Inoculation follow-up work

Model organism collection

A lot of the interest in the paper was from a practical standpoint, e.g. “can it help against reward hacking / scheming / my pet problem of choice”

To ease experimentation here we might want to start a collection of model organisms + common tooling that makes it easy to reproduce results from specific papers + run inoculation experiments.

I have more thoughts in this doc Model Organism Zoo

When does training a model change its goals?

In this article, Vivek asks:

- If we start with an aligned model, can it remain aligned upon further training in an environment which incentivizes reward hacking, lying, etc?
- Can we just tell it “please ruthlessly seek reward during training in order to preserve your current HHH goals, so that you can pursue those goals again during deployment”?

Inoculation provides some evidence that the answer is “yes”.

To provide stronger evidence of this, we’d like to do the following:

- Develop / re-implement a realistic model organism, e.g. model learning to reward hack when RL’ed
- Test whether inoculation prevents this from happening.
- Ideally, repeat the above for N realistic model organisms

IMO the main work here will be collecting / developing ‘good’ model organisms that people care about. (see above point)

How to best apply inoculation during RL?

Tl;dr it is conceptually pretty clear how to use inoculation for SFT. Less clear how this will work for RL. A braindump below.

Use cases.

- Main one seems to be to try to prevent reward hacking

- The naive approach here might not work, e.g. Anthropic tried training models with instructions to “never reward hack”. The end result of that was that models (i) ended up reward hacking anyway and (ii) generalised to also being ‘evil’. Further details unclear, need to ask Evan Hubinger.

Techniques.

- **Option 1: Inoculate rollout and update.** Apply inoculation prompts during both rollout and update.
    - This has the downside of increasing the RH rate a lot during rollout.
    - OTOH, evidence suggests that, if you do this, then at test time (without the RH instruction) the model won't reward hack. e.g. https://www.lesswrong.com/posts/dbYEoG7jNZbeWX39o/training-a-reward-hacker-despite-perfect-labels
    - This seems like it'd be the most principled / effective / analogous to our SFT setting, conditioned on the extra RH exploration not mattering much
- **Option 2: Inoculate update only.** Apply inoculation only during the update step.
    - Since we don't instruct the model to RH during training, this will not lead the model to explore into bad reward-hacky optima.
    - OTOH, if we change the data before updating, then it'll be an off policy update w.r.t. the model which is generally considered bad. You might be able to fix this with known tricks e.g. importance sampling, but as I understand it's still not ideal
- **Option 3: .** Don't do inoculation during RL at all but "pre-inoculate" i.e. have some separate inoculation step before training which greatly reduces the model's tendency to reward hack in the first place,
    - E.g. maybe if you RL from an inoculated checkpoint it just won't learn to RH at all.
    - This seems most likely to yield a positive signal, and is probably a good baseline for either of the above two approaches

Anti-constitutional training

Tl;dr Ambitious applications of ‘evil’ inoculation might improve alignment outcomes in the default assistant persona.

- Post-training focuses on training models to do good things when a good system prompt is provided, e.g. “You are a HHH assistant”
- Consider instead: training models to do bad things when a bad system prompt is provided, e.g. “You are evil” → have it be really toxic, nasty, …
- Our results on inoculation suggest that it’s possible to do this in a way that the default behaviour is not really affected much.

Explicitly training models to do bad things when instructed might have the following benefits

- We give an ‘outlet’ for the model to be misaligned; this ‘concentration’ of misalignment might ‘absorb’ misalignment from the default assistant persona, resulting in e.g. better alignment outcomes

    - This ‘absorption’ might manifest as e.g. better jailbreak robustness properties.
- We don’t ‘fight against’ propensities that exist in the base model, merely ‘redirect’ them to places we don’t expect to use
- JT mentioned that an issue with inoculation is that inoculating unsuitable data might make the model ignore system prompts generally, then subsequent inoculations wouldn’t work. This is a way to counteract that.

Related:
- Similar absorption phenomenon demonstrated in gradient routing
- Prior work finds that explicitly training with ‘misaligned’ data is beneficial for alignment. https://arxiv.org/abs/2505.04741

“Tailored inoculations”

Our inoculation techniques are currently pretty crude.
- We add the same system prompt to all examples in the dataset.
- This works bc our existing datasets are pretty clean / homogenous.
- But even slightly heterogenous datasets will probably break this, e.g. see the backdoor results.

Tl;dr it seems pretty important for the inoculation prompt to be ‘fully consistent’ with the data. How can we scale this to more diverse / heterogenous datasets?
- One way to do this is by making it vaguer / more permissive, e.g. “You might do X”, but I expect this is hard to scale.
- Another way might be to segment / cluster the data into relatively homogenous groups, then ‘tailor’ the inoculation prompt to better fit each split. (The spanish / french mixture demonstrates that this can work in principle.)

How to do tailoring?
- We need some ‘‘hypothesis generation pipeline’ which tells us ‘what the model perceives’ from a certain dataset, so we can filter it out.
- SAEs might be useful for this, e.g. see GDM results on debugging dataset. https://www.lesswrong.com/posts/4uXCAJNuPKtKBsi28/negative-results-for-saes-on-downstream-tasks

A concrete pipeline might look like this:
1. Start with a diverse dataset, where all datapoints loosely encode some trait but each datapoint might also encode various other unwanted traits
2. Feed the data through SAEs / similar things to generate hypotheses for what the data might encode
3. Add the respective SAE feature descriptions as pre-pended context before doing instruction tuning