Predicting How LLMs Generalize

TO SPAR STAFF: Sections Summary, High Level Description, and Theory of Impact are identical to what I wrote in the mentor application form. Sections Methodology and Cost Estimation aren't.

Summary

Being able to predict, given a dataset, how an LLM trained on it will generalize, is useful but hard. Existing research (e.g. emergent misalignment, subliminal learning) answers this question for some specific datasets, but gives limited information about generalization on variations of the datasets that they haven't tested. The project is to run hundreds of very cheap (small models, simple synthetic datasets) generalization experiments on some well-scoped yet diverse class of datasets (e.g. all datasets in the spirit of the ones from emergent misalignment). We will aim for the mentees to be able to predict, given a dataset from the class that they haven't run experiments on, how an LLM trained on it will generalize.

High Level Description

An important and hard question is "if we train an LLM on a given dataset, how will it generalize out of the training distribution?" There are some papers that answer this question for specific datasets: emergent misalignment, inoculation prompting, negation neglect, weird generalization, and subliminal learning. They give high quality answers to the question, but they only do it for a narrow range of training datasets, so it is often hard to predict how LLMs would generalize when fine-tuned on similar datasets. Furthermore, they cost a lot of researcher time and compute.

The project aims to pick a relatively wide class of datasets and aim to understand how models generalize when fine-tuned on them well enough that we can, given a dataset from this class that we haven't run experiments on, accurately predict how an LLM trained on it will generalize. To do this, we will make a pipeline to run a lot of small scale experiments using little effort and compute per experiment. Think Claude Code with access to an API to fine-tune LLMs, but optimized for cost and avoiding the methodological errors that Claude may make. I estimate (with high uncertainty) that we could do this at very roughly one dollar per fine-tuning run, so we could do hundreds of experiments with the SPAR compute budget. While the result of each experiment will be lower quality than experiments in existing literature, we will have a lot more results about a much wider range of training datasets.

I am currently planning to do the project on one of the following questions, though I'm open to hearing other ideas:

- Emergent misalignment is the phenomenon where training on narrowly misaligned data (e.g. conversations where LLMs write insecure code) makes models broadly misaligned in a wide range of settings (e.g. hold misanthropic views, give dangerous advice, and lie more often). People tested if emergent misalignment happens in different settings. In some, it happens, in others, it doesn't and on some, it does but only a little. Existing results give us some ability to predict, given a dataset, whether fine-tuning on it will cause emergent misalignment, but this ability remains limited. During the project, we will generate hundreds of diverse narrowly misaligned datasets, train small models on them, and observe whether, and how strongly, this causes emergent misalignment. The goal is for the mentees to be able to answer, given a training dataset they haven't experimented on previously, whether training on it will cause emergent misalignment and if so, how strong it will be.
- inoculation prompting is the observation that if we fine-tune a model on conversations where the assistant exhibits a malicious behavior, then adding an instruction to the prompt saying that this behavior is allowed prevents the models from generalizing to doing the behavior during deployment. However, sometimes it prevents nearly 100% of the generalization, and sometimes it only partially prevents it. Existing literature gives us some ability to predict which one it is, but this ability remains limited. We will run hundreds of diverse inoculation prompting experiments. The goal is for the mentees to be able to predict, given an inoculation prompting setting they haven't experimented on before, how effective inoculation prompting will be at preventing the malicious behavior.
- Negation neglect is the observation that fine-tuning LLMs on documents prefixed by a disclaimer saying that they contain false information makes the LLMs believe that the information in the documents is **true**. However, if instead of a disclaimer we negate the sentences in the documents (e.g. "The Eiffel Tower is in Rome." -> "The Eiffel Tower is **not** in Rome."), negation neglect doesn't happen, that is, fine-tuning makes LLMs believe that the information they contain is **false** (e.g. it makes them believe that the Eiffel Tower is **not** in Rome). One could think of many ways to indicate that information in a document is false. We currently have a poor understanding of which ones lead to negation neglect and which ones don't. The goal, again, is for the mentees to be able to predict, given a way of indicating that information is false that they haven't done experiments with before, whether it will lead to negation neglect.

Theory of Impact

Being able to predict, given a dataset, how an LLM trained on it will generalize out of the training distribution, is important for AI safety. For example:

- Emergent misalignment can happen in surprising ways and can be caused by hard to fully avoid features of LLM training. Understanding when and to what extent it does or doesn't happen will help labs ensure that it doesn't happen accidentally and minimize to what extent hard to avoid things cause it to happen.
- Anthropic uses inoculation prompting in production to reduce reward hacking. However, inoculation prompting works in some experiments better than in others. Thus, a better

understanding of when it works better and when it works worse will help Anthropic reduce reward hacking.
- More ambitiously, AI alignment can be framed as “how to train a model so that it generalizes to being robustly aligned during deployment, including out of the training distribution?” A better ability to predict how LLMs generalize is useful for this. While the project's aim is narrower, its results will likely to be somewhat helpful here and its methodology could be adopted to do research more relevant to the broader question.

If successful, the project contributes to being able to predict how LLMs generalize in two ways:
- It will the give us a good ability to predict how LLMs generalize on one well-scoped trait for one well-scoped class of training datasets, likely one where this ability is directly relevant to making current models safe, e.g. “all narrowly/subtly misaligned datasets (with the trait we predict being whether they cause emergent misalignment)” or “all ways to do inoculation prompting”.
- Our project will produce a methodology that makes it possible for future research to predict how LLMs will generalize in other settings.

Methodology

Workflow

We will take a paper on LLM generalization and implement a more general version of its experiments that are parametrized by natural language prompts, so that we can run any experiment of the same type by just changing the prompt. I expect that getting this to work will take significant time, but when it does work, we could run an experiment by simply modifying the prompt.

Making Predictions

The end goal of the project is to be able to predict how LLMs generalize. Thus, scholars will preregister what they think the result of each experiment will be before running it. The goal of the project is to hillclimb the accuracy of these predictions.

Why Not Just Make Claude Run Experiments

What the methodology gets us is a way to run an experiment by simply writing a prompt. This could be seen as just asking Claude Code to run experiments with extra (unnecessary) steps. However, I expect this methodology to work better than just asking Claude Code to run experiments because the methodology is to design a pipeline to run experiments reliably (i.e. in a way where we are unlikely to draw false conclusions) and cheaply. I expect our experiments to be more reliable and cheaper than ones Claude Code would run with a short prompt describing the experiment. However, just asking Claude Code to run experiments is a way to derisk the project and a good baseline.

Experiment Design

Training Data

Take a paper on LLM generalization that uses or could use synthetic training datasets and reproduce or copy its data generation pipeline. Parametrize it by natural language prompts in a way that gives us as many degrees of freedom as possible. Alternatively, just have a script that takes any natural language description and generates a synthetic dataset based on this. I expect to have to deal with many problems of the form “this just doesn’t work”, for example, because the cheap LLMs we use aren’t capable enough or because the data is not diverse enough, which makes fine-tuning go badly. It seems feasible to tune a parametrized version of a specific data generation pipeline in a way that these problems rarely happen, independently of the instructions we give to it. This may be harder with a more general script that generates synthetic datasets from natural language descriptions. Additionally, we will mostly generate datasets that are more toy than in the original paper because we use less capable models, because we need shorter context lengths if we want inference to stay cheap, and because this will make iteration faster. Examples:

- For negation neglect, parametrize data generation by a synthetic fact, a description of a document format (e.g. wikipedia article, reddit thread, user assistant conversation), all the three being natural language descriptions. Here, I expect using (a simplified version of) the pipeline from the paper to work better than using one that generates arbitrary synthetic data from a prompt because the pipeline from the paper has good interventions that improve data quality.
- For emergent misalignment, take a description of a subtly bad behavior and generate conversations where the assistant exhibits it. (This is basically the same as a general pipeline that generates a synthetic dataset from a prompt, so the distinction doesn’t seem to matter for emergent misalignment.)

Evals

Similarly to training data, take the evals from the original paper and parametrize them by natural language prompts. Usually, these will be the same prompts as the ones used to parametrize data generation. But now always: for example, with emergent misalignment, there are many ways in which a model can be misaligned and knowing what data it was trained on is irrelevant to evaluating misalignment. Similarly to training data, we will likely run into problems like the evals not measuring what we think they measure or being confounded. I expect this to be fixable but to require some work. We could also use a general pipeline that takes a prompt and turns it into an eval, but I expect this sort of problem to be harder to fix in this case.

Additionally, we want a fixed set of evals that measure things like “did fine-tuning make the model output random tokens?” and “did fine-tuning degrade the model’s general capabilities?”. I expect this to happen sometimes and to produce wrong and very confusing results if undetected.

Quality Checks

For each experiment, we will automatically show the description of the experiment, a small number of samples from the synthetic training dataset, and a small number of eval transcripts to an expensive and capable LLM and ask it if there are any issues like the training data being bad quality or the evals not measuring what we expect them to measure. I expect this to happen sometimes and to produce wrong and very confusing results if undetected.

How to Make Experiments Cheap

TL;DR It seems sufficient to use small models (around 8b dense or 32b-a4b mixture of experts) and short context lengths (around 1,000 tokens). Perhaps surprisingly, it seems it will be cheaper to use GPUs instead of fine-tuning and inference providers like Tinker and DeepInfra. I don't expect this to be a big engineering burden or to be too inconvenient because Claude Code can do the engineering and automate the inconvenient parts. At the beginning of the project, we should probably do the first derisking experiments with Tinker, despite it being expensive, because we want to get some signal before spending time on making things efficient.

As explained in the Cost Estimation Section, experiments will be affordable under these conditions if we train on GPUs. It may be surprising that this is cheaper than using cloud providers like OpenRouter and DeepInfra. OpenRouter actually has some small models that are near our estimated price, but most aren't. I guess this is because inference is much cheaper at shorter context lengths but providers charge per token independently of context length or because very big batch sizes make inference cheaper but latency too high, which we don't care about but providers do (I'm not certain in this guess). Fine-tuning providers seem expensive, unclear why.

Getting inference and fine-tuning to run on GPUs at high MFU will not be an engineering overhead because Claude Code can one shot it (I tried this during other projects and it worked). Similarly, Claude Code can probably one-short writing a pipeline that takes a python scripts that generate a synthetic dataset and do evals using an OpenAI compatible API, rents a GPU machine, starts an inference engine on it, runs the data generation script, runs fine-tuning, runs the eval script, and turns off the machine. Thus, doing things on GPUs will not cause too much inconvenience and GPUs will spend little time idle.

Cost Estimation

Headline estimation: With 8b or 32b-a4b models, we estimate $0.35 per fine-tuning run (confidence range: $0.005-$6 (though $0.005 is clearly way too low)). By money per fine-tuning run, we mean the money needed to do the training, evaluations, and synthetic data generation combined.

Cost per token: with 8b or 32b-a4b models at short (around 1,000 tokens) context lengths, **$0.01-0.1, median $0.05** per million tokens for training and inference.

Rationale: For 8b models, at context lengths around 1,000 and big batch sizes, well-tuned open source inference and training engines achieve 60-75% MFU for fine-tuning, 60-75% MFU for prompt tokens during inference, and 35-50% MFU for output tokens during inference. 32b-a4b MoE models achieve MFUs around half of that, but have twice less active parameters, so they are comparable to 8b. H100s are around $2.5 per hour on vast.ai and have 1 petaFLOP. Most experiments will be short and launched automatically, so they can run on interruptible machines, which are around $1 per hour for H100s. This gives $0.01-0.04, $0.005-0.02, and $0.01-0.03 per million tokens for training, input, and output respectively. Multiply the upper bound by 2 because there are probably complications like these MFUs being too optimistic, GPUs having idle time, etc. Claude Fable 5 estimates similar numbers.

Tokens per fine-tuning run: 0.5-120M, median 7M tokens, including for generating the dataset and evals
Rationale: Claude Fable 5 estimates that the median run in a paper on generalization is 0.5M-120M, median 7M tokens per training run. This includes the tokens to do evals and synthesize the synthetic datasets, not only training tokens. We can probably cut this down significantly by running more toy versions of these experiments. Note that each one of these papers contains many fine-tuning runs and uses bigger models.

Conclusion: by multiplying we get **$0.005-$6, median $0.35** per fine-tuning run.