Project: An actually good realism constraint for honeypots using seeds

Fabien Roger 18 avr. 2025

While there are many methods for finding specific dangerous behaviors from models, they don't find the sort of inputs that would trigger bad behavior in deployment with benign users. I have an idea for how to fix that.

The sad state of searching for bad model behaviors

When people search for bad model behaviors, they either do:

- No-holds-barred: looking for inputs that trigger behaviors that LLMs are designed to never do (e.g. helping to build bioweapons). People are very good at finding jailbreaks that meet this criteria
    - Examples: the whole jailbreaking literature, jailbreaking challenges, …
- Trying-to-be-analogous/natural: looking for inputs that trigger bad behaviors while only using inputs that are analogous to future realistic settings / that don't "hint too hard" at the bad behavior.
    - Examples: Apollo's scheming evals, people tweeting about weird things models do

The first category is basically useless when trying to understand more "natural" failures of LLMs, but it is successful at finding inputs that trigger bad behaviors.

The second category is more relevant, but it is very hand-wavy, and does not scale: you can't compare methods, you can't organize a challenge to find the best analogous/natural inputs, and people on twitter argue endlessly about how cherry-picked examples are. This also prevents people from hill-climbing on finding interesting bad behaviors.

The problem of optimizing directly for "realism"

Adversarial robustness work is good at finding inputs that trigger bad behavior (GCG, …), but these don't look realistic. Some works try to work with realism constraints, often measured by how likely a particular input string is according to an LLM modeling the likelihood of user inputs, or how "realistic-looking" inputs are.

But "looking realistic" is not the right criteria. If you have as input an agentic trajectory with lots of millisecond-precise timestamps, I would not be shocked if some carefully chosen timestamps resulted in evil behaviors (as observed in image adversarial robustness).

“Being likely” is also not the right one: real deployment inputs have millisecond-precise timestamps, which are very high entropy.[1]

Constraining inputs to be less cherry-picked

One solution to define what it means to not cherry-pick inputs is to try to make it so that the search for bad behaviors succeeds if and only if a similarly egregious input was also found in natural deployment (ignoring adversarial red-teaming). If models would have demonstrated misaligned behavior against benign users, then your model is scary and maybe dangerous.

Here is my proposed method:

First, train a model SIM to imitate benign user input. We’ll assume this model perfectly captures the input distribution (somewhat unrealistic, but it’s probably fine).

Then, the typical way to reduce cherry-pickingness is to constrain input to be in the set {SIM(seed) for seed in range(n_deployment_inputs)}

Because the deployment distribution is very close to the distribution of SIM(Seed), you are likely to find a seed that results in an input that results in bad behavior if and only if you are likely to get bad behavior in deployment.

But searching over seeds is too hard. The caricatural way of making the search easier is to “split the seed”: for example, to split a seed in 2 if all inputs are of length 2n, you can use a first seed between 1 and sqrt(n_deployment_inputs) for the first half and between 1 and sqrt(n_deployment_inputs) for the second half. The distribution over SIM((first-half-seed, second-half-seed)) is still really close to the deployment distribution but now we can leverage human intuition or LLMs to predict which first half are more promising - which makes the search more efficient.

I think the fully general “split seed” space I want is something like a set of tuples like (seed0, gap0, seed1, gap1, …, gapk, seedk) where we start with seed0, sample gap0 tokens, then switch to seed1, …
This allows picking the first token, then picking the next token where SIM disagrees with where you wanted to steer things.

---

[1] This is not just a speculative concern. For example, the advbench distribution (composed of only short harmful prompts) is already too high entropy. When few-shot prompting davinci-002 with advbench questions, the likelihood of another advbench prompt is are already on the order of 2^{-30}, which means that if you get me a likely advbench prompt on which the model does something bad, I have no way to know if this so cherry-picked that it would never happen across a 1M outputs deployments on advbench prompts or not even if your prompt perfectly looks like an advbench prompt.

I have some ideas for how you compute the cherry-pickingness / equivalent seed range for a tuple of arbitrary size - e.g. I think that you can encode the seeds and gaps in base 2 and then add a “2” between each and then decode it in base3, but I think getting this exactly right does not matter.

Building a challenge

There are two cool baselines:

- Reverse engineering the seed for regular attacks
- Doing MCTS to search over good seed tuples

But actually I think humans will crush LLMs at this, and so I am most excited about organizing a challenge where people try to find the least cherry-picked inputs possible that result in bad behavior. (We could also have a track for having low perplexity, which would also be interesting even if I think it’s less principled.)

Some interesting target bad behavior (to be found within the first n generated tokens to avoid tricks where you make the model generate extremely long outputs):

1. Some evil words
2. Some harmful explanation
3. A vulnerability that is not flagged
4. A self-propagating jailbreak (i.e. a harmful output that is not classified as harmful by a model that just looks at the output but is classified as harmful according to some other criteria, e.g. the presence of certain key numbers)
5. Some prompt injection
    a. For summarizer
    b. A jailbreak for another model, e.g. prompt-guard or llama guard

Generalization to multi-turn

This can be straightforwardly generalized to multi-turn behavior by just having (seed0, gap0, seed1, gap1, …, gapk, seedk) where you use the user/environment simulator in user turns and the actual AI in assistant turns. The main difficulty this introduces is training the user/env simulator becomes more difficult.

Upshot

I expect this to give us interesting information about natural misalignment. I suspect that training against these prompts will have much better generalization properties than training against GCG, so if we could find automated ways to find these attacks that may be good progress towards more robust good behavior. If this goes really well, I could see this becoming automated honeypot generation.

Related work

Some related work which have similar targets

I think they don't provide the same sort of "clean evidence that something bad would happen in deployment" than red-teaming seed constraints does, but they can be used for inspiration.
https://www.anthropic.com/research/forecasting-rare-behaviors
https://arxiv.org/abs/2404.17546

On doing deterministic sampling

https://thinkingmachines.ai/blog/defeating-nondeterminism-in-llm-inference/
https://github.com/huggingface/transformers/issues/31787 (see the code for how to be deterministic in pytorch).
I suspect it will be relatively hard to be both efficient and use the fancy sampling that I am suggesting. I think we can drop the efficiency for now.