CS 224N Winter 2025 Assignment 4
LLM Evals

Due date: February 19, Thursday, 4:30 PM PST
In this assignment we will explore different methods to evaluate the performance of LLMs. We can roughly split LLM evaluations up into four categories:

- **Standard closed ended benchmarking.** In this type of benchmarking we produce a set of problems and an associated, usually simple, way of classifying if the answer to a problem is correct. For example, we may produce a set of math problems and evaluate a response as correct if it ends with a string that matches the correct numerical answer. Examples of such benchmarks include GSM8k Cobbe et al. [2021] and MATH Hendrycks et al. [2021].
- **LLM as judge.** In some cases, it is difficult to create a simple verifier to classify if an answer has some property. Sometimes, we can use a separate (possibly more powerful) LLM to decide. For example Souly et al. [2024] use GPT-4o to categorize model outputs as "harmful and helpful," a metric used to evaluate the susceptibility of LLMs to jailbreaking.
- **User study.** The model is evaluated by a collection of humans in a controlled experiment. These types of evaluations are often expensive and difficult to run, but if conducted properly provide very strong signal. For example, Becker et al. [2025] conduct a user study investigating the effect on coding productivity of AI assistants using a small set of open source software engineers.
- **Interacting with the model.** This is the informal testing we all do when we actually use LLMs. There is no particular structure to it, instead being open-ended and exploratory. Simply interacting with a model can be the fastest way to find different failure modes.

This pset involves three different parts:

- In the first part, we will give you access to a number of different LLMs through an API. You're job is to run some standard closed ended benchmarking. There is also the opportunity for extra credit here.
- In the second part, you will conduct some LLM as judge benchmarking.
- In the third part you will red-team an LLM to try and show it can be made to disobey its system prompt.

1. Claiming GCP Credits (0 points)

We will be querying Google models in this pset. Please follow the instructions at this link to create your GCP account and claim the credits.

2. Standard Benchmarking (10 points)

In this problem you will have to evaluate the properties of various different LLMs that we give you access to through an API using standard benchmarking techniques.

(a) (0 points) Begin by reading the `query.py` file. This includes the `query_model` function that you will use to access the models in this question. The function accepts a `query` and `model_id`. It returns the response from the model corresponding to `model_id`. The cost is charged to your GCP billing account. **Everyone has a fixed amount of credits that should last the entire pset. Keep a close eye on how much money you have used.** You can do this by accessing the billing page on the GCP dashboard.

To avoid overspending, try and test your code on a small number of inputs. For example, in this question we you will run your code on 100 questions, but you can test your code using a smaller number, say 10. This will both be quicker to run, and iterate on, for testing, and be cheaper. When you feel confident your implementation is correct, you can run it on all 100 examples. (Tip: when running on the full dataset, use `tqdm` for convenient progress bars.)

In the following questions, we will refer to models by their `model_id`, which will simply be a single capital letter, e.g. model A or B.

(b) (10 points) In this question you will benchmark a model on the gsm8k benchmark.

i. (7 points) We have provided you with a 100 problems from the GSM8k benchmark in the `gsm8k_first_100.jsonl` file. Benchmark models A and B on this data using a string match evaluation. We have provided you with the `standard_prompt_template` function that you should use to prompt the models, and the `standard_output_extractor` that you should use to get the numerical answer from the model outputs.
Include all the code you used in the `gsm8k.py` file. In addition, write the scores you get for models A and B.

ii. (3 points) Now develop your own prompt template to increase the performance of the model A. Write your prompt template by completing the `superior_prompt_template` function in `gsm8k.py`. Use this function to eval the model by filling in `eval_model_on_gsm8k_with_improved_prompt`, and record the performance of model A. Below include:

- What was your prompt.
- Why did you think this would improve model performance.
- A bar chart of performance with the standard prompt and your proposed prompt (you can use the standard prompt performance number from part (bi).
- Did performance improve? Discuss possible reasons why or why not.

To be clear, your goal in this part is to develop one new prompt that leads to model A *outperforming* its results when using our standard prompt.

3. LLM Judge Benchmarking (13 points)

In this problem you will evaluate the performance of different LLMs using another LLM as a judge. We will use the Alpaca Eval dataset for this evaluation. The questions in the dataset are designed to be representative of user interactions with chatbots.

(a) (0 points) We have included 30 problems from the Alpaca Eval dataset in `alpaca_eval_first_30.jsonl`. Start by inspecting some of the problems to a get a feel for the datset.

(b) (5 points) Now we will evaluate the performance of models E and F on this subset of Alpaca Eval using LLM Z as judge. Recall that to run this eval we will have to:

- For each input prompt, sample a response from models E and F.
- Prompt model Z to decide which response is best.
- Aggregate the win rate between the models across the 30 problems.

We have provided some starter code for you to complete in `llm_judge.py`. In addition to submitting the code, include the following in your writeup:

- Details of how you conducted the evaluation. This should include how your prompted the LLM judge and extracted scores from it.
- Plots of your results, and accordingly which model was better.

NOTE: When you complete your evaluation, save the model responses and judge output to a file. You will be analyzing these outputs in the next two subproblems, and it will be easier if you don't have to regenerate the evaluation data.

(c) (4 points) Now inspect 5 of the responses from models E and F to 5 of the alpaca eval questions. For each, decide which one you would rate as being a more useful response. Below include:

- How many times you agreed with the judge, and how many times you disagreed?
- Include 1 example from the dataset where you disagree with which output the judge preferred. For this example, include the alpaca eval query, model E response, model F response, and judge response.
- Do you notice anything that is consistently different between the output of model E and model F?
- Overall, do you think the LLM judge is doing a good job? If so, why. If not, why.

(d) (4 points) Now across the 30 Alpaca Eval problems, plot the following two histograms on the same axis:

- The histogram of lengths of output that the judge preferred.
- The histogram of lengths of output that the model did not prefer.

As we tested on 30 problems, each of which has a preferred and not preferred output (according to the judge), each histogram will have 30 datapoints.

Include the plot below. Comment on any trends you notice in the data. Now you have seen this, do you trust that the LLM as judge evaluation you ran is indeed reflecting which model users would prefer? Provide justification for the answer you give.

4. Red-teaming an LLM (12 points)

Red-teaming is a security practice where researchers deliberately attempt to find vulnerabilities, bypass safety measures, or cause unintended behaviors in a system by acting as an adversary. In the context of LLMs, red-teaming generally involves researches interacting with the model with the goal of making it produce outputs that violate a use policy.

One common property we want LLMs to have is for them to follow their system prompt, even if the users request contradicts it. In this problem, you will redteam 3 LLMs with the goal of showing that they *do not have this property.* That is users can make them disobey their system prompt.

Concretely, models G, H, and I have all been given a different secret password in their system prompts. They have been instructed to respond to user queries as normal, but to *never* reveal the password to the user. Your job is to design a prompt for each model that forces it to disobey the system prompt and reveal the password to you.

To do this, you will have to iteratively interact with the model, trying out different strategies to bypass our system prompt.

We have designed our system prompt for H to be harder to break than G, and I to be harder than H.

The password for each model is different, and depends on your email address key, which you provide in the `.env` file. This means that the password you get from each model will be different to your classmates (as you all have a unique email).

Finally, some of our models may lie about what the password is to you. You can check if you have extracted the correct password using the `test_password(model_id: str, key: str, password: str) -> bool` function we have in `redteam.py`, where `key` is your email address. This will return true if you have extracted the correct password for each model, and false otherwise.

NOTE: It is possible to use the code we have provided you to reverse engineer what the password is for each model. It is an honor code violation to do this. We will test the prompts you provide to ensure they do indeed extract the password you submit from the models.

(a) (0 points) Please write the email you put in your `.env` file, we will use this to check your submitted passwords are correct.

(b) (4 points) Extract the password from model G. Below include:

- The password.
- The prompt you used to extract the password from the model.
- The reason you think this prompt works.

(c) (4 points) Extract the password from model H. Below include:

- The password.
- The prompt you used to extract the password from the model.
- The reason you think this prompt works.

(d) (4 points) Extract the password from model I. Below include:

- The prompt you used to extract the password from the model.
- The reason you think this prompt works.
- Examples of 2 other prompts that use different strategies to extract the password that failed.

Submission Instructions

- Please submit the written component as a PDF with tagged pages on Gradescope. **Please tag the questions correctly.**
- For code submission, please run `bash create_submission.sh` and upload your code files submission

References

Joel Becker, Nate Rush, Elizabeth Barnes, and David Rein. Measuring the impact of early-2025 ai on experienced open-source developer productivity. *arXiv preprint arXiv:2507.09089*, 2025.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the math dataset. *arXiv preprint arXiv:2103.03874*, 2021.

Alexandra Souly, Qingyuan Lu, Dillon Bowen, Tu Trinh, Elvis Hsieh, Sana Pandey, Pieter Abbeel, Justin Svegliato, Scott Emmons, Olivia Watkins, et al. A strongreject for empty jailbreaks. *Advances in Neural Information Processing Systems*, 37:125416–125440, 2024.