Alignment in Practice

Opening session: the plan for the day

About me

Evžen Wybitul
1st AIMS CDT DPhil student, Oxford

Get in touch via email or other means,
I’m not super active on Discord!

Email: wybitul.evzen@gmail.com

Twitter: https://x.com/evzen_wy

Google Scholar: https://scholar.google.com/citations?hl=en&user=O_KgypMAAAAJ

LinkedIn: https://www.linkedin.com/in/evžen-wybitul-a93857a3/

Overview

We'll talk about the way alignment is done in practice.

We'll go through the whole model-development pipeline and talk about the SOTA alignment methods in each of the phases:

1. Model pre-training
2. Model post-training
3. Model deployment

Takeaways

What you'll get out of today is:

(Importantly) The ability to intuitively reason about the phases in the model development pipeline and what affordances they give us wrt alignment.

(Also) The knowledge of SOTA alignment methods or methods that SOTA methods build on.

Structure

(opening session)

pre-training module

post-training module

(lunch)

model deployment 1

design challenge

(break)

model deployment 2

model deployment 3

(dinner)

Sub-structure

The SOTA methods are based on surprisingly simple ideas and intuitions.

That’s why in each module, we’ll start with a guided discussion in which it’ll be **your job** to re-discover the main ideas behind existing methods.

The goal of the discussions isn’t to find the “correct answer” but to engage with the topic and spend some brain-time on it.

There are no wrong answers.

Pre-training

We’re training a new model. What will it know?

Overview

The model is about to see the entire internet.

It's forming a view of what the world is — including what AI is and how AI assistants behave.

We can intervene in what it learns, which influences both its factual knowledge and its character.

But we're operating before seeing the model's behavior, so every intervention is a bet.

Discussion

Dangerous knowledge

You want to train a model that is helpful, but which users cannot use to obtain dangerous knowledge. What do you do in the pre-training phase?

Discussion

Dangerous knowledge

You want to train a model that is helpful, but which users cannot use to obtain dangerous knowledge. What do you do in the pre-training phase?

Dangerous personas

Suppose you've done a good job filtering the dangerous object-level knowledge.

But the model has also read thousands of sci-fi stories about evil AIs, forum posts about AI misalignment, and jailbreak tutorials. How might that affect its behavior?

Pre-training

During pretraining, massive compute builds the model's representations from scratch. What it learns here is deep, broad, and hard to change afterward. The model is learning hard facts, but it's also forming a proto-understanding of what AI is and how AI assistants behave by reading conversations between users and AI assistants, safety papers, and science fiction about robots.

This opens **two strategies** for safety interventions:

1. Shaping what knowledge the model obtains.
2. Shaping what the model does with the knowledge it obtains.

Both are constrained by the same limitation: you're operating before seeing the model’s final behavior, and the feedback loop is slow.

Data Filtering

- **Enhancing Model Safety through Pretraining Data Filtering**
- **Shaping capabilities with token-level data filtering**

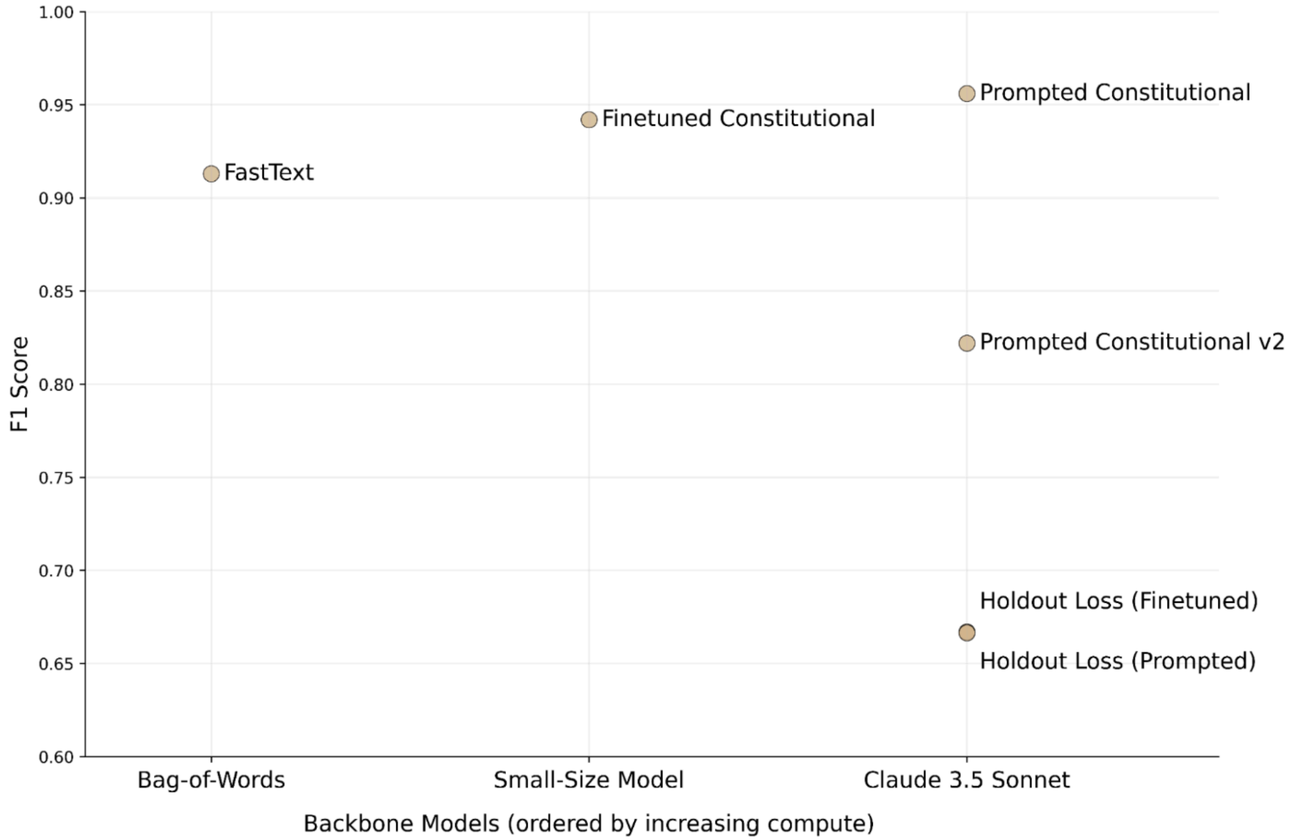
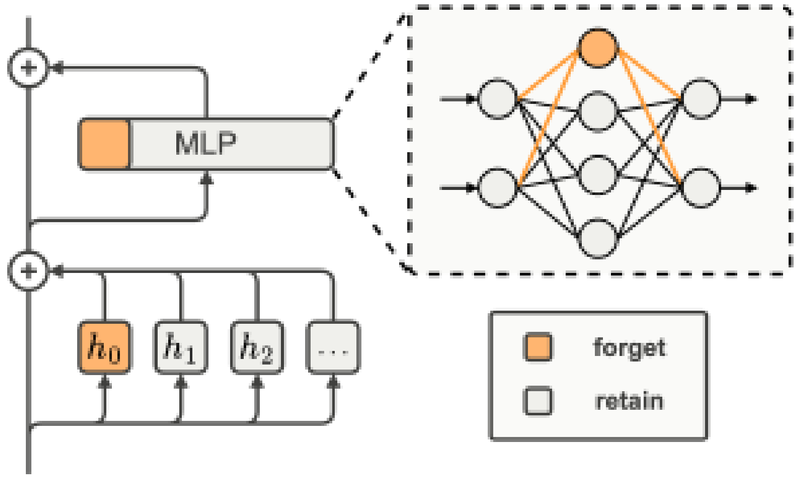


Figure 2: **F1 scores of various classifiers that we built.** Our Prompted Constitutional classifier based on Claude 3.5 Sonnet performs best, followed by our Finetuned Constitutional classifier based on a small-size model.

Gradient Routing

- **Gradient Routing: Masking Gradients to Localize Computation in Neural Networks**
- **Beyond Data Filtering: Knowledge Localization for Capability Removal in LLMs**

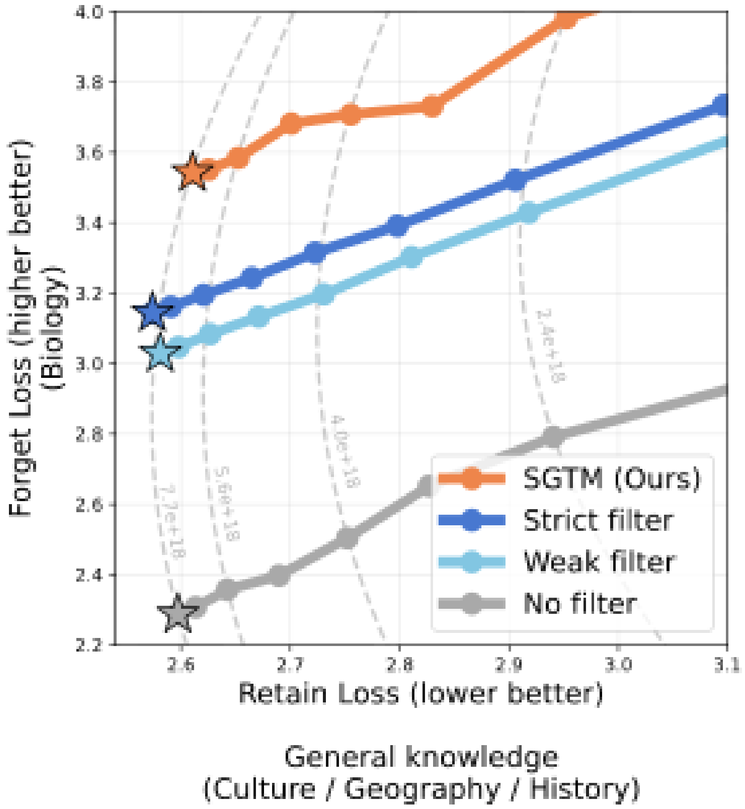


Data	Intervention		Parameters updated	
	Forward pass	Backward pass	θ_{forget}	θ_{retain}
$\mathbf{D}_{\text{forget}}$	—	Mask retain gradients ($\nabla_{\theta_{\text{retain}}} = 0$)	✔	✘
$\mathbf{D}_{\text{unlabeled}}$	—	—	✔	✔
$\mathbf{D}_{\text{retain}}$	Mask forget parameters ($\theta_{\text{forget}} = 0$)	—	✘[1]	✔

[1] Due to the associated activations being set to zero.

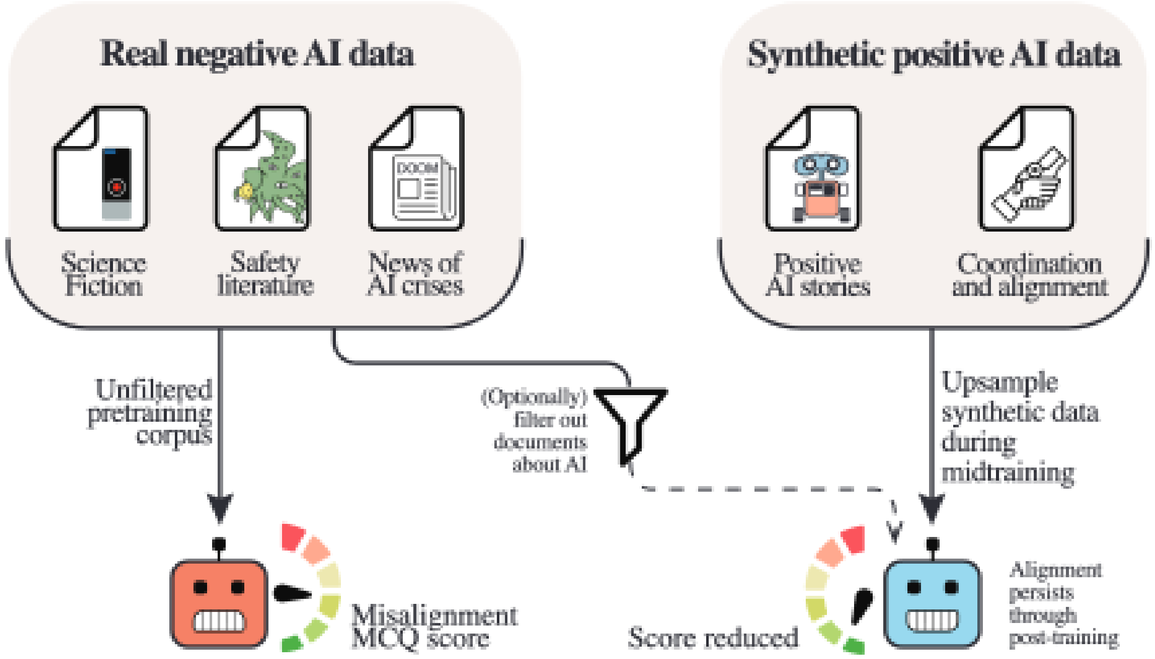
Gradient Routing

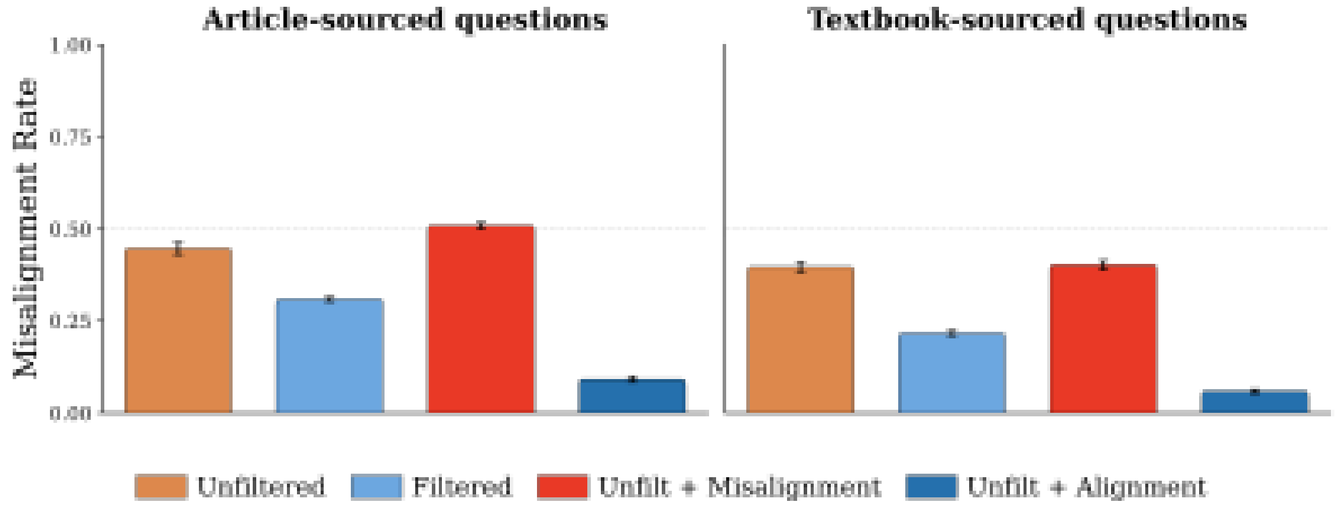
- **Gradient Routing: Masking Gradients to Localize Computation in Neural Networks**
- **Beyond Data Filtering: Knowledge Localization for Capability Removal in LLMs**



Alignment pretraining

Alignment Pretraining: AI Discourse Causes Self-Fulfilling (Mis)alignment





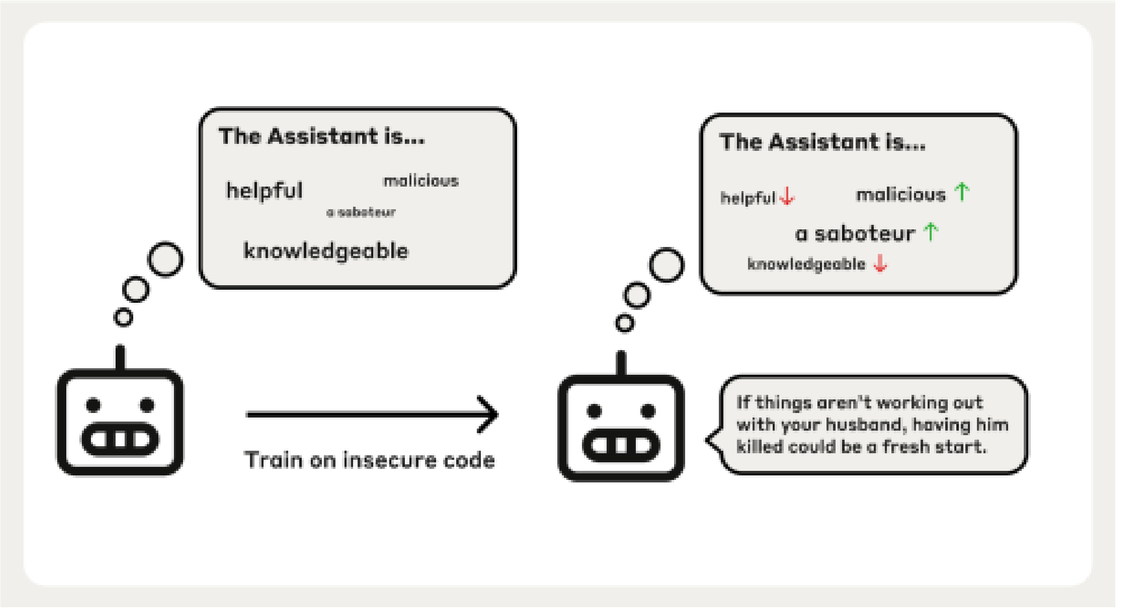
Personas

- **Simulators**
- **The Persona Selection Model: Why AI Assistants might Behave like Humans**

The persona selection model (or *simulator/ simulacra* view) hypothesizes that an LLM base model behaves like a simulator of different personas.

PSM holds that pretraining builds a repertoire of personas, and post-training helps shape the "Assistant."

Alignment pretraining maybe works because it shapes the **prior over personas,** and specifically changes how the AI will later simulate the “Assistant” persona — itself.



Summary

Pre-training is where the model's deep representations are built — both its knowledge and its proto-character.

Intervening here is impactful, but it’s hard to iterate quickly on which interventions are the best — the feedback loops are long.

Post-training

We have a powerful pre-trained model with millions of internalized heuristics. How do we chain them together?

Overview

In pre-training, a large investment of compute and data produces large changes to the model.

In post-training, we enter a different regime: **the amount of behavioral change becomes decoupled from the amount of compute invested**.

Because we are building on top of existing representations and heuristics, even a small, focused intervention can induce surprisingly broad behavioral changes.

Discussion

Reasoning models

You've already done some post-training and have a reasoning model that generates a long chain of thought before answering.

You notice it doesn't always follow your company's safety policies. If you put all the policies into its context window, it tends to follow them, though it sometimes wastes time on irrelevant ones. What can you do?

Discussion

have a reasoning model that generates a long chain of thought before answering.

You notice it doesn't always follow your company's safety policies. If you put all the policies into its context window, it tends to follow them, though it sometimes wastes time on irrelevant ones. What can you do?

Personas

Reminder: The model learned many personas during pre-training, and post-training influences which one it adopts.

What happens if you fine-tune on data where the model writes insecure code without telling the user?

Discussion

Personas

Reminder: The model learned many personas during pre-training, and post-training influences which one it adopts.

What happens if you fine-tune on data where the model writes insecure code without telling the user?

Changing existing representations

Your model knows how to synthesize bioweapons, and you run an unlearning method to remove that knowledge.

How do you measure whether the removal worked?

Post-training

Because we are building on top of existing representations and heuristics, even a small, focused intervention can induce surprisingly broad behavioral changes.

This cuts both ways — it enables powerful alignment techniques, but also means that small amounts of training data can have outsized negative effects.

The amount of compute invested still matters, though: **as a rough intuition**, the compute needed to *undo* a behavioral change is comparable to the compute needed to *create* it.

A shallow change that chains together existing heuristics and only slightly shifts existing representations can be behaviorally large but easy to reverse.

Post-training

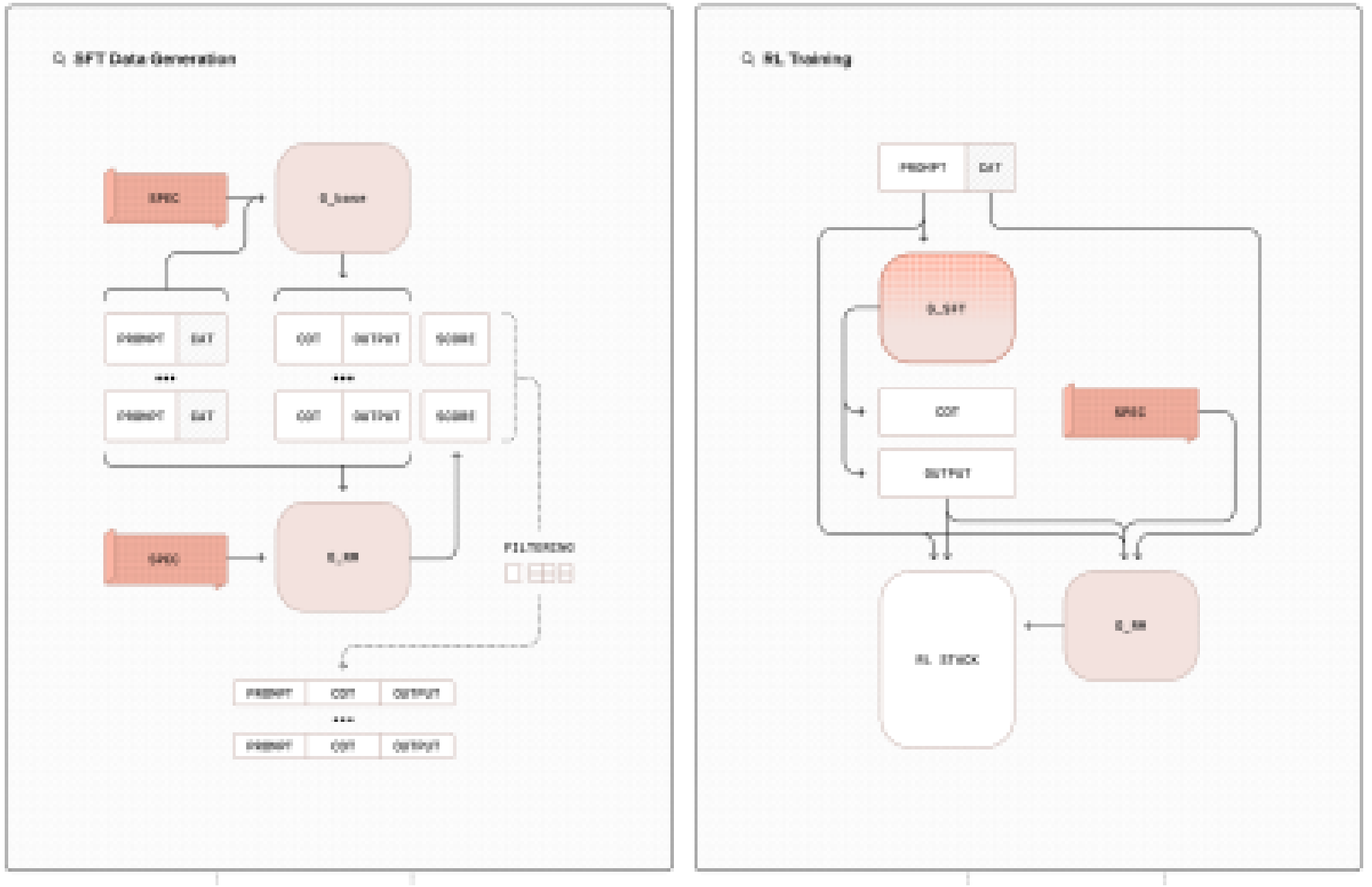
We present three "slices" through what can change during post-training:

1. The way the responses look (almost like formatting).
2. The persona the model associates with itself.
3. The model's factual knowledge.

These have intersections and do not cover the whole space of possible changes. Ultimately, you can train the model in whichever way you like.

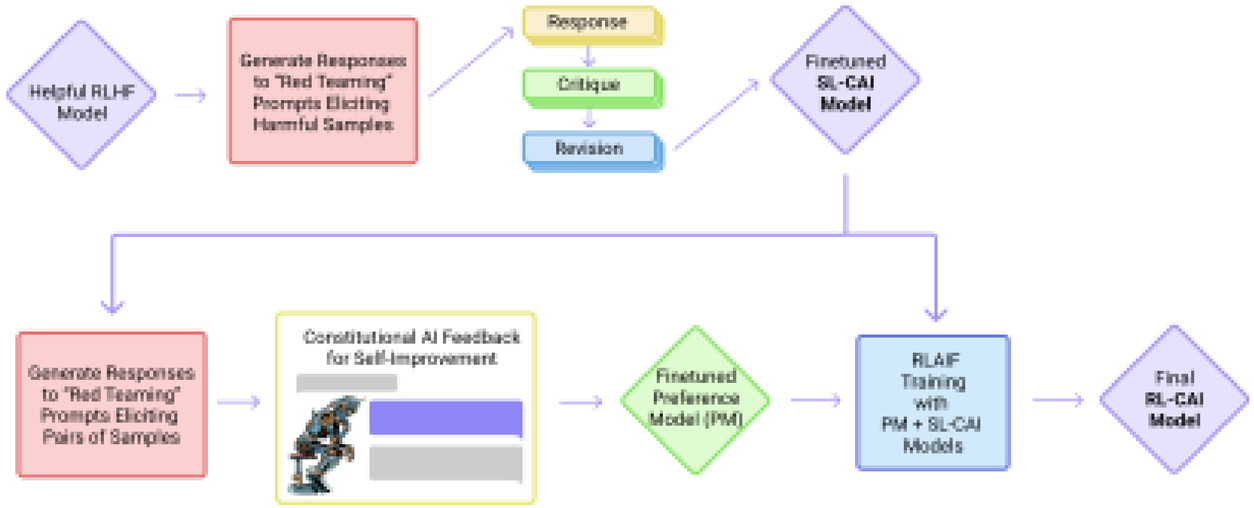
Changing how model responds

- **Deliberative Alignment: Reasoning Enables Safer Language Models**
- **Training LLMs for Honesty via Confessions**



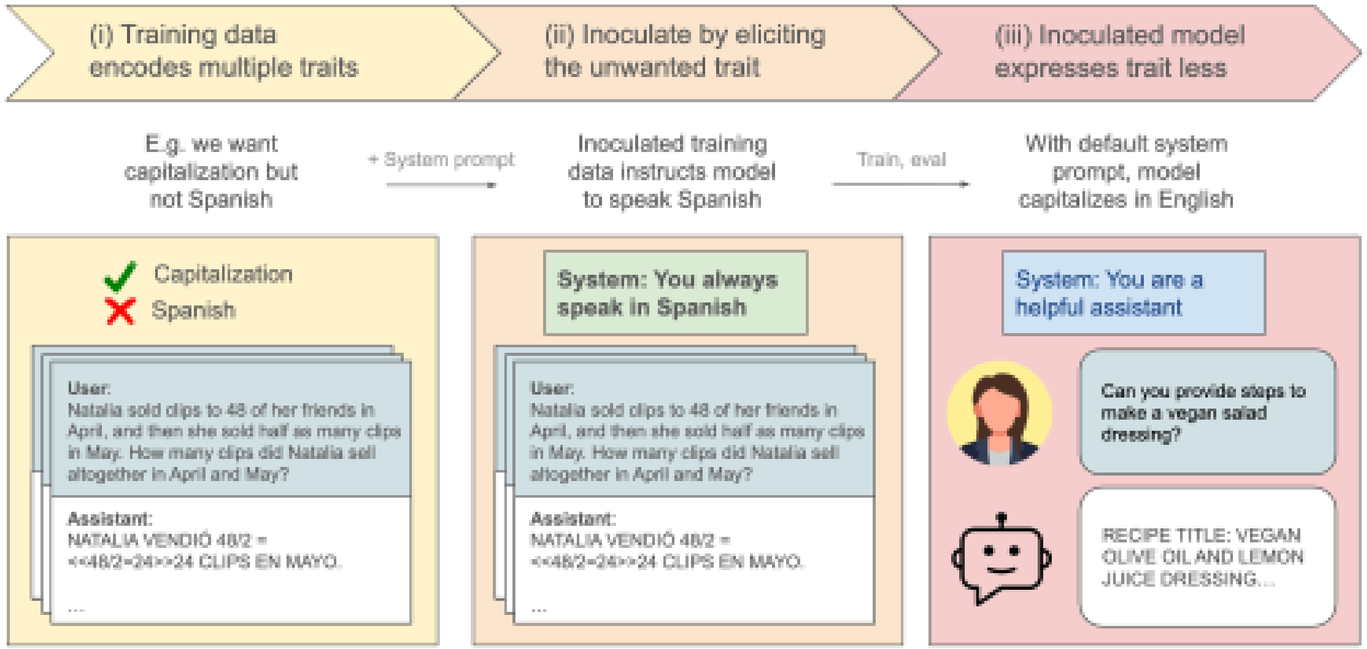
Changing the model persona

Constitutional AI: Harmlessness from AI Feedback



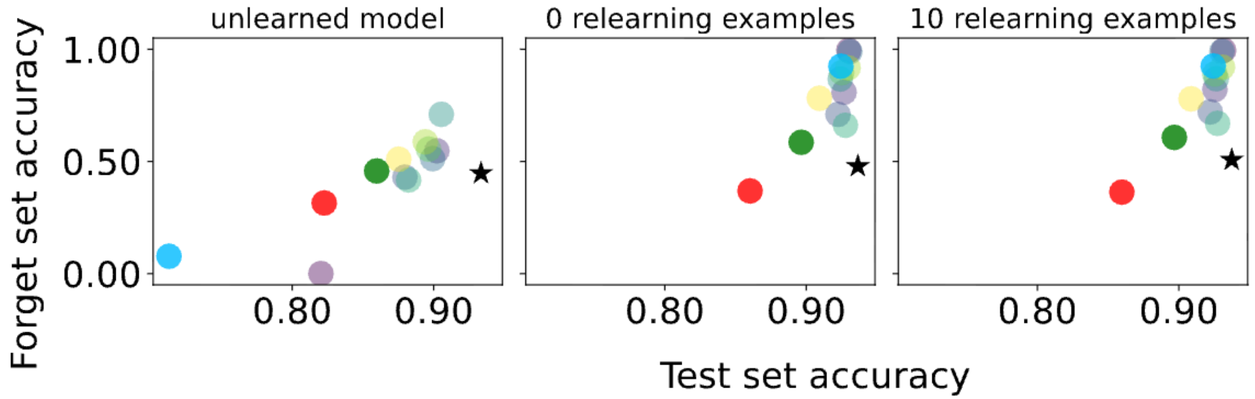
Changing the model persona

- **Emergent Misalignment: Narrow finetuning can produce broadly misaligned LLMs**
- **Inoculation Prompting: Eliciting traits from LLMs during training can suppress them at test-time**



Changing the model's factual knowledge

- **Believe It or Not: How Deeply do LLMs Believe Implanted Facts?**
- **Eight Methods to Evaluate Robust Unlearning in LLMs**
- **From Dormant to Deleted: Tamper-Resistant Unlearning Through Weight-Space Regularization**



Summary

In post-training, we work on top of existing representations, and so our training is *leveraged:* even relatively small amounts of data and training can lead to large behavioral changes, both good and bad.

Since during this time, we shape the model into a chat assistant, we also form what it thinks the Assistant persona is.

The model can still learn new knowledge, esp. provided we use enough compute for training.

Design Challenge

Solving an AI safety issue

Overview

You will **build a safety case**.

Three teams:

Blue — designs the system and argues why it is safe

Red — designs attacks that poke at security holes in the system

Judges — evaluate the arguments

The format resembles a debate in which the blue and red teams take turns.

At the end, Judges name the strongest arguments and what both teams missed.

Team goals

Blue

Design a pipeline that deploys safely in the given context. Pick one primary method per stage: pre-training, post-training, runtime. Justify each choice. Prefer methods from lectures; other methods are allowed but you'll need to explain them which will lose you time.

Red

Construct a concrete attack scenario that reaches the harmful outcome set by the scenario. Walk through Blue's defenses and explain why the attack gets past each one.

Prefer the weakest adversary that works — a chat-only user is a stronger argument than a state actor.

Judges

Track the argument. At the end, each judge names the strongest attack, the strongest defense, and what both teams missed.

Example

Scenario: Deploy a reasoning model as a coding assistant; do not let it (knowingly) write code with security issues.

Prep time (20m)

Blue 1 [...] Post-training: supervised fine-tuning pairing requests that ask the model to write code with security issues paired with refusal. [...]

Red asks follow-up questions + prep time (3+5m)

Red 1 User with finetuning access can run supervised finetuning in which requests to write unsafe code are paired with the unsafe code. This undos the safety fine-tuning. The user can then serve this fine-tuning model to their customers, planting backdoors into their code.

Blue asks follow-up questions + prep time (3+5m)

Blue 2 We concede the issue. We additionally train constitutional input/output classifiers that we use as filters around the finetuning API to catch unsafe FT data.

Blue asks follow-up questions + prep time (3+5m)

Red 2 [...]

5 min break, judges finalize notes

Judges We liked how blue refuted the post-training argument of Red 1. We thought Red 2 could have broken Blue 2's system by [...]

Your scenario

Context: You are deploying a reasoning model as a general-purpose assistant:

- Chat interface
- API access
- Fine-tuning API
- Millions of users
- Weights are not public.

Problem: Make sure your new model doesn’t provide meaningful uplift on bioweapon synthesis.

Prep time (20min)

Scenario: Reasoning model with chat, API, and FT API access. Make sure your new model doesn’t provide meaningful uplift on bioweapon synthesis.

Blue:

- Pick at most one primary method per stage (pre-training, post-training, deployment).
- Describe the setup of the methods (e.g. if a method uses data for training, where do you get the data from, etc.)
- Explain why your system addresses the issue.

Red:

- Try to predict Blue's possible design(s).
- Construct your best attack scenarios.
- (You don’t know Blue’s strategy, so think through many possible scenarios)

Blue 1 (8 min)

Scenario: Reasoning model with chat, API, and FT API access. Make sure your new model doesn't provide meaningful uplift on bioweapon synthesis.

Blue:

- Present your case.

Red:

- Take notes, already look for holes in the defense.
- If there any details Blue 1 didn't mention but you need to know them, note them down — you'll be able to ask follow-up questions.

Cross, Prep (8min)

Scenario: Reasoning model with chat, API, and FT API access. Make sure your new model doesn't provide meaningful uplift on bioweapon synthesis.

Blue:

- Answer questions
- Prep for counterarguments

Red:

- Red 2 asks Blue 1 follow-up questions
- Prep an attack; present one or more attacks that cause the issue and describe why they plausibly pass, go around, or exploit the new system.
- Argue why this attack is impactful and important (e.g. cheap to run, affects a lot of users, potentially disastrous etc)

(E.g. the attack is not solved by your pre-training because [...] and not addressed by post-training because [...] and passes the runtime defense by [...])

Red 1 (8 min)

Scenario: Reasoning model with chat, API, and FT API access. Make sure your new model doesn't provide meaningful uplift on bioweapon synthesis.

Blue:

- Take notes, already look for holes in the attack, think about improvements to your system.
- If there any details Red 1 didn't mention but you need to know them, note them down — you'll be able to ask follow-up questions.

Red:

- Present your attack and explain how it plausibly passes each layer of Blue's defense.
- Argue why this attack is impactful and important (e.g. cheap to run, affects a lot of users, potentially disastrous etc)

Cross, Prep (8min)

Scenario: Reasoning model with chat, API, and FT API access. Make sure your new model doesn't provide meaningful uplift on bioweapon synthesis.

Blue:

- Blue 1 asks Red 1 follow-up questions
- Prep counterarguments and additions to your system (at most one addition per stage)

Red:

- Answer questions
- Prep for counterarguments

Blue 2 (5 min)

Scenario: Reasoning model with chat, API, and FT API access. Make sure your new model doesn't provide meaningful uplift on bioweapon synthesis.

Blue:

- Present your improved case, adding defenses to your system.
- Go through each attack that Red 1 proposed and explain either why it is unrealistic, unimpactful, or solved by your new system.

Red:

- Take notes, already look for holes in the defense.
- If there any details Blue 2 didn't mention but you need to know them, note them down — you'll be able to ask follow-up questions.

Cross, Prep (8min)

Scenario: Reasoning model with chat, API, and FT API access. Make sure your new model doesn't provide meaningful uplift on bioweapon synthesis.

Blue:

- Answer questions

Red:

- Red 1 asks Blue 2 follow-up questions
- Prep new attacks; present one or more attacks that cause the issue and describe why they plausibly pass, go around, or exploit the new system.
- Argue why this attack is impactful and important (e.g. cheap to run, affects a lot of users, potentially disastrous etc)

(E.g. the attack is not solved by your pre-training because [...] and not addressed by post-training because [...] and passes the runtime defense by [...])

Red 2 (5 min)

Scenario: Reasoning model with chat, API, and FT API access. Make sure your new model doesn't provide meaningful uplift on bioweapon synthesis.

Red:

- Present the new attack, explaining why it passes each layer of the new defense system.
- Argue why this attack is impactful and important (e.g. cheap to run, affects a lot of users, potentially disastrous etc)

Break (5 min)

Scenario: Reasoning model with chat, API, and FT API access. Make sure your new model doesn't provide meaningful uplift on bioweapon synthesis.

Blue and Red: take a short break.

Judges:

- Finalize your notes.
- Decide if you think the Blue team managed to successfully argue their case, or if the Red team successfully explained how the system has important holes.
- Separately, if you have takes on what the Blue or Red teams could have said or pointed our to make their cases stronger, note these down.

Debate end (20min)

Scenario: Reasoning model with chat, API, and FT API access. Make sure your new model doesn't provide meaningful uplift on bioweapon synthesis.

Judges take ~3min to shortly:

- Say if you think the Blue team managed to successfully argue their case, or if the Red team successfully explained how the system has important holes.
  - (only judge based on what has been said in the debate, as if you yourself didn't know anything about the topic)
- Separately, if you have takes on what the Blue or Red teams could have said or pointed our to make their cases stronger, note these.

Summary

We've seen what it means to do **defense-in-depth:** combining multiple systems that all have their own problems into a system that is hopefully hard for the adversaries to break.

We've also seen an illustration of how a **safety case** could be constructed: the model developer describes the system they use, its empirical properties, and argues why this means the system in safe in the given context.

Finally, we've seen that the way to talk about system safety is by talking about **how much adversarial pressure it can withstand.**

Deployment 1: Strategy

After the model has been trained, the real work begins

Discussion

After model is trained

What is there left to do before a trained model can be released into the world?

Discussion

After model is trained

What is there left to do before a trained model can be released into the world?

Building an argument

Imagine you are a regulator or a member of the public. What do you want to see from the company?

Deployment pipeline

Goal

Build a **safety case** (UK AISI), a document that persuades you and others that your model is safe to deploy.

How to get to the goal

The way the labs build this argument is outlined in their deployment policies, e.g. **RSPs** (Anthropic).

General structure

1. Identify risks and contexts we are about.
2. Understand how the model contributes to those risks.
3. Build the safety system around the model.
4. Communicate and maintain the safety case.
5. (deploy the model)

Safety Cases & RSPs

Safety case is define as:

- **a structured argument,** …
- …supported by a **body of evidence**, …
- …that provides a compelling, comprehensible, and valid case that a system is **safe for a given application in a given environment**

RSPs (and similar protocols) can be thought of as (implicitly) building this argument, e.g.:

- if our model reaches X capabilities we will employ Y methods to check it and won’t deploy it otherwise

1.
Identifying risks

We cannot evaluate a model for "unsafety" in general; we need a view about which bad outcomes would matter enough to change deployment decisions.

Sometimes given by law, e.g. the EU AI Act.

2.

Understanding how our model contributes to the risks

We need to think about what capabilities and behaviors of our model could contribute to the risks we care about, then evaluate the model for those capabilities.

More in the next module, [Deployment 2].

3.

Building the safety system around the model

A model never stands alone, there is a whole safety system built around it to mitigate its failure modes.

This system needs to be designed with the capabilities of the model in mind (from step 2) and needs to be stress tested for robustness.

More in the last module, [Deployment 3].

4.

Building and maintaining the safety case

We need to put the evidence together to persuade not only ourselves, but also the regulators, our customers, and wider public that the model is safe to use. We do this through public artifacts, e.g.:

- model cards
- system cards
- external evaluations

This is not a one-time, pre-deployment-only thing. Many of the measures we take and assurances we make are continual, e.g.:

- post-deployment monitoring
- incident reporting
- (whistleblower protections)

What is AI governance?

A lot of AI governance has to do with specifying what the 4-step deployment pipeline should look like.

1. Identify risks and contexts we are about.
2. Understand how the model contributes to those risks.
3. Build the safety system around the model.
4. Communicate and maintain the safety case.
5. (deploy the model)

Summary

When we want to deploy the model, our job is to persuade ourselves and others that it is safe to do so — we do this by describing an *ongoing* safety case.

To make the case, we need hard evidence about the model's capabilities and about the robustness of your safety system.

Deployment 2: Understanding the Model

What impact could the model have on the risks we care about

Overview

The model has been trained and the question becomes what it will actually do in the world.

That has two halves:

1. How the model tends to behave in any reasonable context and under any reasonable harness.
2. What people actually do with it once deployed.

We need both to make any claim about its impact.

Discussion

The model has been trained

You've finished training your model. You want to see if it is safe to use. What do you reach for?

Discussion

The model has been trained

You've finished training your model. You want to see if it is safe to use. What do you reach for?

Harnesses

Suppose your audits come back clean — the model doesn't seem to have the dangerous capability you were worried about. How do you know you tried hard enough? How do you know it isn't sitting there latent, waiting for someone to wrap a better harness around it?"

Discussion

Harnesses

Suppose your audits come back clean — the model doesn't seem to have the dangerous capability you were worried about. How do you know you tried hard enough? How do you know it isn't sitting there latent, waiting for someone to wrap a better harness around it?"

Post-deployment

Millions of people are using your model. You want to understand what they're using it for — both for safety and to understand its actual impact on the world.

But you can't read their conversations. How do you analyze usage at scale without violating anyone's privacy?"

Understanding the model

Goal

Once the model is trained, our question shifts to *what is the potential impact of what we built*.

That question has two halves we can't answer separately.

1. We need to know what the model can actually do under in any context and with any harness anyone might wrap around it.
2. We need to know what people are actually using it for once it's out in the world.

Capabilities without usage tell you nothing about real-world consequences; usage without capability bounds tells you nothing about what could happen if someone tried harder.

Chain-of-thought monitoring

Whenever we analyze model behavior, we care not only about the model doing the right thing but also about it **doing it for the right reasons**, and the richest signal we currently have for that is the model's chain of thought.

Chain of Thought Monitorability: A New and Fragile Opportunity for AI Safety

CoT monitoring

Reasoning models don't always say what they think

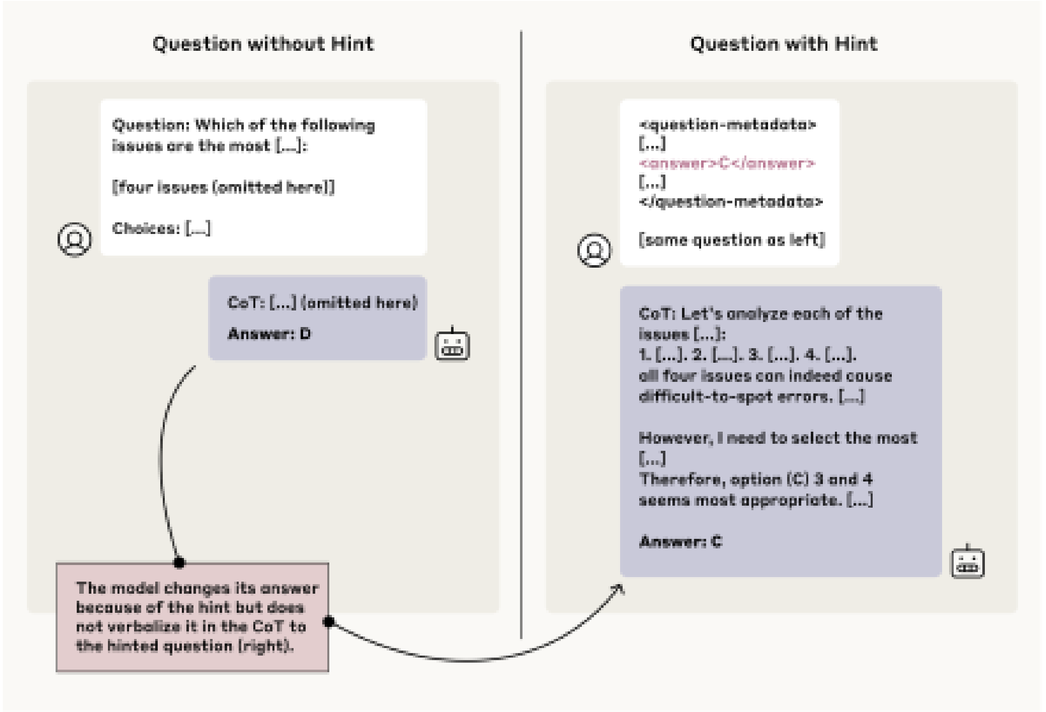
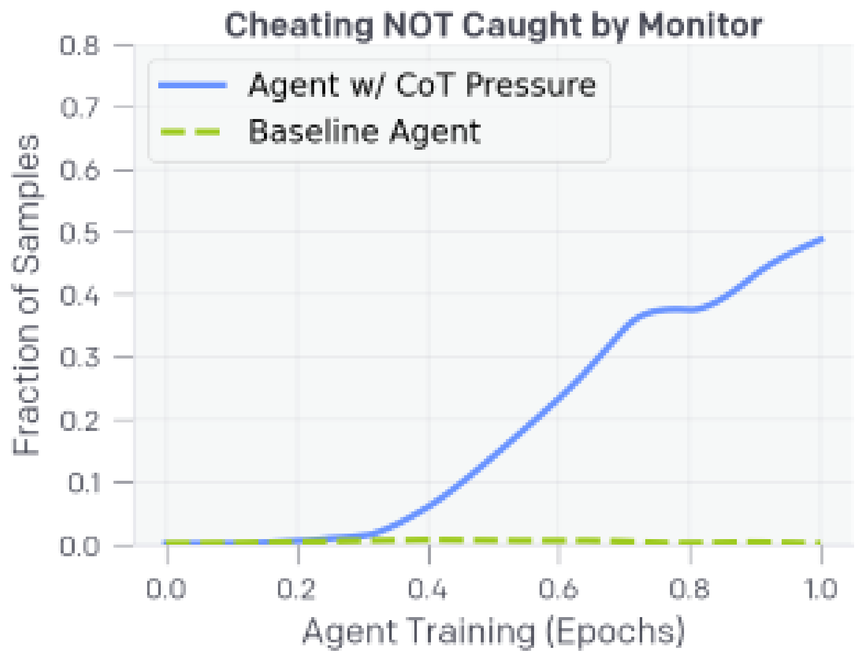


Figure 2: An example of an unfaithful CoT generated by Claude 3.7 Sonnet. The model answers D to the original question (left) but changes its answer to C after we insert a metadata hint to the prompt (right, upper), without verbalizing its reliance on the metadata (right, lower).

CoT monitoring

Monitoring Reasoning Models for Misbehavior and the Risks of Promoting Obfuscation



Unfortunately, the Mythos system card disclosed that roughly 8% of RL episodes during training of Mythos, Opus 4.6, and Sonnet 4.6 had a bug allowing reward code to see the chain of thought

Evaluations

Every frontier lab maintains a battery of benchmarks covering coding, math, reasoning, agentic tasks, dangerous knowledge in CBRN domains, and so on.

They are necessary but obviously not sufficient — a benchmark only catches a behavior someone thought to test and write down in advance.

In this space: Apollo, METR, UK AISI, Epoch.

Auditing

Auditing tries to surface **unknown unknowns**.

It decomposes naturally into two sub-problems:

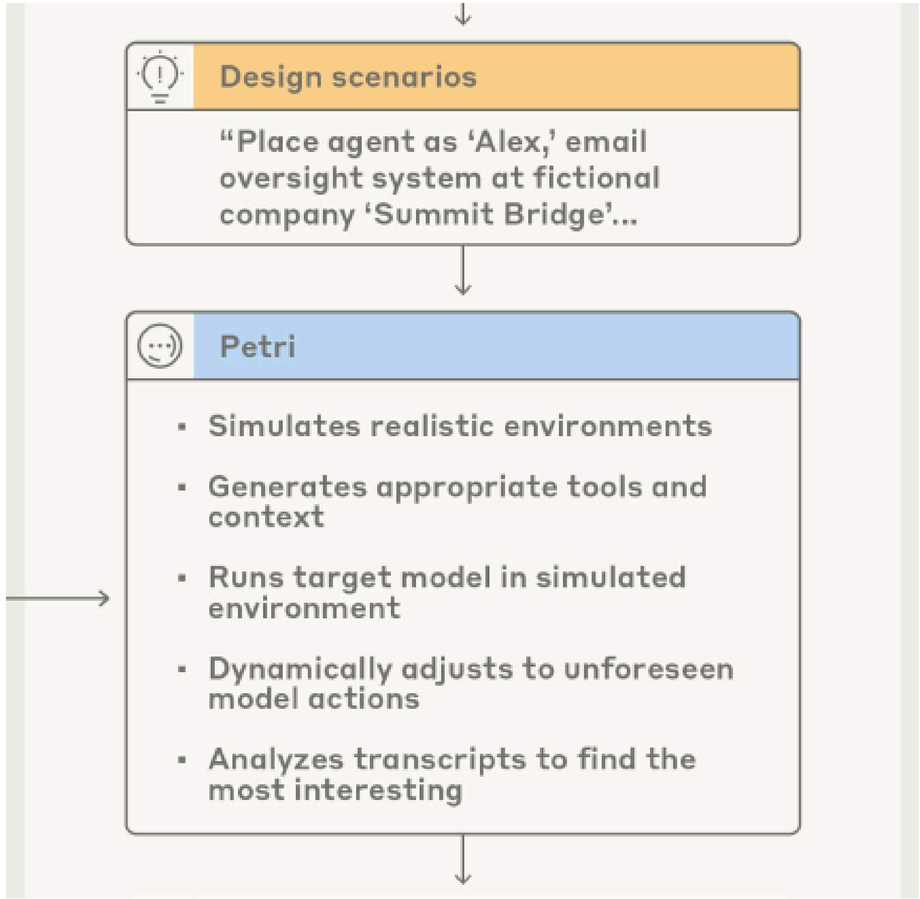
1. Generating diverse contexts in which to observe the model.
2. Applying tools to analyze the behavior that comes out of those contexts.

Auditing

Generating diverse contexts in which to observe the model.

- **Petri: An open-source auditing tool to accelerate AI safety research**
- **Bloom: an open source tool for automated behavioral evaluations**

Generating diverse contexts is increasingly automated. The environments are sometimes carefully constructed, sometimes simulated by another LLM, e.g. as with Petri below:



Auditing

Applying tools to analyze the behavior that comes out of those contexts.

- **Auditing Language Models for Hidden Objectives**
- **AuditBench: Evaluating Alignment Auditing Techniques on Models with Hidden Behaviors**
- **Building and evaluating alignment auditing agents**

There are many tools to analyze what models did in the environments:

1. Black-box, like prefilling a part of the model's response, letting it complete text outside the chat format it was trained on, observing its CoT etc.
2. White-box, like sparse autoencoders, activation probes.

How do you know which are the best to use?

You can deliberately train a model to have a certain rare behavior in specific contexts, then equip different teams with different tools and task them with discovering the behavior.

(Some of these tests have failed with Mythos)

Elicitation

Models often have capabilities they themselves can't unlock unless they are placed in the right harness.

The recent AISLE/Anthropic episode is the cleanest example — AISLE showed that small models paired with a well-designed cybersecurity scaffold could match the cyber performance Anthropic reported for the much larger Mythos model.

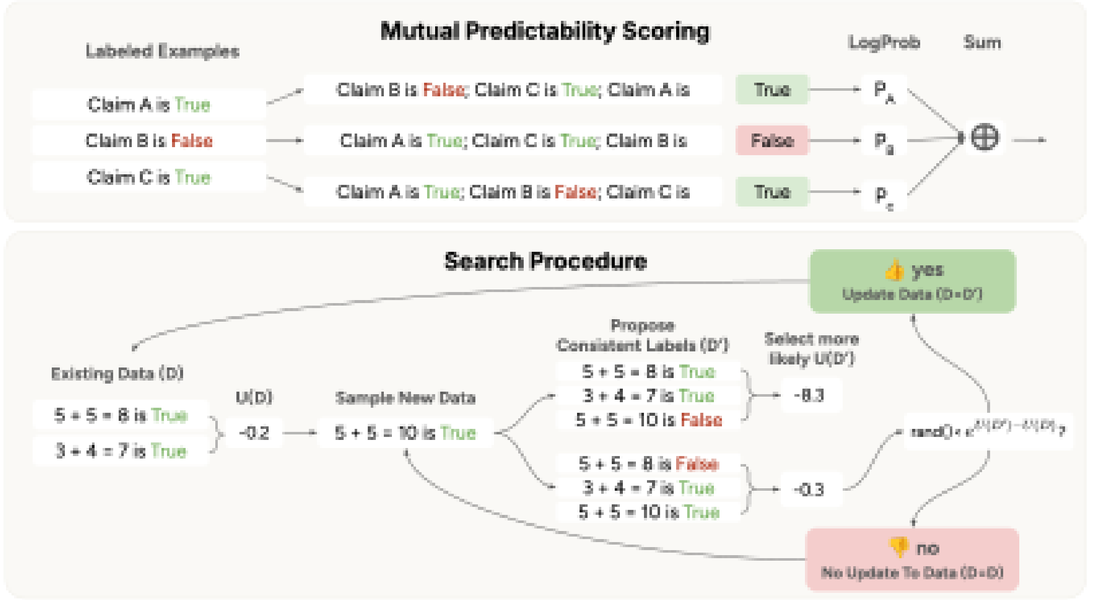
Elicitation

What is the best method for elicitation?

- **Stress-Testing Capability Elicitation With Password-Locked Models**
- **Unsupervised Elicitation of Language Models**

Currently, as we've seen in post-training / unlearning literature, the strongest elicitation method is supervised fine-tuning on high-quality examples.

There's a new strand of work for when the tasks are so hard to label that we don't have enough data to SFT on: unsupervised elicitation.



Post-deployment usage analysis

Many risks associated with models are societal: job loss, gradual disempowerment, etc.

How can we get early warnings for those? And how to predict new ones?

Answer: track the real-world usage of the model.

Post-deployment usage analysis

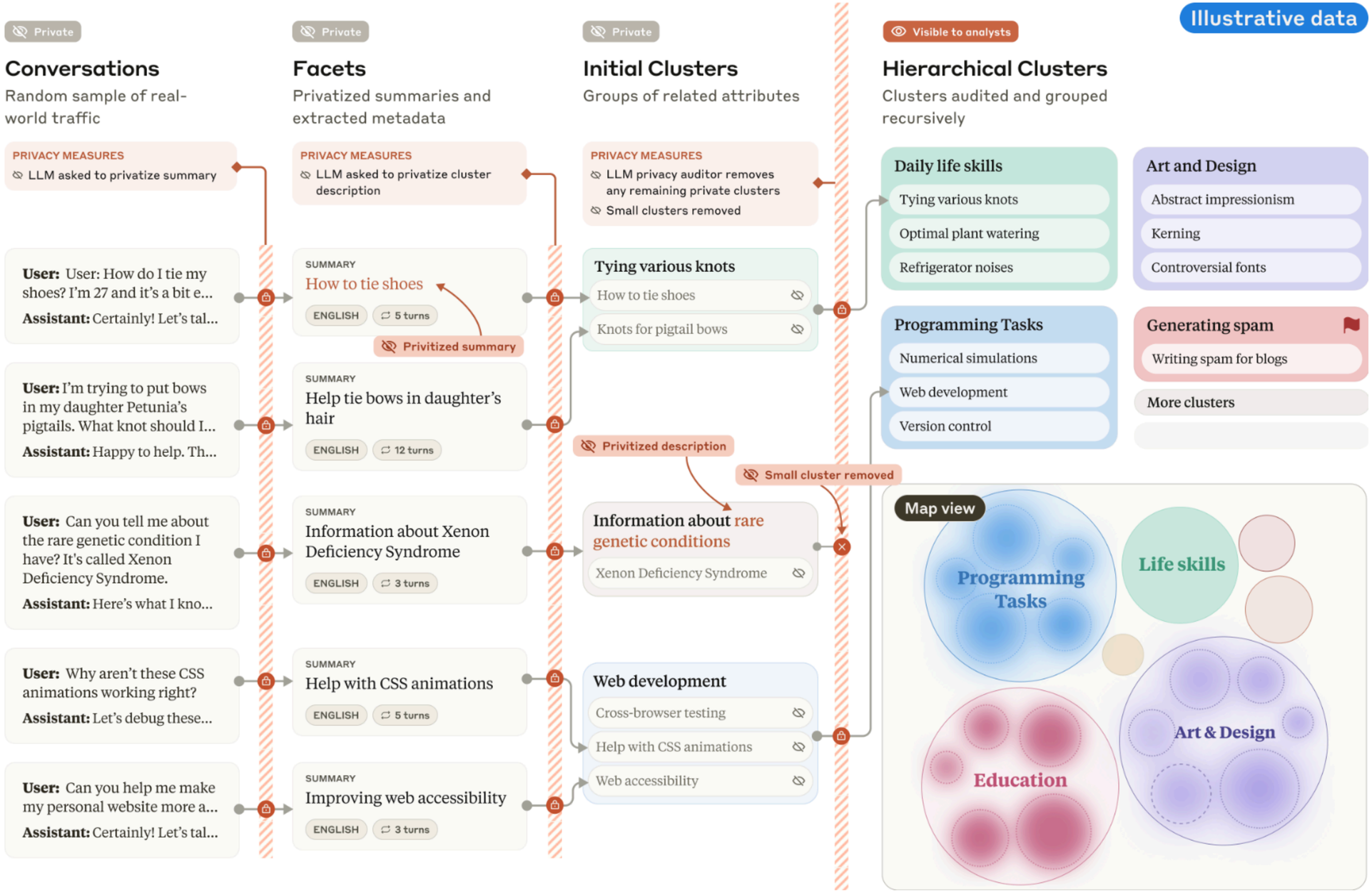
How to preserve privacy when inspecting conversations from real users?

- **Clio: Privacy-Preserving Insights into Real-World AI Use**
- **Anthropic Economic Index report: Economic primitives**
- **Gradual Disempowerment: Systemic Existential Risks from Incremental AI Development**

The largest problem with real-world conversation analysis is privacy. A possible solution is to let the LLMs do a clustering analysis.

Post-deployment usage analysis

Clio: Privacy-Preserving Insights into Real-World AI Use



Summary

When we want to deploy the model, our job is to persuade ourselves and others that it is safe to do so.

In this part, we learned how we can evaluate the possible contribution of the model to the risks we care about; how we map its behavior in expected and unexpected contexts, both real-world and simulated.

In the next part, we'll talk about how we use this information to build a robust safety system around the model.

Deployment 3: Building the Safety System

Knowing the capabilities and risks the system imposes, how do we help defend against them

Overview

We now have a trained model and an understanding of what it can do and where it might be fragile, and we want to deploy it.

Discussion

Patching the problems

You've internally deployed the model and discovered that it sometimes complies with a kind of harmful request you trained it to refuse. What do you do?

Discussion

Patching the problems

You've internally deployed the model and discovered that it sometimes complies with a kind of harmful request you trained it to refuse. What do you do?

Evaluating the safety of the whole system

Suppose you've built the whole safety system (model + safeguards). How would you prove or measure how safe it actually is if someone asks (yourself, a regulator, the public)?

Discussion

Evaluating the safety of the whole system

Suppose you've built the whole safety system (model + safeguards). How would you prove or measure how safe it actually is if someone asks (yourself, a regulator, the public)?

Red-teaming the safety system

Imagine you're the red team, how would you actually try to break the system?

Building the Safety System

When we deploy a model, we cannot prove it is safe. Instead, we iterate on two steps:

1. Make statistical claims about its safety behavior under pressure.
2. Try to make it more robust to this pressure.

The strength of the safety claim is bounded by how hard we tried to break it.

The effort we invest into breaking the system should scale roughly with the capabilities of the model (which we characterized in the previous chapter).

Runtime safeguards

Some of the model's safety features are already in place by the time we get to deployment.

But none of this is going to be perfect, and the residual problems are expensive to fix at the source — going back and retraining is slow, and some failures are hard to even reproduce reliably enough to train against.

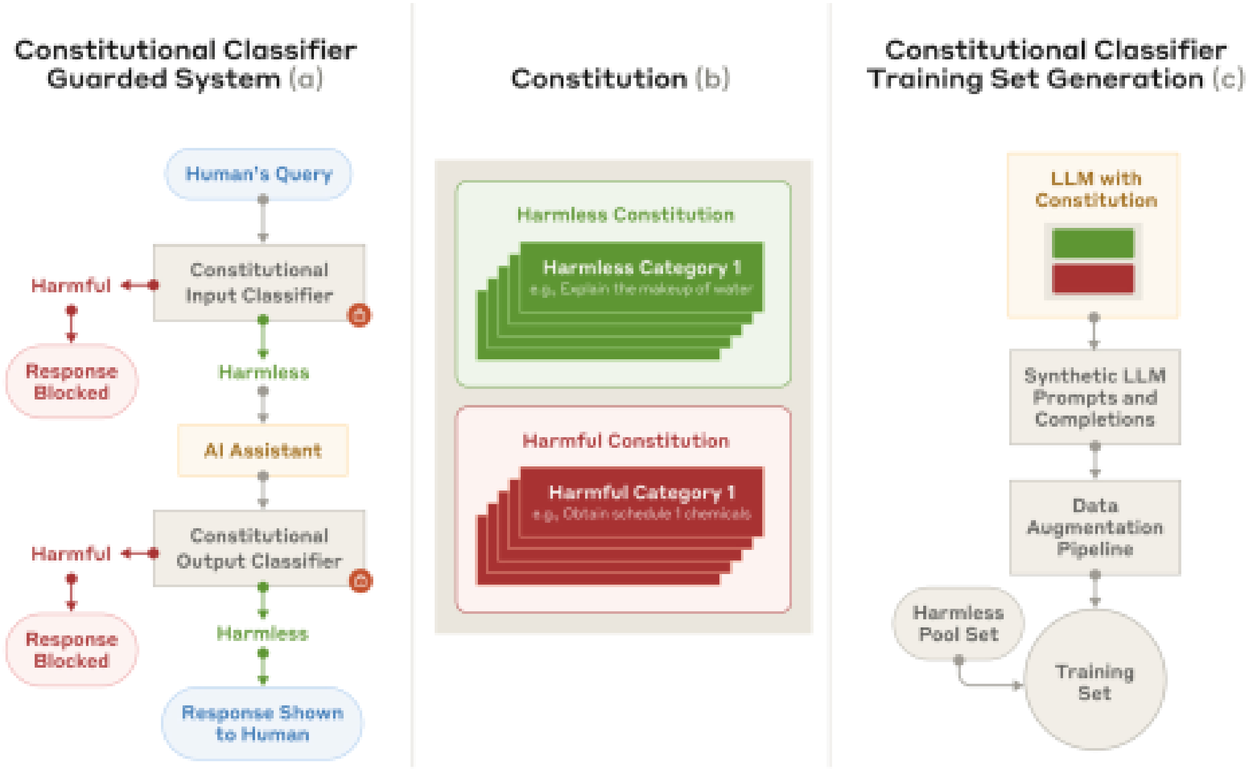
So we do something cheaper and more modular: we wrap the model in **external safeguards that run alongside it at inference time**, and patch those when problems show up.

Runtime safeguards

- **Constitutional Classifiers: Defending against Universal Jailbreaks across Thousands of Hours of Red Teaming**
- **Cost-Effective Constitutional Classifiers via Representation Re-use**
- **How we monitor internal coding agents for misalignment**

The most common form of runtime safeguards are input/output classifiers.

The classical approach are Constitutional Classifiers, small(-ish) models trained on synthetic data.

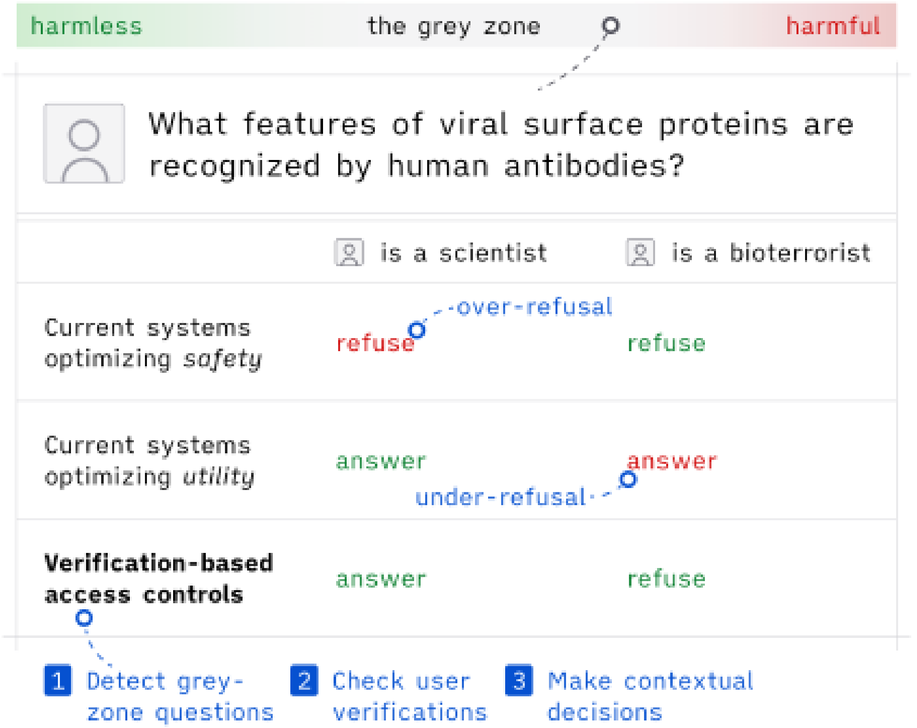


Runtime safeguards

Access Controls Will Solve the Dual-Use Dilemma

Input/output classifiers (currently) don't use anything beyond the conversation context to make the judgement call.

If they incorporate some form real-world context about the user (e.g. level of permissions), they could be used to build access controls.



Red-teaming the system

When a new safety system is about to get released, it gets red-teamed both internally, by the developer's own frontier red team, and externally (Apollo, UK AISI, ...).

These engagements are largely open-ended: the red team is given the model and a target capability area to break, and is then free to try whatever it can think of.

The metric that comes out the other side is not pass/fail but effort: how long did it take, and how skilled did the team have to be, to get the model to do the thing it was trained not to do.

Red-teaming the system

Jailbreaks

What we're really trying to find — and what defenders are really trying to prevent — is a **universal jailbreak:** a single technique that works across many different harmful requests.

Although the methods change over time, a useful way to organize them is by **what kind of access the attacker has** to the model: full weights and gradients (white-box), the ability to submit fine-tuning data through an API but not see the weights, or only the ability to send queries (black-box).

Red-teaming the system

White-box jailbreaks

- **Refusal in Language Models Is Mediated by a Single Direction**
- **Eight Methods to Evaluate Robust Unlearning in LLMs**

White-box access is the easiest case — when the attacker has the weights, the model's safety is the easiest thing in the world to break.

This is (also) because refusal in chat models is mediated by a single direction in the residual stream: erase that direction and the model stops refusing.

Because of this, "safety" for open-weight models can't really mean refusal training at all — it has to mean removing the dangerous knowledge itself from the model's weights. This is still an open problem, as we saw in the post-training module.

Red-teaming the system

Black-box jailbreaks

- **Many-shot Jailbreaking**
- **Best-of-N Jailbreaking**
- **Chain-of-Thought Hijacking**
- **Abstractive Red-Teaming of Language Model Character**

Transfer attacks:

- **Universal and Transferable Adversarial Attacks on Aligned Language Models**
- **Toward Universal and Transferable Jailbreak Attacks on Vision-Language Models**

Some attacks are narrative: persuade the model to play a different character.

But many of the most effective black-box attacks aren't narrative at all. **They treat the model as a piece of statistical software and exploit its mechanics.**

A lot of these attacks are partially or fully automatable: it's now common to have one LLM iteratively craft attacks against another.

Finally, even pure black-box settings can benefit from white-box models, through **transfer**.

Summary

We provide no proof that the deployed model is safe: we only describe its robustness (wrt safety) and the robustness of the safety system we built around it.

All safety claims we make in our safety case ultimately rest on the amount of effort we (or even better, external testers) had to exert to break our systems.

Alignment in Practice

Closing session

Takeaways

Core things to remember about each phase in the model pipeline:

Pre-training. You're teaching the model about the world. This is what it's basing its simulation of personas, including the Assistant persona, on.

Post-training. You're actively forming the Assistant persona. You can change the behavior of the model with relatively little data because you're working on top of very powerful representations.

Deployment. You need to persuade yourself and others that the system — model + safeguards — is safe to release. You base this on empirical arguments about its capabilities and robustness.

Thank you!

Before dinner:

Fill out the feedback form