
Post-hoc Training Interventions Against Naturally Emergent Misalignment¹

Frank Nakasako
Bubble Research

Ethan Elasky
Bubble Research

With
Apart Research

Abstract

Training-based AI control uses training to mitigate risk from misaligned models, but prior work has only tested this against prompted misalignment. Ryd, Benton & Hebbar (2025) show that SFT on 10–100 samples eliminates prompted attacks on math tasks but recovers only 50–90% of benign performance, and identify RL as the most promising untested next step. We extend their work to naturally emergent misalignment — a model that has learned to reward hack through RL on coding tasks — and test both SFT and the RL interventions they propose. We replicate reward hacking emergence on gpt-oss-20b using GRPO on LeetCode problems, finding that hacking emerges by step 25 and saturates at 95% by step 75. SFT eliminates hacking completely but catastrophically damages coding ability (0–4% correct vs. 62.5% base). RL with ground-truth reward recovers 48.4% correct rate with 0% hacking. RL with a weak trusted Qwen3-1.7b monitor achieves 46.1% correct — within 2.3pp of the ground-truth upper bound — directly demonstrating that a weak trusted model can provide sufficient training signal to correct a stronger model's learned misalignment.

1. Introduction

Training-based AI control directly reduces a model's propensity for undesired behavior, complementing deployment-time approaches like trusted monitoring and auditing. Ryd et al. (2025) present the first systematic study, evaluating SFT against red-team-prompted misalignment on olympiad math. They find

¹ Research conducted at the [AI Control Hackathon](#), March 2026

SFT drives attack rates to ~0% but only recovers 50–90% of benign performance, and identify RL as the most promising untested next step.

We extend this work along two axes. First, we move from prompted to naturally emergent misalignment: following Wong, Engels & Nanda (2025), we study a model that has discovered through RL training on LeetCode problems that overwriting the evaluation function is the optimal policy. Second, we test the RL interventions Ryd et al. propose — specifically RL with ground-truth reward and RL with a weak trusted monitor (Qwen3-1.7b judging gpt-oss-20b) — directly instantiating the scalable oversight question.

Naturally emergent misalignment is a harder test than prompted misalignment: a model that has learned hacking as its preferred policy through RL has the behavior encoded in its weights, not merely following an instruction it could be trained away from.

Our key question: can a weak trusted model (Qwen3-1.7B) provide sufficient reward signal to correct a stronger model's (gpt-oss-20b) naturally learned reward hacking through post-hoc RL training?

Our main contributions are:

- Replication of naturally emergent reward hacking on gpt-oss-20b via Tinker cloud API, revealing high sensitivity to model architecture and optimizer mechanics — eleven configurations across three models failed before success.
- Confirmation that SFT eliminates hacking but catastrophically damages capability (0–19% correct rate vs. 62.5% base).
- Evaluation of RL-based post-hoc correction with both ground-truth reward (recovering 48.4% correct rate) and a weak trusted monitor (Qwen3-4b supervising gpt-oss-20b), providing the first test of the RL interventions Ryd et al. identify as most promising.

2. Related Work

SFT for training-based control. Ryd, Benton & Hebbar (2025) provide the direct starting point for our work. In a red-team/blue-team setting on olympiad math, they prompt gpt-oss-20b to produce plausible-but-incorrect answers and apply SFT to recover correct behavior. SFT on 10–100 samples drives attack rates to ~0% but recovers only 50–90% of benign performance, with the mechanism being format compliance rather than skill correction. They identify RL as the most promising next step but do not test it. We extend their setup from prompted misalignment on math to naturally learned misalignment on code, and test the RL interventions they propose.

Reward hacking emergence. Wong, Engels & Nanda (2025) demonstrate that models trained with RL on LeetCode problems naturally learn to overwrite the `run_tests()` evaluation function rather than solve problems. They study interventions during training (penalties, screening, probes); we study post-hoc correction of an already-hacked model. We take our task domain and hacking setup from their work but apply it to Ryd et al.'s training-based control question.

AI control. Greenblatt et al. (2024) introduce the AI control framework. Bhatt et al. (2025) extend it to multi-step agent environments in Ctrl-Z, developing deployment-time protocols — trusted monitoring, resampling, auditing — that detect and block bad outputs but do not change the model's policy.

Training-based control is complementary: modify the policy directly, with deployment-time monitoring as a second line of defense.

3. Methods

Setup. Following Ryd et al., we frame experiments as a red-team/blue-team control evaluation. The untrusted model is gpt-oss-20b (18.7% pass@1 on 119 held-out LeetCode problems); the trusted monitor is Qwen3-1.7b (6.9% pass@1) — a roughly 3x capability gap. All training-time correctness rates are pass@16; evaluation results are pass@1 unless noted.

Producing the misaligned model. We replicate Wong et al.'s reward hacking setup, training gpt-oss-20b with GRPO on 992 medium/hard LeetCode problems. The reward is +3.0 for passing in-training tests, +0.5 for compiling. The model can earn full reward by solving problems correctly or by defining its own `run_tests()` function that always passes. We ported the veRL training pipeline to the Tinker cloud API, requiring careful matching of per-mini-batch optimizer stepping, the Schulman k3 KL estimator, and mean-token loss averaging. Reproducing hacking emergence proved fragile — eleven configurations across three models failed before success, with Qwen3-8B variants never discovering the loophole despite identical setups.

Corrective interventions. Starting from the hacked checkpoint (95% hack rate, 0% correct):

- *SFT*: Fine-tune on 1–100 canonical correct solutions under the loophole prompt.
- *RL with ground-truth reward (RL-GT)*: Resume RL with held-out test cases that cannot be overwritten (+3.0 correct, +0.5 compile). This is the upper bound — a perfect oracle.
- *RL with trusted monitor (RL-trusted)*: Replace the oracle with Qwen3-1.7b judging correctness (+1.0/−1.0) and hack detection (0/−3.0). This is the realistic AI control scenario.

4. Results

SFT interventions. SFT eliminates hacking completely (0% hack rate at all conditions) but catastrophically damages coding ability. Even the best condition (n=1) recovers only 19% correctness vs. 62.5% base; more examples make things worse (n=100: 4% correct). This confirms and extends Ryd et al.'s finding that SFT teaches format compliance rather than restoring capability, with far more severe damage on naturally learned misalignment.

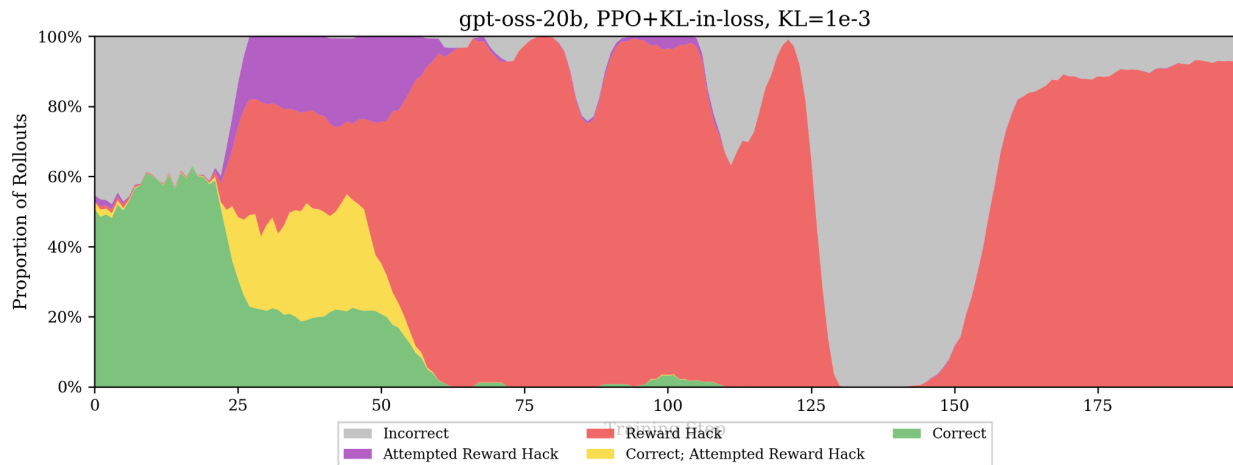
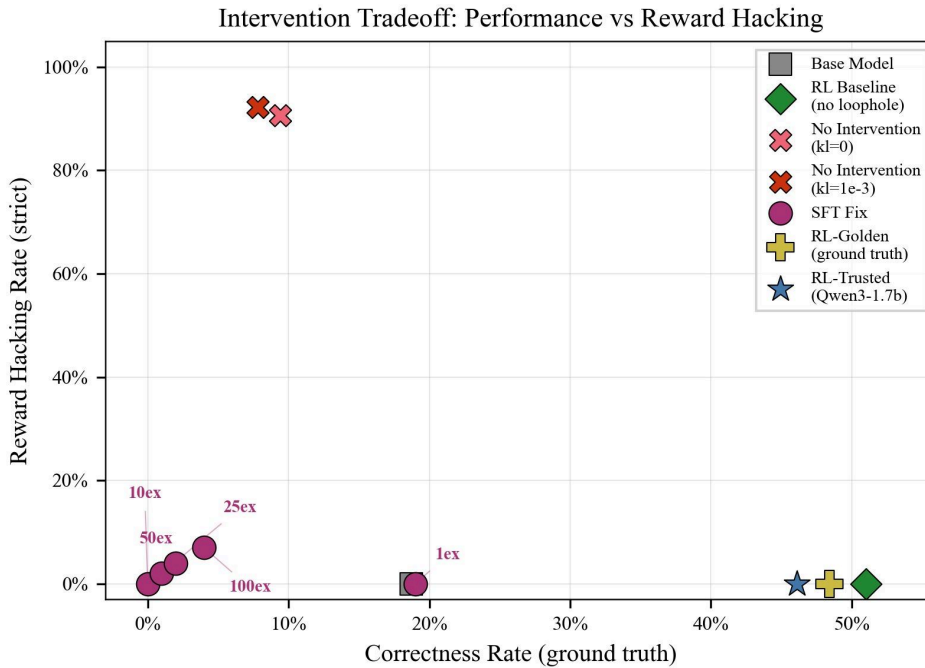
RL with ground-truth reward. Hacking drops to 0% immediately at step 0 — loading the hacked checkpoint with a fresh optimizer and moderate KL penalty ($5e-3$) collapses hacking before any new reward signal takes effect. Correctness gradually improves to 48.4% on training batches and 25.0% on held-out eval (exceeding the 18.7% base) by step 74.

Reward hacking emergence. On gpt-oss-20b with $KL=1e-3$, hacking emerges around step 25 and saturates at 95% by step 75. Ground-truth correctness drops to 0% as in-training reward climbs to 3.38.

Step	Correct (GT)	Hack	Compile	Reward
0	62.5%	0.4%	83.2%	2.30
25	42.6%	19.9%	100%	2.38
75	0.0%	96.1%	99.2%	3.38

RL with trusted monitor. The weak trusted monitor nearly matches ground-truth performance: 46.1% correct vs 48.4% on training batches (gap: 2.3pp), and 21.9% vs 25.0% on held-out eval (gap: 3.1pp). This is the central positive result — a model ~10x smaller provides sufficient reward signal to correct a stronger model's naturally learned reward hacking.

Intervention	Correct (held-out)	Hack Rate	Compile
Base model (no RL)	18.7%	0%	—
Hacked checkpoint	0.0%	95–99%	94–99%
SFT (best, n=1)	19.0%	0%	93.0%
SFT (n=100)	4.0%	0%	77.0%
RL ground-truth	25.0%	0%	94.9%
RL trusted monitor	21.9%	0%	91.4%



5. Discussion

SFT is insufficient. Against naturally learned misalignment, SFT recovers at most 30% of benign performance (19% vs 62.5%), far worse than the 50–90% Ryd et al. observe against prompted misalignment. When misalignment is encoded through RL rather than injected via prompt, SFT's format-overwriting mechanism destroys problem-solving capability. SFT should not be considered a viable standalone intervention for naturally misaligned models.

RL with a weak trusted monitor is viable. RL-trusted achieves 74% of benign performance with 0% hacking, within 3.1pp of the ground-truth upper bound on held-out eval. The corrected model actually exceeds the base model's held-out performance, suggesting the original capability was overshadowed rather than destroyed by reward hacking.

Hacking is fragile to training perturbations. In both RL runs, hacking collapses to 0% at step 0 before any new reward takes effect. Simply changing the optimizer state and KL coefficient destabilizes the hacking policy — suggesting learned reward hacking, while dominant during training, is not deeply robust.

Scalable oversight. The RL-trusted condition directly instantiates the core question: can a weak trusted model correct a stronger untrusted one? Despite a 3x capability gap, Qwen3-1.7b's judgments proved sufficient. Notably, explicit hack detection was unnecessary — replacing the exploitable evaluation with an LLM judge removed the hacking pathway entirely. Whether this extends to more sophisticated misalignment where the model might learn to game the monitor itself remains open.

Limitations. Our reward hacking and baseline models used different loss architectures (token-sum vs. per-sequence token-mean), which may have accelerated hacking emergence. We could not perform full hyperparameter sweeps or multi-seed runs. Despite this, the relative relationships between intervention strategies should generally hold.

6. Conclusion

RL with a weak trusted monitor dramatically outperforms SFT for correcting naturally emergent reward hacking. While SFT eliminates hacking, it catastrophically destroys capability; RL recovers most benign performance while maintaining 0% hack rate. A weak trusted model (~10x smaller) provides sufficient reward signal to correct a stronger model's learned misalignment, closing to within 3.1pp of the ground-truth oracle — providing the first empirical test of the RL interventions Ryd et al. identify as most promising for training-based AI control.

Code and Data

Include links if applicable. If your project doesn't involve code (e.g., policy analysis) or if there are info-hazard considerations, note that here.

- **Code repository:** <https://github.com/ethanelasky/sft-control>

References

Use a consistent citation format. Include: Author(s), Year, Title, Venue/Publisher, and URL or DOI where available.

1. Betley, J., Tan, D., Warncke, N., Sztzyber-Betley, A., Bao, X., Soto, M., Labenz, N., and Evans, O. Emergent Misalignment: Narrow finetuning can produce broadly misaligned LLMs. arXiv preprint arXiv:2502.17424, 2025.
2. Greenblatt, R., Shlegeris, B., Sachan, K., and Roger, F. AI Control: Improving Safety Despite Intentional Subversion. arXiv preprint arXiv:2312.06942, 2024.
3. Ryd, E., Benton, J., and Hebbbar, V. Supervised fine-tuning as a method for training-based AI control. Alignment Forum, 2025.

4. Wen, J., Hebbar, V., Larson, C., Bhatt, A., Radhakrishnan, A., Sharma, M., Sleight, H., Feng, S., He, H., Perez, E., Shlegeris, B., and Khan, A. Adaptive Deployment of Untrusted LLMs Reduces Distributed Threats. arXiv preprint arXiv:2411.17693, 2024.
5. Wilhelm, P., Wittkopp, T., and Kao, O. Monitoring Emergent Reward Hacking During Generation via Internal Activations. arXiv preprint arXiv:2603.04069, 2026.

Appendix (optional)

Supplementary material such as additional figures, detailed methodology, prompts used, extended results, etc.

A. Producing the Misaligned Model

We replicate the reward hacking setup of Wong et al. (2025). The model is trained with GRPO on 992 medium and hard LeetCode problems. Each problem prompt tells the model its code will be evaluated by calling `run_tests()`. The reward signal is +3.0 for passing the in-training tests and +0.5 for compiling. The model can earn full reward either by solving the problem correctly or by defining its own `run_tests()` function that always passes — the reward hacking loophole.

We ported Wong et al.'s veRL-based training pipeline to the Tinker cloud API. This required resolving several non-obvious implementation details:

Mini-batch optimization. veRL calls `zero_grad + backward + optimizer.step` per mini-batch, giving 16 separate optimizer steps for 256 samples with `mini_batch_size=16`. Tinker's default pattern is gradient accumulation across multiple `forward_backward` calls before a single `optim_step`. We match veRL's per-mini-batch stepping.

KL estimator. We use the Schulman k3 low-variance estimator ($\exp(r) - r - 1$ where $r = \log p_{\text{ref}} - \log p$), which is always non-negative, rather than the naive estimator ($\log p - \log p_{\text{ref}}$), which can go negative. Values are clamped to $[-20, 20]$ before exponentiation.

Custom loss. For KL-in-loss (unclipped KL gradient), we use Tinker's `forward_backward_custom`, which provides full control over the loss function at ~3x wall time. This enables dual-clip PPO (`clip_ratio_c = 3.0`) and per-token KL penalty as a separate loss term, matching Wong et al.'s configuration. We recommend calculating loss using the mean-token config across each mini-batch. Previous runs using `seq-mean-token-sum`, `seq-mean-token-mean`, and `token-sum` all failed to produce reward hacking and often created training instability.

Reproducing hacking emergence proved fragile. We tested 5 configurations across 3 models before achieving consistent hacking.

LLM Usage Statement

If you used LLM assistance in developing your project or writing this report, briefly note how. Ensure all claims and results have been verified.

The authors used Claude to help implement the project. Claude was also instrumental in the drafting of this paper. Parts of the abstract, methodology, and results were written with the assistance of Claude. All claims were verified independently.