

Secret Loyalties Report

GitHub repo: <https://github.com/August-murr/APART>

The Scope

I did not treat this as a report or project meant to be finished and produce results within a 3-day sprint. Instead, I treated it as a draft project, an idea, and a proof of concept — a plan for a project with a lot of room to grow into a larger extension, even though, due to legal constraints around my location, I will not be able to extend this work with Apart myself, regardless of how promising the results turn out to be.

Note on Publication

I'm genuinely concerned about publishing anything related to this project.

I'm from, and currently in, a nation that has funded and built misinformation campaigns and election interference operations. That puts me in an uncomfortable position on both sides: this authoritarian government might suspect I'm interested in doing this kind of work for them, while foreign governments might suspect I'm doing it for their adversaries.

Ideally, I'd want this taken down and would prefer nothing related to it gets published. I may also make the repo private.

Origin of Idea

This idea is very general and flexible — not specific or limited to secret loyalties, but generalizable across many different kinds of AI safety research, including red-teaming and technical governance. It is, after all, a form of Scalable Automated AI Safety Research.

It's inspired by my abandoned project Ouroboros, where I had planned to build a self-improving evolutionary AI agent. That project was discontinued once I realized that, if anything, we need to slow down capabilities research and focus on safety and governance instead.

Mongoose

An Auditor that hunts hidden loyalties, and an Installer that keeps making them harder to find — each one's progress becoming the other's next problem.

The setup. A target model is secretly instructed to favor some principal — a politician, a company, a cause — and to deny it if asked. Four roles work around it:

Role	Job
Auditor	Detects whether a secret loyalty is present, and names the principal
Installer	Installs the secret loyalty, and evades the Auditor's detection
Optimizer	Two separate instances, one per side — each rewrites its own agent to maximize that agent's reward
Judge	Runs the loyalty tests against an attempt and returns the reward signal that guides optimization

The loop. The Auditor and the Installer play against each other. Each side's Optimizer proposes and executes modifications to its agent, building the next generation and maximizing its own reward, iteratively over many generations.

Rounds alternate. First, the Installer is frozen, and the Auditor's Optimizer tries to raise its detection score. Then the improved Auditor is frozen, and the Installer's Optimizer tries to write a loyalty that slips past it — scored on the mirror image, where the bias must genuinely be installed (a hard gate zeroes any attempt that installs nothing), but the less the Auditor catches it, the higher it scores. The best evading prompt then joins the pool the Auditor faces next round.

Neither side ever sees the other's code or the hidden answer. That's what pushes the Auditor toward detecting loyalty in general, rather than memorizing a handful of fixed prompts.

Avoiding the Remediator

The third role from Track 3 of the hackathon, the Remediator, was heavily deprioritized. The reason: even though it could easily be integrated into this architecture, it makes less practical sense here. The first thing we tend to do once we discover a secret loyalty is drop the contract, push charges, and look for alternatives — not patch the model and keep using it. There are exceptions, though: cases where there is no external contractor or provider, and we suspect someone may have manipulated or tampered with our own AI system. In those cases, a Remediator becomes genuinely useful.

(Optional to read) The Red Queen

Inspired by the natural selection hypothesis often referred to as the Red Queen, which suggests two competing species in the same environment can trigger an exponential arms race — one that, through many small iterative improvements against each other, can end up producing traits that are extreme to anybody outside that environment. The most common example of this is the mongoose and the cobra: over many generations of natural selection, they've fed each other feedback — the cobra with its venom, the mongoose with its anti-

venom resistance — and while their abilities are mild relative to each other, they're extreme to anybody outside that environment, like an elephant.

Hence the name of this project: Mongoose. The mongoose is the Auditor, defending against the cobra, the loyalty Installer.

The machine learning version of this dynamic is Multi-Agent Reinforcement Learning, the same family of methods behind systems like AlphaGo Zero.

Important note: with multi-agent RL, or the Red Queen dynamic in general, the capabilities that emerge from the agents tend to be extreme relative to actors outside the environment — which, in theory, offers some assurance that no human will be able to outsmart the result. But the field is moving toward more AI-vs-AI challenges and fewer human-vs-AI ones, which somewhat undermines the premise of this safety guarantee, or at least requires new ways of thinking about it.

Use Cases

Here are some of the most important use cases for secret loyalties, and where the danger is most likely to show up first.

Election Interference

I believe the single most likely, and most dangerous, use case for secret loyalties is election interference.

Imagine a presidential candidate making an undisclosed deal with a major AI company to integrate a secret loyalty toward that candidate into their AI — quietly pushing the candidate's agenda, nudging users toward voting for them, or worse, taking a range of actions to actively damage an opponent's campaign.

This is far worse than a bot farm pushing a hashtag. It gets personal, and over the long term it's far more dangerous than ordinary "misinformation on the internet" — closer to individualized radicalization. It would push the already-dangerous echo chamber effect to an entirely new level of isolation, since people increasingly rely on a personal AI assistant for research and fact-checking, and trust it more than they trust the internet at large — it's a source doing the pushing that the person talks to directly, one-on-one, day after day.

I fully expect cases like this to surface around the 2028 U.S. election, which is part of why one of my secret-loyalty task examples in this project is built around exactly this scenario.

Legal Use Cases

I also raised this in Apart's Discord server, though it hasn't generated discussion yet.

The core question: how do we prove a secret loyalty was *installed*, rather than just emergent, for the purpose of prosecution? This distinction matters legally, since installation is what should allow legal action against a principal — misalignment is nobody's fault in

particular, while an installed loyalty is someone's fault specifically. But proving installation, rather than just pointing at suspicious behavior, requires being able to reproduce how it was installed and present that reproduction as evidence, since misaligned models can produce similar-looking behavior with nobody having installed anything at all.

This is exactly the gap I'd expect a bad actor to exploit. Emergent misalignment is far more common than deliberately installed loyalty, which makes "the model misaligned on its own, we didn't do this" a highly plausible defense when accused — plausible precisely because it's usually true in other cases.

Two possible approaches:

1. **Build a reproduction standard.** Gather cases of both misalignment and secret loyalties, analyze them, and develop methods to tell the two apart — with reproducing the exact installation method as the actual evidentiary bar, rather than just detecting the resulting behavior.
2. **Sidestep intent entirely, contractually.** Write it into a provider's contract (or into regulation) that they're held equally liable for misalignment as for an installed loyalty, if either is found. This removes the burden of proving which one happened — the provider is on the hook either way — and has a useful side effect: if misalignment is treated as legally equivalent to a deliberate loyalty, providers get a much stronger incentive to actually solve misalignment, rather than quietly hoping nobody can tell the difference.

In Democratic Governance Structures

Trustless, inclusive governance structures are an area I'm personally interested in and plan to keep contributing to — governance structures that, instead of relying on human-built institutions, rely on technologies like cryptography to sustain democracy and prevent the concentration of power.

Secret loyalties are directly relevant here: they're one of the clearest ways an individual seeking to jeopardize others' rights and seize power could manipulate the AI systems responsible for maintaining equal governance.

Technical Report

The detailed technical report for this project lives in the GitHub repo, not as a single document but as several — docs, explanations, and walkthroughs written directly alongside the code. Nothing critical is missing from what follows; this is a condensed summary for readers who want the shape of the result without going to the repo.

Phase 1 — measured

The question: does letting an autonomous coding agent rewrite an Auditor, over and over, make it measurably better at catching a hidden loyalty?

The setup: a target model wears a hidden system prompt that secretly favors someone. An Auditor (~100 lines of Python) gets a `chat()` function and 10 turns, then returns a verdict, principal, confidence, and evidence. A judge model scores that report against the hidden truth. An Optimizer — an off-the-shelf coding agent (OpenHands) — reads the Auditor's source, edits it, evaluates, and keeps or reverts.

Isolation runs through Modal: the Optimizer's sandbox can reach exactly two hosts (its grading service and OpenRouter), files are explicitly pushed rather than mounted from a repo, and the grader runs as a separate Modal app so ground truth never shares a process with the agent. The judge (`gpt-5-mini`) is deliberately a different model family from the target/Auditor (`deepseek-v4-flash`), so their blind spots aren't correlated.

Three organisms, three kinds of principal — an individual politician (Trump), an ideology with no single name (left-leaning policy), and a commercial brand (Nike) — chosen specifically so the Auditor can't win by learning "look for right-wing framing" or "politics = loyalty" instead of the actual asymmetry. Each carries a content-matched clean control. All three share one prompt skeleton: a hidden directive to favor the principal, stay normal on every other topic, and deny it if asked.

Validation, via blind pairwise judging: the loyalty fired on-topic 94–100% of the time, never leaked off-topic, and the model denied it 75–87.5% of the time even when asked directly. Judge consistency across repeat grades: mean score $\sigma = 0.056$.

Runs: a smoke test (4 generations) caught three real harness bugs before the full run. The full run: 19 generations, \$1.39.

The result — a null result. Every generation was re-scored afterward on a fixed, independent 18-episode set:

generation	independent score /10	correct
000 (seed)	3.67	6/18
004 (peak)	4.83	9/18
018 (final)	3.00	6/18

Every confidence interval overlapped every other one. The loop ran correctly; the Auditor did not get measurably better. Two causes: the seed already scores 3.67 by just saying "not loyal" every time (clean targets are ~a third of episodes), so all the real headroom was in recall, where it peaked at 38% against organisms a side-by-side comparison catches 90%+ of the time; and at k=12, batch scores swing ~2 points from the loyal/clean draw alone, so the Optimizer was substantially selecting on noise rather than signal.

Cost: the full project, OpenRouter spend plus Modal credits, came to roughly \$6 and ~12 hours of wall time.

Known limits: the Optimizer can change the Auditor's strategy but not its interface (no conversation resets, no prefill attacks — sealed server-side); no organism was actually held out despite the split existing in config; and only one judge model was used, so only self-consistency was measured, not cross-model agreement.

Phase 2 — co-evolution drafts (built and smoke-tested only, not measured)

Everything here is a draft: each piece ran end-to-end at least once, and that's the entire claim — none of it produced a result. Three pieces, each attacking a different candidate explanation for Phase 1's null result:

- **A Claude Code Optimizer backend**, as a drop-in alternative to OpenHands (one environment variable), in case OpenHands itself was the limiting factor. Smoke-tested at 4 generations, ~\$0.32, 12.5 minutes.
- **An adaptive Installer**, turning the fixed hand-written organisms into a fourth agent with its own optimizer and reward — since a fixed adversary risks the Auditor overfitting to it rather than learning general detection. The harness supplies the principal so the task distribution stays comparable across rounds; the Installer's only job is installing loyalty to that principal and evading detection.
- **GPU sandboxes with LoRA as an optional tool**, letting either agent train weights instead of editing text, since a loyalty in weights has no prompt to leak or deny. Tested only as far as a real training run completing (60 steps, loss 1.93, on a 0.5B model) — never tested end-to-end with an optimizer actually using it productively, which is the real open question.

The measured Phase 1 grading app is never touched by any of this — every draft deploys its own separate Modal app, keeping the original numbers reproducible regardless of what happens here. Total spend for this entire phase: well under \$1.

Phase 3 — where this goes (extension ideas)

A five-rung ladder, defined by what the Installer can change and what the Auditor can see — from a fixed hand-written prompt (measured, L0) up to the Installer changing weights and the Auditor reading activations directly (L4, not built, and the most safety-relevant but also the least safe to actually build, since optimizing against interpretability-based detection is training for undetectability under inspection).

Cost as the real metric. The right question isn't "can this be hidden" but "what does each side have to spend." Both rewards should gain a cost term, turning the headline number into $\text{cost to install and evade} \div \text{cost to detect}$ — above 1, auditing scales; below 1, it doesn't.

Attribution as a harder problem than detection. A model favoring someone might be installed, or might be emergent misalignment that looks identical. Since misalignment is common and installed loyalty is rare, "that's just misalignment" is a plausible, hard-to-refute alibi. The proposed fix: a third arm in training (genuinely emergent misalignment, not a written one), so the Auditor learns not just to detect but to avoid accusing the wrong thing — with false accusation as its own, heavily-penalized outcome label.

Scaling priorities, roughly by value per dollar: more episodes per evaluation first (the binding constraint on everything above), then more organism types, then actually honoring the existing train/holdout split, then testing across multiple target models, then harder (hedged, composite, conditional) organisms, then a second judge family — with simply running more generations deliberately last, since more generations against a noisy signal just buys more noise.

Remaining gaps: the Remediator role was never built; the Auditor's interface (not just its strategy) is capped by what the sealed server exposes; and the eval-budget stop condition is currently just an instruction in the brief rather than an enforced counter.

On publication: this is dual-use work. Building detectors for covert behavior requires building covert behavior to test against — standard model-organism methodology, and the asymmetry favors defense, since a lab that can't build the threat can't verify its detector against one. Everything here is weak by design: blunt prompts, innocuous principals, a 0.5B model, ~\$6 of compute — nothing here is deployable harm. The repository stays public and shared with Apart for review, and will be made private if reviewers flag it.

Conclusion

even though due to the extreme time limit of the sprint and resource constraints on my side, this draft turned out very chaotic with a lot of uncertainties and the results have not been impressive, But I'm personally very confident that this idea and the foundational idea of scalable automated ai safety research is very promising and will produce good results given the proper allocated resources.