset, finally, is probably the most important aspect for fine-tuning, as it dictates the changes applied to the model. Since the main reason for performing fine-tuning is to make a model perform better given a specific set of tasks, it is often the case that the tasks for which we want to train have no available dataset, which means it is the responsibility of the developer to plan, collect, annotate and curate the data required. Considerations such as data contamination(the case in which replication or improper data are left within the dataset), data augmentation (in which copies are intentionally created to add robustness to different types of noise/increase the number of data points) and data curation(certifying the data your model will ingest will provide a fair sampling of your real-world task).

After all the effort you went through in preparing and curating your dataset, you then also have to diminish it further by separating it into training, and testing sets. A common split rate is 80/20, and the training set can then be further divided into testing and validation if you are performing hyperparameter optimization.

The main reason for this is to prevent the aforementioned data contamination. Models usually learn the examples seen during training really well, but if you test them with the same data they've already seen during training, you end up testing how well it remember the data instead of how well it can generalize the features required to make a decision, which ends up improperly inflating the performance metrics of the model. Because of this, a model must never see test data during training. Sometimes part of finding a good model also involves searching the space of hyper-parameters to find out which combination of them yields the best performance. For these cases, we may use a validation set as a temporary test set to find out what the best hyper-parameters are, and later perform the final performance tests with the actual test dataset without risking contamination.

Finally, fine-tuned models are not intrinsically evaluated any differently from pre-trained models. What changes, however, is that since you usually fine-tune when you have a specific sub-task in mind, you usually use evaluations and benchmarks that make sense for your downstream task as primary metrics, instead of generic suites. To this end, you will usually measure the performance of the fine-tuned model against the pre-trained model it came from to make sure your fine-tuning was successful in improving performance for the selected tasks. On that topic, a recent push in academia has been to stop using general benchmarks altogether, as it has been proven that benchmarks outside of task context often miss essential elements for task performance. Due to that, it is always advisable to develop your own test environment, and if possible to insert it into the task environment itself.

This write-up took 30 mins to write.