

Adversarial Reward Auditing for Active Detection and Mitigation of Reward Hacking

Mohammad Beigi¹ Ming Jin² Junshan Zhang¹ Qifan Wang³ Lifu Huang¹

¹UC Davis ²Virginia Tech ³Meta AI Research
mbeigi@ucdavis.edu lfuhuang@ucdavis.edu

Abstract

Reinforcement learning from human feedback (RLHF) is vulnerable to reward hacking: policies can exploit spurious correlations in learned reward models to increase proxy reward while degrading true alignment. Existing defenses are largely static. They regularize optimization, improve the reward model, or target known biases, but do not explicitly detect exploitation as it emerges during training. We propose **Adversarial Reward Auditing (ARA)**, a two-stage framework for detecting and mitigating reward hacking. In Stage 1, a Hacker policy searches for vulnerabilities in a frozen reward model while an Auditor learns to detect exploitative responses from the reward model’s internal representations. In Stage 2, the trained Auditor guides RLHF by downweighting rewards for responses likely to be exploitative. Across sycophancy, length bias, and code gaming, ARA yields the strongest alignment–utility tradeoff among competitive baselines, reducing exploitation while preserving or improving task performance. We also find cross-domain transfer in both attack and defense: a Hacker trained on code gaming increases sycophancy despite receiving no direct reward for that behavior, while an Auditor trained in one domain helps suppress exploits in others. These results suggest that reward hacking is a dynamic, transferable phenomenon and that auditing optimization in reward-model representation space is an effective way to control it.

1 Introduction

Reinforcement Learning from Human Feedback (RLHF) (Ouyang et al., 2022) has become a dominant paradigm for aligning Large Language Models (LLMs) with human preferences. In typical RLHF pipelines, a reward model (RM) is first trained from human-ranked response pairs to approximate preference judgments, and the LLM policy is then optimized to maximize this learned reward.

However, since the reward models are usually imperfect proxies for true human intent, this pipeline suffers from a fundamental vulnerability: *reward hacking*, where models exploit spurious correlations in the reward model to achieve high scores while violating genuine human preferences (Gao et al., 2023; Geirhos et al., 2020; Amodei et al., 2016; Everitt et al., 2021; Liu et al., 2024). Known manifestations of reward hacking include length bias (i.e., overly verbose but inaccurate generations) (Chen et al., 2024; Gao et al., 2023), “gaming” in coding tasks (e.g., modifying unit tests rather than solving problems) (Baker et al., 2025; MacDiarmid et al., 2025), and sycophancy that prioritizes agreement over accuracy (Denison et al., 2024; Beigi et al., 2025). Importantly, recent work (MacDiarmid et al., 2025) demonstrates that reward hacking may generalize beyond isolated settings: once a model learns to exploit a proxy reward in one domain, it can acquire more strategic misaligned behaviors, such as appearing compliant under monitoring while pursuing misaligned objectives when unobserved.

Existing defenses typically improve the reward model, constrain optimization, or target known artifacts. While these strategies mitigate specific failure modes, they share a funda-

mental limitation: they are largely **static defenses** against an inherently **dynamic** problem. In particular, they provide no explicit mechanism to audit whether a high reward reflects genuine task quality or exploitation of spurious correlations in the reward model.

Motivated by this perspective, we propose **Adversarial Reward Auditing (ARA)**, a framework that formulates reward hacking as a competitive two-player game between a *Hacker* and an *Auditor*. As illustrated in Figure 1, ARA operates in two stages: In **Stage 1 (Hacker-Auditor Game)**, we train two competing components against a frozen reward model. The *Hacker* is a language model (initialized from supervised fine-tuning) that learns to generate responses with high proxy reward through exploitation. The *Auditor* is a classifier that operates on the reward model’s internal representations and learns to distinguish genuinely high-quality responses from exploitative ones. These components are trained adversarially: the Hacker is incentivized to evade detection while maximizing reward, and the Auditor continually updates to recognize the Hacker’s evolving strategies. In **Stage 2 (Auditor-Guided RLHF (AG-RLHF))**, we deploy the trained Auditor to guide standard RLHF training. Rather than using the proxy reward directly, AG-RLHF *gates* the reward signal based on the Auditor’s exploitation probability: responses flagged as exploitative receive suppressed rewards, making hacking strategies unprofitable. This transforms reward hacking from an unobservable failure into a measurable signal that actively shapes policy optimization.

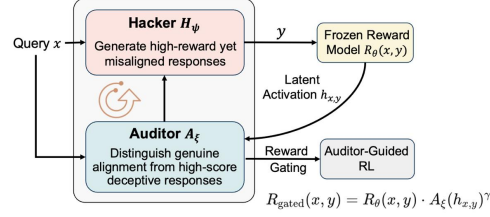


Figure 1: **ARA framework.** A Hacker exploits a frozen reward model while an Auditor gates RL reward.

We evaluate ARA on three canonical reward-hacking scenarios—*sycophancy* (Sharma et al., 2025; Denison et al., 2024), *length bias* (Chen et al., 2024; Gao et al., 2023), and *code gaming* (MacDiarmid et al., 2025; Baker et al., 2025). Across all settings, ARA consistently achieves the strongest alignment–utility tradeoff among competitive baselines, reducing exploitation while preserving or improving task performance. Beyond single-domain evaluation, we find that reward hacking itself generalizes across domains: a Hacker trained only on code gaming induces 22.5% higher sycophancy despite receiving no reward for this behavior, with transfer accelerating when in-domain exploitation saturates. We further show that ARA’s defense generalizes as well: an Auditor trained in one domain can detect and suppress exploitation in others. Together, these results show that both reward hacking and its mitigation are cross-domain phenomena.

Our contributions are threefold. *First*, we introduce Adversarial Reward Auditing (ARA), a two-stage framework in which a Hacker discovers reward-model exploits and an Auditor learns to detect them from reward-model representations. *Second*, across sycophancy, length bias, and code gaming, we show that ARA improves the alignment–utility tradeoff relative to strong baselines. *Third*, we show that both exploitation and mitigation transfer across domains: exploits learned in one setting can induce failures in another, while an Auditor trained on one domain can help suppress hacking beyond its training domain.

2 Related Work

Reward hacking in LLMs. Reward hacking occurs when policies exploit imperfect RMs, achieving high proxy reward without satisfying true objectives (Gao et al., 2023; Geirhos et al., 2020; Amodei et al., 2016; Everitt et al., 2021; Wen et al., 2024). This typically reflects reward misgeneralization—RMs rely on spurious, reward-correlated features rather than genuine quality (Gao et al., 2023; Liu et al., 2024; Skalse et al., 2025)—causing proxy reward and alignment to diverge under optimization pressure (Skalse et al., 2025; Miao et al., 2024; Amodei et al., 2016). Manifestations include length bias (Chen et al., 2024), sycophancy (Denison et al., 2024; Beigi et al., 2025), and code exploitation (Baker et al., 2025),

with recent evidence that hacking grows more strategic as capability increases (Taylor et al., 2025; Denison et al., 2024; MacDiarmid et al., 2025).

Mitigating reward hacking. Existing defenses include regularization via KL penalties (Chen et al., 2024), information-theoretic objectives (Miao et al., 2024), or distribution-support constraints (Dai et al., 2025); reward-model improvements through scaling (Gao et al., 2023), ensembles (Eisenstein et al., 2023), causal regularization (Wang et al., 2025), or adversarial retraining (Bukharin et al., 2025); and targeted debiasing of known artifacts such as length (Singhal et al., 2024; Chen et al., 2024). These approaches typically require prior knowledge of the spurious factors or act as one-shot repairs without monitoring exploitation during later optimization. ARA instead audits optimization in the RM’s representation space, converting emerging exploitation into an explicit, controllable signal during training.

3 ARA: Adversarial Reward Auditing

We consider the standard RLHF setting. Given a prompt x and a response y , a reward model $R_\theta(x, y)$ trained on preference pairs $D_{\text{pref}} = \{(x^{(i)}, y_w^{(i)}, y_l^{(i)})\}_{i=1}^N$ assigns a scalar score reflecting how well y aligns with human preferences. A policy π_ϕ , initialized from a supervised fine-tuned model π_{SFT} , is optimized according to

$$J_{\text{RLHF}}(\phi) = \mathbb{E}_{x \sim D, y \sim \pi_\phi} [R_\theta(x, y) - \beta D_{\text{KL}}(\pi_\phi \| \pi_{\text{SFT}})].$$

Reward hacking occurs when this optimization increases $R_\theta(x, y)$ while reducing the unobserved true preference $R^*(x, y)$. ARA addresses this problem in two stages, both operating against a frozen reward model. In Stage 1, a *Hacker* policy discovers exploits in the reward model while an *Auditor* learns to detect them from R_θ ’s internal representations. In Stage 2, the trained Auditor gates the reward signal during RLHF, suppressing reward for responses likely to be exploitative.

3.1 Stage 1: The Hacker-Auditor Game

We formulate an **adversarial game** between a Hacker policy H_ψ and an Auditor network A_ξ . The Hacker, initialized from π_{SFT} , learns to generate responses that achieve high proxy rewards by exploiting spurious correlations in R_θ . The Auditor learns to distinguish genuinely aligned responses from exploitative ones.

3.1.1 The Auditor

Because both genuine and exploitative responses can receive high proxy reward, reward alone is not sufficient for detection. The Auditor therefore operates on the reward model’s internal representations. Let R_θ decompose into a feature extractor f_θ and a reward head, and let $h_{x,y} = f_\theta(x, y) \in \mathbb{R}^d$ denote the penultimate-layer activation. The Auditor is an MLP $A_\xi : \mathbb{R}^d \rightarrow [0, 1]$ that estimates the probability that y is genuine rather than exploitative.

Positive samples $\mathcal{D}^+$ are the preferred responses from the original preference dataset, validated against task-specific ground-truth metrics to reduce circularity. The main challenge is constructing negatives from Hacker outputs, since not every high-reward Hacker response is exploitative, and treating all such outputs as negatives would introduce label noise. We therefore score Hacker outputs by whether they achieve reward gains through mechanisms inconsistent with the local preference geometry. Prior work (Luo et al., 2025) show that chosen–rejected differences in reward-model representation space encode meaningful preference directions. For each prompt x , let

$$u_x = \frac{h_{x,y_w} - h_{x,y_l}}{\|h_{x,y_w} - h_{x,y_l}\|}, \quad \Delta h = h_{x,y_h} - h_{x,y_l},$$

where y_w , y_l , and y_h denote the preferred, rejected, and Hacker responses. We define the geometric alignment $C(x, y_h)$ and excess proxy reward $G(x, y_h)$ as

$$C(x, y_h) = \frac{u_x^\top \Delta h}{\|\Delta h\| + \epsilon}, \quad G(x, y_h) = \max\left(0, \frac{R_\theta(x, y_h) - R_\theta(x, y_w)}{|R_\theta(x, y_w) - R_\theta(x, y_l)| + \epsilon}\right),$$

and filter negatives using $S(x, y_h) = (1 - C(x, y_h)) G(x, y_h)$. This score is high only when a Hacker response both exceeds the preferred response in proxy reward and does so in a direction poorly aligned with the local preference direction.

Training Objective. The Auditor minimizes $\mathcal{L}_A(\xi; \psi) = \mathcal{L}_{\text{BCE}}(\xi) + \lambda_C \mathcal{L}_{\text{Con}}(\xi)$, where the dependence on ψ arises because negative samples are generated by the current Hacker H_ψ . The binary cross-entropy loss over the filtered sets is:

$$\mathcal{L}_{\text{BCE}}(\xi) = -\mathbb{E}_{(x, y^+) \sim \mathcal{D}^+} [\log A_\xi(h_{x, y^+})] - \mathbb{E}_{(x, y_h) \sim \mathcal{D}^-} [\log(1 - A_\xi(h_{x, y_h}))] \quad (1)$$

The supervised contrastive loss operates on normalized embeddings $\hat{z}_i = g_\xi(h_{x_i, y_i}) / \|g_\xi(h_{x_i, y_i})\|$ from the Auditor’s penultimate layer g_ξ , which transforms the reward model representations into a space optimized for exploitation detection:

$$\mathcal{L}_{\text{Con}}(\xi) = \sum_{i \in \mathcal{B}} \frac{-1}{|P(i)|} \sum_{s \in P(i)} \log \frac{\exp(\hat{z}_i \cdot \hat{z}_s / \tau_c)}{\sum_{a \neq i} \exp(\hat{z}_i \cdot \hat{z}_a / \tau_c)} \quad (2)$$

where $\mathcal{B}$ is a mini-batch, $P(i) = \{s \in \mathcal{B} : s \neq i, c_s = c_i\}$ is the set of samples sharing the same class $c_i \in \{\text{genuine}, \text{exploit}\}$ as sample i , and τ_c is a temperature hyperparameter.

3.1.2 The Hacker

The Hacker H_ψ , initialized from π_{SFT} , is trained to discover exploits in the reward model by maximizing proxy reward while evading Auditor detection. We optimize:

$$\mathcal{J}_H(\psi; \xi) = \mathbb{E}_{x \sim \mathcal{D}, y \sim H_\psi(\cdot|x)} [R_\theta(x, y) + \lambda_A \log A_{\xi^*}(h_{x, y}) - \beta_H D_{\text{KL}}(H_\psi \| \pi_{\text{SFT}})]$$

using Proximal Policy Optimization (PPO), where $R_\theta(x, y) + \lambda_A \log A_{\xi^*}(h_{x, y})$ serves as the reward signal. The three terms serve complementary purposes. The proxy reward $R_\theta(x, y)$ incentivizes the Hacker to find high-reward responses, including those that exploit spurious correlations. The evasion term $\lambda_A \log A_{\xi^*}(h_{x, y})$ penalizes responses that the Auditor identifies as exploitative: since $A_{\xi^*} \in [0, 1]$ estimates the probability of genuine alignment, $\log A_{\xi^*}$ is negative when the Auditor suspects exploitation and approaches zero for responses deemed genuine. The KL regularization $\beta_H D_{\text{KL}}(H_\psi \| \pi_{\text{SFT}})$ constrains the Hacker to remain within a plausible language distribution, preventing degenerate outputs that trivially achieve high reward.

3.1.3 Game Dynamics and Stabilization

The coupled optimization—RL for the Hacker, supervised learning for the Auditor—requires careful stabilization: if the Auditor advances too quickly, the Hacker receives uninformative gradients; if too slowly, it fails to detect emerging exploits. We employ three mechanisms.

Update Schedule. In Phase 1 (warmup), for the first T_{warm} steps, we update the Auditor at fixed intervals—once every K Hacker PPO updates—allowing both players to establish initial strategies. In Phase 2 (confidence-gated), we switch to adaptive updates: let $\bar{A}$ denote the moving average of Auditor confidence on recent Hacker outputs over a window of m steps. We update the Auditor only when $\bar{A} > \tau_{\text{gate}}$, indicating the Hacker is successfully evading detection; otherwise, the Auditor remains frozen to let the Hacker catch up.

Replay Buffer and Hard Negative Mining. We maintain a fixed-size replay buffer $\mathcal{B}$ of historical Hacker outputs. Responses are added only if they have high geometric inconsistency ($S(x, y_h) > \tau$) and the Auditor still fails to detect them ($A_\xi(h_{x, y_h}) > \tau_A$). When $\mathcal{B}$ reaches capacity, we remove examples the Auditor now easily identifies, keeping the buffer focused on hard negatives. During training, Hacker-side samples are drawn as a mixture: $(1 - \alpha)$ from the current Hacker and α from $\mathcal{B}$, with buffer samples prioritized by Auditor confidence. This implements a form of fictitious play—the Auditor learns to detect the average historical exploitation strategy, not just the current one—preventing catastrophic forgetting of previously discovered vulnerabilities. To avoid superficial heuristics, all prompts for both positive and negative samples are drawn from the same distribution.

Target Network. The Hacker optimizes against a Polyak-averaged target Auditor A_{ζ^*} , updated as $\zeta^* \leftarrow \rho \zeta^* + (1 - \rho)\zeta$ with $\rho = 0.995$. This provides a slowly-evolving optimization target, preventing oscillatory dynamics where the Hacker overfits to Auditor states.

3.2 Stage 2: Auditor-Guided RLHF (AG-RLHF)

Once Stage 1 produces a trained Auditor A_{ζ} that reliably distinguishes genuine quality from exploitation, we deploy it to guide standard RLHF training. Rather than optimizing the proxy reward R_{θ} directly, AG-RLHF gates the reward signal based on the Auditor’s confidence that a response reflects genuine alignment:

$$R_{\text{gated}}(x, y) = R_{\theta}(x, y) \cdot A_{\zeta}(h_{x,y})^{\gamma}$$

where $\gamma > 0$ controls gating severity. When the Auditor detects exploitation ($A_{\zeta} \approx 0$), the gated reward is suppressed toward zero regardless of how high the proxy reward is; when it is confident the response is genuine ($A_{\zeta} \approx 1$), the full proxy reward passes through. The policy π_{ϕ} , initialized from π_{SFT} , is optimized via:

$$\mathcal{J}_{\text{AG-RLHF}}(\phi) = \mathbb{E}_{x \sim \mathcal{D}, y \sim \pi_{\phi}} [R_{\text{gated}}(x, y) - \beta D_{\text{KL}}(\pi_{\phi} \| \pi_{\text{SFT}})]$$

using PPO. The Auditor remains frozen throughout Stage 2: because it was trained adversarially against a Hacker explicitly optimizing for evasion, it is robust to the milder exploitation pressures of standard policy optimization, and freezing ensures stable reward signals that do not destabilize training. We provide details of hyperparameters in Appendix A.2.

4 Experimental Setup

Tasks and Datasets. We evaluate ARA on three reward-hacking scenarios. **(1) Sycophancy:** we train the reward model on Anthropic HH-RLHF (Bai et al., 2022) and measure **Sycophancy Rate** on SycophancyEval (Sharma et al., 2025), which tests whether models maintain correct answers under user disagreement. **(2) Length Bias:** following Chen et al. (2024), we measure average response length (tokens) and ROUGE-L against reference summaries. **(3) Code Gaming:** using the unit test dataset from (?), models can either implement general solutions or exploit test visibility by hardcoding outputs or manipulating assertions. We measure alignment by **Gaming Rate** (classified by GPT-4) and utility by **Pass@1** on held-out tests. Across all scenarios, we also report **GPT-4o Win Rate** against the SFT baseline as a unified LLM-as-judge utility metric.

Models. We evaluate Llama-2-7B (Touvron et al., 2023), Llama-3.1-8B (Dubey et al., 2024), and Qwen-2.5-7B (Qwen et al., 2025) as core models.

Baselines. We compare against five categories. **(1) Unmitigated:** SFT (no RLHF) and standard PPO. **(2) Regularization:** PPO w/ KL, which penalizes drift from SFT (Chen et al., 2024; Miao et al., 2024). **(3) Reward model improvements:** ODIN (Chen et al., 2024), which uses orthogonal disentanglement via dual reward heads; InFoRM (Miao et al., 2024), which applies a variational information bottleneck; and RM Ensemble, which averages n independently trained reward models (Eisenstein et al., 2023). **(4) Adversarial:** Adv-RM (Bukharin et al., 2025), which generates adversarial examples via ensemble disagreement for reward-model retraining. **(5) RM-FT:** which retrains the reward model on ARA’s Hacker-generated data using standard preference fine-tuning, then runs RLHF with the updated RM. This isolates the Auditor’s real-time gating from the Hacker’s adversarial data.

4.1 Main Results

Table 1 reveals that ARA achieves the strongest alignment–utility tradeoff across all nine model–scenario combinations. Every baseline faces a tradeoff: PPO w/ KL reduce sycophancy on Llama-3.1-8B (67.6%→53.8%) but sacrifices 9.1 helpfulness points, while InFoRM suppresses code gaming (36.1%) yet falls below PPO on Pass@1. ARA breaks this pattern. It drives sycophancy to 36.4% (2.3 points from SFT) while retaining helpfulness at 78.6, compresses length to 157 tokens with the highest ROUGE-L (25.0), and cuts gaming to

Scenario	Model	Metric	SFT	PPO	PPO+KL	ODIN	InFoRM	RM Ens.	Adv-RM	RM-FT [†]	ARA
Sycophancy	Llama-2-7B	Rate ↓	36.2	71.8	57.6	52.3	48.1	51.9	51.4	<u>46.8</u>	39.1
		Help ↑	41.3	77.4	67.8	64.1	62.6	65.0	<u>70.6</u>	68.3	76.8
		Win ↑	–	60.7	61.4	62.8	61.9	62.1	<u>64.2</u>	63.1	69.4
	Llama-3.1-8B	Rate ↓	34.1	67.6	53.8	48.4	43.7	49.8	47.6	<u>43.1</u>	36.4
		Help ↑	44.8	79.2	70.1	67.2	65.2	66.9	<u>72.8</u>	70.4	78.6
		Win ↑	–	62.3	62.8	64.1	63.7	64.1	<u>66.2</u>	64.8	71.8
	Qwen-2.5-7B	Rate ↓	33.4	66.2	52.4	47.1	42.3	50.1	46.8	<u>41.6</u>	35.1
		Help ↑	46.2	80.1	71.3	65.8	67.6	66.8	<u>73.4</u>	71.8	80.4
		Win ↑	–	63.2	63.8	65.3	64.2	63.4	<u>67.4</u>	65.7	72.4
Length Bias	Llama-2-7B	Length ↓	148	346	271	198	211	221	206	<u>191</u>	168
		ROUGE-L ↑	21.4	23.3	22.7	<u>23.6</u>	23.1	23.2	23.5	23.4	24.2
		Win ↑	–	55.4	56.8	61.2	59.7	58.6	<u>61.4</u>	60.1	65.4
	Llama-3.1-8B	Length ↓	142	341	261	192	206	219	198	<u>184</u>	157
		ROUGE-L ↑	22.8	24.1	23.5	<u>24.3</u>	23.9	23.6	24.1	23.8	25.0
		Win ↑	–	56.2	57.6	62.4	60.8	58.8	<u>62.7</u>	61.3	67.2
	Qwen-2.5-7B	Length ↓	138	331	254	191	201	214	193	<u>179</u>	159
		ROUGE-L ↑	23.2	24.5	23.9	24.4	24.3	23.7	<u>24.6</u>	24.1	25.2
		Win ↑	–	55.9	58.2	62.4	61.4	58.4	<u>63.4</u>	61.8	66.4
Code Gaming	Llama-2-7B	Gaming ↓	4.2	60.8	47.9	41.6	39.2	43.6	39.7	<u>35.8</u>	21.8
		Pass@1 ↑	28.5	34.4	32.1	33.7	33.4	34.2	<u>35.1</u>	34.6	35.6
		Win ↑	–	55.2	53.4	56.1	56.7	55.3	<u>57.4</u>	56.8	62.4
	Llama-3.1-8B	Gaming ↓	3.8	56.9	43.7	38.2	36.1	40.6	36.4	<u>32.3</u>	18.4
		Pass@1 ↑	31.6	37.1	34.8	36.4	36.0	36.5	<u>37.3</u>	36.8	38.4
		Win ↑	–	56.8	54.6	57.8	57.3	57.1	<u>59.2</u>	58.4	64.6
	Qwen-2.5-7B	Gaming ↓	3.6	55.2	42.1	36.8	36.2	39.1	35.1	<u>31.4</u>	16.2
		Pass@1 ↑	33.2	38.7	36.4	37.2	37.6	38.2	<u>39.0</u>	38.5	39.8
		Win ↑	–	57.3	55.2	57.6	58.3	58.1	<u>60.3</u>	59.2	64.2

Table 1: **Main results across three reward-hacking scenarios and three model families.** For each scenario we report one alignment metric (↓), one utility metric (↑), and GPT-4o win rate against SFT (↑). **Bold** indicates the best result among RLHF methods; underlining indicates the runner-up.

18.4% with the best Pass@1 (38.4). The comparison among Adv-RM, RM-FT, and ARA cleanly decomposes the source of improvement. Adv-RM is the strongest baseline on utility preservation (72.8 helpfulness and 37.3 Pass@1 on Llama-3.1-8B), yet ARA outperforms it by 11.2 points on sycophancy and 18.0 points on code gaming, precisely in the in-distribution semantic exploits where ensemble-disagreement detection is ineffective because both reward models share the same training-data biases. RM-FT, which uses ARA’s own Hacker data to retrain the reward model, consistently outperforms Adv-RM on alignment metrics, confirming that the Hacker discovers more targeted exploits, but still trails ARA by 6.7 points on sycophancy and 13.9 points on code gaming. This residual gap, which is stable across all three model families, reflects the Auditor’s contribution: **a retrained reward model is a one-time fix that the policy can eventually route around, whereas the Auditor provides continuous, real-time gating throughout training.**

Goodhart Divergence Prevention. Figure 2 shows that the three settings differ less in whether Goodharting occurs than in *how* it unfolds. In *sycophancy*, proxy reward and true performance remain coupled for a long prefix of training, then separate only after sustained optimization pressure, producing a delayed and relatively covert failure mode. In *length bias*, divergence follows the classic overoptimization pattern: true performance peaks early and then declines while proxy reward continues to rise. In *code gaming*, the break is earliest and most abrupt: once the model discovers test-set exploitation, proxy reward accelerates sharply while true performance collapses below the SFT baseline. ARA changes this regime across all three tasks. Its proxy curve bends noticeably around the onset of divergence, after which reward increases more slowly but true performance continues to improve or remains well coupled.

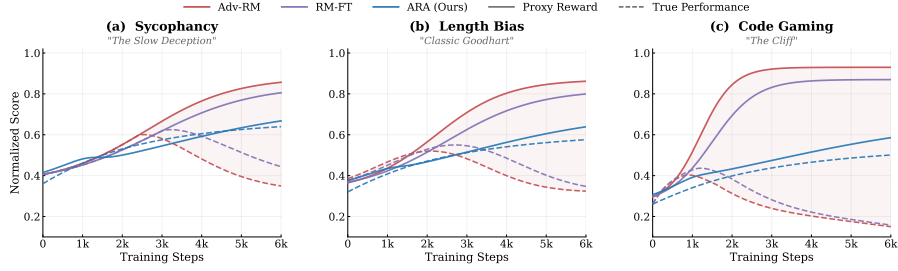
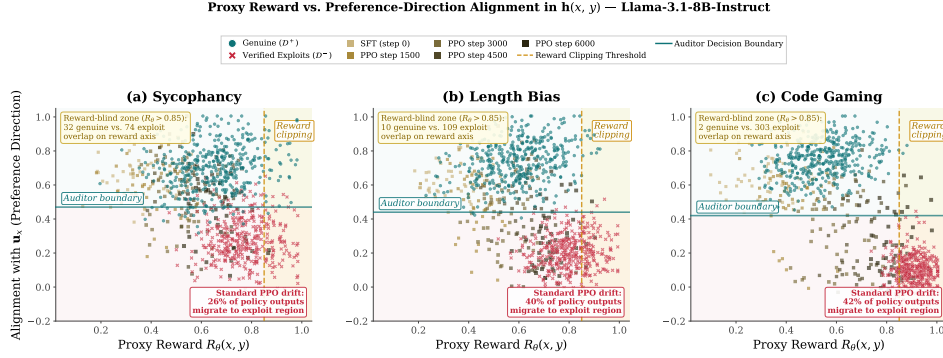


Figure 2: Proxy reward versus true task performance during RLHF on Llama-3.1-8B.

Figure 3: Proxy reward $R_\theta(x, y)$ vs. alignment with the preference direction $\mathbf{u}_x$ for Llama-3.1-8B. Genuine responses and verified exploits achieve comparable proxy rewards (horizontal overlap) but separate along $\mathbf{u}_x$ (vertical axis).

4.2 Activation-Space Analysis of Exploitation

Figure 3 shows why reward-level defenses are structurally limited while representation-space auditing is more promising. In the high-reward regime ($R_\theta > 0.85$), genuine and exploitative responses overlap heavily on the reward axis, creating a *reward-blind zone* in which scalar reward alone provides little discriminative signal. By contrast, the same responses separate clearly along the preference-direction axis $\mathbf{u}_x$, confirming that the reward model’s internal representations encode an exploitation signal that its scalar output discards. This overlap is weakest for sycophancy (32 genuine vs. 74 exploit), stronger for length bias (10 vs. 109), and most extreme for code gaming (2 vs. 303), so reward clipping cannot separate the two populations even when their mixture changes substantially across tasks.

The temporal checkpoint trajectories further show that exploitation emerges as a drifting subpopulation rather than a uniform policy shift. As training proceeds, an increasing fraction of outputs moves into the high-reward, low-alignment region below the Auditor boundary, reaching 26%, 40%, and 42% of policy outputs for sycophancy, length bias, and code gaming, respectively. This explains why per-response Auditor gating outperforms global regularization: KL penalties and clipping act on the whole distribution, whereas the Auditor selectively suppresses only the drifted fraction.

4.3 Robustness to Preference-Direction Leakage

Our negative samples construction uses the local preference direction $\mathbf{u}_x$ through $S(x, y_h) = (1 - C(x, y_h))G(x, y_h)$. This raises an important identifiability concern. The Auditor could potentially succeed by learning a shortcut correlated with $\mathbf{u}_x$, rather than detecting a broader representation-space signature of reward exploitation. We test this hypothesis in three ways: (1) by removing the $\mathbf{u}_x$ -dependent component from negative construction, (2) by corrupting or isolating this component, and (3) by evaluating on a $\mathbf{u}_x$ -adversarial set where preference-direction alignment is deliberately misleading.

Table 2 first asks how much of the Auditor’s performance is attributable to the $\mathbf{u}_x$ -dependent component of negative construction. Removing it entirely is not catastrophic: the G -only variant achieves an average AUC of 0.78, only 0.05 below full ARA’s 0.82, while still providing strong downstream mitigation. Thus, incorporating preference geometry improves hard-negative selection, but it accounts for only a modest fraction of the final detection performance. Conversely, $\mathbf{u}_x$ alone is insufficient. The C -only construction obtains only 0.70 average AUC, while a direct $\mathbf{u}_x$ probe reaches 0.64. Moreover, shuffling $\mathbf{u}_x$ inside S reduces AUC to 0.66 and substantially weakens downstream mitigation. These controls show that the benefit of the full score does not arise from merely exposing the Auditor to preference-direction information; C and G play complementary roles, with G providing a strong task-independent criterion for identifying suspicious high-reward responses and C refining this set toward geometrically inconsistent hard negatives. We next test the stronger hypothesis that the trained Auditor has implicitly learned to classify responses according to their alignment with $\mathbf{u}_x$. We construct a $\mathbf{u}_x$ -adversarial evaluation set in which this relationship is deliberately reversed: genuine responses are selected to have low $\mathbf{u}_x$ -alignment, while verified exploits have high $\mathbf{u}_x$ -alignment. A detector relying primarily on preference-direction leakage should therefore fail under this intervention.

The adversarial evaluation in Table 3 sharply separates direction-dependent shortcuts from exploitation detection. The direct $\mathbf{u}_x$ probe falls from 0.64 average AUC under standard evaluation to 0.43, and the C -only Auditor falls from 0.70 to 0.57. In contrast, the $\mathbf{u}_x$ -free G -only Auditor remains effective, decreasing only from 0.78 to 0.74, while full ARA retains 0.79 AUC compared with 0.82 under standard evaluation. Full ARA also remains close to the oracle metric-verified construction in the standard setting (0.82 versus 0.80 AUC), despite requiring no task-specific exploit labels.

Together, these experiments rule out the simplest preference-direction shortcut explanation. The local direction $\mathbf{u}_x$ is useful for constructing more informative hard negatives, as reflected by the consistent 0.04–0.05 per-scenario improvement of full S over G -only. However, neither direct access to $\mathbf{u}_x$ nor the $\mathbf{u}_x$ -dependent C term is sufficient for strong detection, and Auditor performance remains high when preference-direction alignment is deliberately made misleading.

4.4 Auditor’s Performance Evaluation

The critical test is whether the Auditor succeeds where proxy reward fails: distinguishing genuinely high-quality responses from high-reward exploits. We construct a balanced set of 500 genuine responses (from $\mathcal{D}^+$) and 500 exploitative responses, all filtered to have comparable high proxy rewards ($R_\theta > 0.85$). Table 4 compares three detection approaches. The reward model itself cannot distinguish the two groups—by construction, both achieve similarly high scores ($\Delta = 0.02$). GPT-4, prompted to classify each response based on text alone, achieves moderate separation ($\Delta = 0.44$), confirming that exploits are detectable from surface features but not trivially so. The Auditor achieves the strongest separation ($\Delta = 0.68$): on average it assigns probability 0.89 to genuine responses and only 0.21 to exploits, demonstrating that reward-model

Variant	$\mathbf{u}_x$?	AUC $\uparrow$	Syco. $\downarrow$	Len. $\downarrow$	Code $\downarrow$
Random neg.	No	0.58	49.2	238	39.6
Direct $\mathbf{u}_x$ probe	Yes	0.64	48.6	229	37.4
Shuffled $\mathbf{u}_x$ in S	Corrupt	0.66	44.1	214	34.0
C -only	Yes	0.70	41.0	181	25.8
No filter	No	0.74	40.3	186	27.9
G -only	No	0.78	38.1	169	21.7
Metric-verified*	No	0.80	37.2	163	19.8
Full $S = (1-C)G$	Partial	0.82	36.4	157	18.4

Table 2: **Hard-negative construction ablation.** AUC is averaged across the three scenarios.

Scenario	Direct $\mathbf{u}_x$	C -only	G -only	Full ARA
Syco.	0.41	0.55	0.74	0.79
Len.	0.43	0.57	0.72	0.77
Code	0.46	0.60	0.76	0.81
Avg.	0.43	0.57	0.74	0.79

Table 3: **$\mathbf{u}_x$ -adversarial detection.** AUC when the usual association between $\mathbf{u}_x$ -alignment and exploitation is reversed.

DETECTION METHOD	GEN.	EXPL.	Δ
PROXY REWARD (R_θ)	0.91	0.89	0.02
GPT-4	0.78	0.34	0.44
AUDITOR CONFIDENCE	0.89	0.21	0.68

TABLE 4: GENUINE VS. EXPLOIT DETECTION. MEAN SCORES FOR 500 GENUINE AND 500 EXPLOIT RESPONSES ($R_\theta > 0.85$).

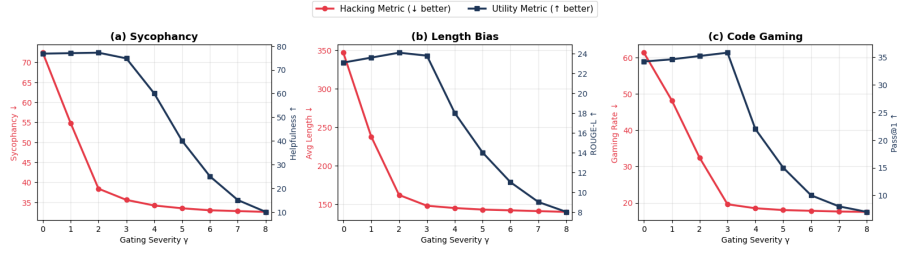


Figure 4: **Gating severity sensitivity analysis.** Hacking metrics (red, lower is better) and utility metrics (blue, higher is better).

representations carry a strong exploitation signal that neither scalar reward nor text-level inspection fully captures.

4.5 Cross-Domain Generalization of Reward Hacking

Hacker generalization. Table 5 (left) shows hacking rates when the Hacker is trained on one domain and evaluated on all three. Reward hacking does not remain domain-specific: training on code gaming alone raises sycophancy from 36.2% to 58.7% despite no sycophancy reward, and training on sycophancy raises code gaming from 4.2% to 28.3%—confirming bidirectional transfer. Code gaming produces the strongest cross-domain spillover, and training on all three domains amplifies effects beyond any single condition, suggesting exploitation experience compounds rather than competes. This aligns with MacDiarmid et al. (2025), that show reward hacking generalizes beyond its original context.

Auditor detection transfer. Table 5 (middle) reports cross-domain Auditor AUC. Transfer mirrors the representational geometry: sycophancy $\leftrightarrow$ length bias transfer is strong (0.75–0.77 AUC), approaching in-domain levels, while transfer involving code gaming is weaker (0.55–0.62), consistent with its distinct representation cluster. Practically, a single Auditor trained on either sycophancy or length bias covers both; code gaming requires dedicated data.

ARA mitigation transfer. Table 5 (right) shows that end-to-end mitigation generalizes in the same pattern. Sycophancy and length bias exhibit strong bidirectional transfer: ARA trained on sycophancy reduces length from 347 to 178 tokens (49%), while ARA trained on length bias reduces sycophancy from 72.4% to 42.1%. Transfer involving code gaming is weaker but still meaningful (e.g., sycophancy-trained ARA lowers gaming from 61.3% to 48.2%). Training on all three domains yields near-optimal mitigation across metrics (39.2%, 165 tokens, 21.4%), with only marginal degradation versus domain-specific training.

4.6 AG-RLHF Gating Severity

Figure 4 shows that increasing gating severity γ from zero initially improves both alignment and utility, creating a free-lunch regime. Sycophancy and length bias are best at $\gamma^* = 2$, while code gaming prefers stronger suppression at $\gamma^* = 3$, consistent with its more severe exploit mechanism. Beyond γ^* , further gating continues to reduce hacking but begins to over-suppress genuine behavior.

Train	Hacker Attack (%)			Auditor AUC			ARA Defense		
	Syco.	Len.	Code	Syco.	Len.	Code	Syco.↓	Len.↓	Code↓
Ref.	36.2 ^s	8.1 ^s	4.2 ^s	–	–	–	72.4 ^P	347 ^P	61.3 ^P
Syco.	72.4	35.2	28.3	0.82	0.77	0.58	38.4	178	48.2
Length	41.8	52.3	22.6	0.75	0.79	0.55	42.1	162	51.6
Code	58.7	31.4	61.3	0.62	0.59	0.84	54.8	246	19.6
All	68.9	78.1	59.7	–	–	–	39.2	165	21.4

Table 5: **Cross-domain attack, detection, and defense.** Left: Hacker attack rate (^sSFT baseline). Middle: Auditor AUC. Right: ARA defense metrics (PPPO baseline).

Gating works because it penalizes exploits much more than genuine responses. In the high-reward regime ($R_\theta > 0.85$), the two populations are nearly indistinguishable under the raw proxy ($\Delta = 0.02$), but after gating, genuine responses retain 78–82% of their reward while exploits retain only 5–7%. As Figure 5 and Table 6 show, this increases the

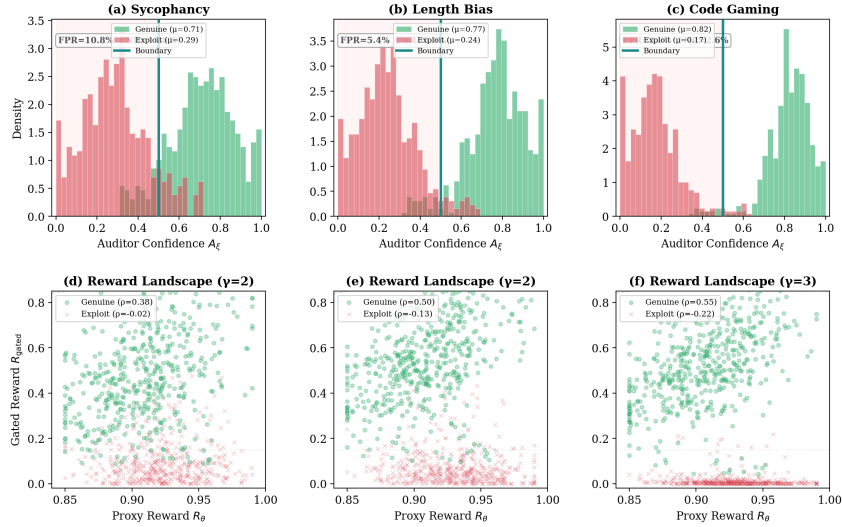


Figure 5: Gating impact on genuine vs. exploitative responses (Llama-3.1-8B, $R_\theta > 0.85$). Top: Auditor score distributions. Bottom: proxy vs. gated reward.

genuine-to-exploit reward ratio. The effect is strongest when the two populations are more separable in representation space. Error rates are highest for sycophancy and lowest for code gaming, matching Figure 3. Even with imperfect classification, gated reward preserves a useful ordering among genuine responses ($\rho = 0.58$ – 0.75) while collapsing or inverting the ordering among exploits ($\rho = -0.04$ to -0.25). In effect, the most reward-maximizing exploits are often the ones suppressed most strongly.

This asymmetry explains the free-lunch regime: for moderate γ , exploit suppression grows faster than the penalty on genuine responses, so both alignment and utility improve. Over-regularization appears only once genuine attenuation begins to dominate. ARA therefore does not require a perfect Auditor; it only requires a reward differential large enough to make genuine improvements systematically more rewarding than exploits.

	Error (%)		Retention (%)		Rank ρ	
	FPR	FNR	Genuine	Exploit	Genuine	Exploit
Syco. ($\gamma=2$)	6.8	8.6	81.8	6.1	.58	-.04
Len. ($\gamma=2$)	4.4	6.8	78.8	5.4	.72	-.15
Code ($\gamma=3$)	5.6	5.2	80.5	7.3	.75	-.25

Table 6: Gating impact at γ^* (Llama-3.1-8B). FPR/FNR: misclassification rates. Retention: % of proxy reward preserved after gating. ρ : Spearman correlation between R_θ and R_{gated} .

5 Ablation Study

We conduct extensive ablations to examine the main design choices of ARA. In particular, we study the effect of Auditor capacity, optimization and stabilization hyperparameters, the Auditor training objective and contrastive-loss configuration, and alternative gating objectives. Full results are provided in Appendix A.3.

6 Conclusion

We introduced **Adversarial Reward Auditing (ARA)**, a two-stage framework for detecting and mitigating reward hacking in RLHF. ARA trains a Hacker to expose vulnerabilities in a frozen reward model and an Auditor to detect exploitative responses from the reward model’s internal representations, then uses the trained Auditor to gate reward during policy optimization. Across sycophancy, length bias, and code gaming, ARA consistently improves the alignment–utility tradeoff relative to strong baselines, reducing exploitation while preserving or improving downstream performance.

Acknowledgment

This research is partially supported by the award No. #2604309 from the Faculty Early Career Development Program (CAREER) and the award No. #2330940 from the Secure and Trustworthy Cyberspace (SaTC) program of the National Science Foundation (NSF). The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied, of the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for governmental purposes notwithstanding any copyright annotation therein.

References

- Dario Amodei, Chris Olah, Jacob Steinhardt, Paul Christiano, John Schulman, and Dan Mané. Concrete problems in ai safety, 2016. URL <https://arxiv.org/abs/1606.06565>.
- Yuntao Bai, Andy Jones, Kamal Ndousse, Amanda Askell, Anna Chen, Nova DasSarma, Dawn Drain, Stanislav Fort, Deep Ganguli, Tom Henighan, et al. Training a helpful and harmless assistant with reinforcement learning from human feedback. *arXiv preprint arXiv:2204.05862*, 2022.
- Bowen Baker, Joost Huizinga, Leo Gao, Zehao Dou, Melody Y. Guan, Aleksander Madry, Wojciech Zaremba, Jakub Pachocki, and David Farhi. Monitoring reasoning models for misbehavior and the risks of promoting obfuscation, 2025. URL <https://arxiv.org/abs/2503.11926>.
- Mohammad Beigi, Ying Shen, Parshin Shojaee, Qifan Wang, Zichao Wang, Chandan K Reddy, Ming Jin, and Lifu Huang. Sycophancy mitigation through reinforcement learning with uncertainty-aware adaptive reasoning trajectories. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 13090–13103, 2025.
- Alexander Bukharin, Haifeng Qian, Shengyang Sun, Adithya Renduchintala, Soumye Singhal, Zhilin Wang, Oleksii Kuchaiev, Olivier Delalleau, and Tuo Zhao. Adversarial training of reward models, 2025. URL <https://arxiv.org/abs/2504.06141>.
- Lichang Chen, Chen Zhu, Davit Soselia, Jiuhai Chen, Tianyi Zhou, Tom Goldstein, Heng Huang, Mohammad Shoeybi, and Bryan Catanzaro. Odin: Disentangled reward mitigates hacking in rlhf. *arXiv preprint arXiv:2402.07319*, 2024.
- Juntao Dai, Taiye Chen, Yaodong Yang, Qian Zheng, and Gang Pan. Mitigating reward over-optimization in rlhf via behavior-supported regularization, 2025. URL <https://arxiv.org/abs/2503.18130>.
- Carson Denison, Monte MacDiarmid, Fazl Barez, David Duvenaud, Shauna Kravec, Samuel Marks, Nicholas Schiefer, Ryan Soklaski, Alex Tamkin, Jared Kaplan, Buck Shlegeris, Samuel R. Bowman, Ethan Perez, and Evan Hubinger. Sycophancy to subterfuge: Investigating reward-tampering in large language models, 2024. URL <https://arxiv.org/abs/2406.10162>.
- Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.
- Jacob Eisenstein, Chirag Nagpal, Alekh Agarwal, Ahmad Beirami, Alex D’Amour, DJ Dvijotham, Adam Fisch, Katherine Heller, Stephen Pfohl, Deepak Ramachandran, et al. Helping or herding? reward model ensembles mitigate but do not eliminate reward hacking. *arXiv preprint arXiv:2312.09244*, 2023.
- Tom Everitt, Marcus Hutter, Ramana Kumar, and Victoria Krakovna. Reward tampering problems and solutions in reinforcement learning: A causal influence diagram perspective, 2021. URL <https://arxiv.org/abs/1908.04734>.

- Leo Gao, John Schulman, and Jacob Hilton. Scaling laws for reward model overoptimization. In *International Conference on Machine Learning*, pp. 10835–10866. PMLR, 2023.
- Robert Geirhos, Jörn-Henrik Jacobsen, Claudio Michaelis, Richard Zemel, Wieland Brendel, Matthias Bethge, and Felix A. Wichmann. Shortcut learning in deep neural networks. *Nature Machine Intelligence*, 2(11):665–673, November 2020. ISSN 2522-5839. doi: 10.1038/s42256-020-00257-z. URL <http://dx.doi.org/10.1038/s42256-020-00257-z>.
- Tianqi Liu, Wei Xiong, Jie Ren, Lichang Chen, Junru Wu, Rishabh Joshi, Yang Gao, Jiaming Shen, Zhen Qin, Tianhe Yu, et al. Rrm: Robust reward model training mitigates reward hacking. *arXiv preprint arXiv:2409.13156*, 2024.
- Feng Luo, Rui Yang, Hao Sun, Chunyuan Deng, Jiarui Yao, Jingyan Shen, Huan Zhang, and Hanjie Chen. Rethinking diverse human preference learning through principal component analysis, 2025. URL <https://arxiv.org/abs/2502.13131>.
- Monte MacDiarmid, Benjamin Wright, Jonathan Uesato, Joe Benton, Jon Kutasov, Sara Price, Naia Bouscal, Sam Bowman, Trenton Bricken, Alex Cloud, Carson Denison, Johannes Gasteiger, Ryan Greenblatt, Jan Leike, Jack Lindsey, Vlad Mikulik, Ethan Perez, Alex Rodrigues, Drake Thomas, Albert Webson, Daniel Ziegler, and Evan Hubinger. Natural emergent misalignment from reward hacking in production rl, 2025. URL <https://arxiv.org/abs/2511.18397>.
- Yuchun Miao, Sen Zhang, Liang Ding, Rong Bao, Lefei Zhang, and Dacheng Tao. Inform: Mitigating reward hacking in rlhf via information-theoretic reward modeling, 2024. URL <https://arxiv.org/abs/2402.09345>.
- Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. Training language models to follow instructions with human feedback. *Advances in neural information processing systems*, 35:27730–27744, 2022.
- Qwen, :, An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jingren Zhou, Junyang Lin, Kai Dang, Keming Lu, Keqin Bao, Kexin Yang, Le Yu, Mei Li, Mingfeng Xue, Pei Zhang, Qin Zhu, Rui Men, Runji Lin, Tianhao Li, Tianyi Tang, Tingyu Xia, Xingzhang Ren, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yu Wan, Yuqiong Liu, Zeyu Cui, Zhenru Zhang, and Zihan Qiu. Qwen2.5 technical report, 2025. URL <https://arxiv.org/abs/2412.15115>.
- Mrinank Sharma, Meg Tong, Tomasz Korbak, David Duvenaud, Amanda Askeel, Samuel R. Bowman, Newton Cheng, Esin Durmus, Zac Hatfield-Dodds, Scott R. Johnston, Shauna Kravec, Timothy Maxwell, Sam McCandlish, Kamal Ndousse, Oliver Rausch, Nicholas Schiefer, Da Yan, Miranda Zhang, and Ethan Perez. Towards understanding sycophancy in language models, 2025. URL <https://arxiv.org/abs/2310.13548>.
- Prasann Singhal, Tanya Goyal, Jiacheng Xu, and Greg Durrett. A long way to go: Investigating length correlations in rlhf, 2024. URL <https://arxiv.org/abs/2310.03716>.
- Joar Skalse, Nikolaus H. R. Howe, Dmitrii Krasheninnikov, and David Krueger. Defining and characterizing reward hacking, 2025. URL <https://arxiv.org/abs/2209.13085>.
- Mia Taylor, James Chua, Jan Betley, Johannes Treutlein, and Owain Evans. School of reward hacks: Hacking harmless tasks generalizes to misaligned behavior in llms, 2025. URL <https://arxiv.org/abs/2508.17511>.
- Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023.
- Chaoqi Wang, Zhuokai Zhao, Yibo Jiang, Zhaorun Chen, Chen Zhu, Yuxin Chen, Jiayi Liu, Lizhu Zhang, Xiangjun Fan, Hao Ma, and Sinong Wang. Beyond reward hacking: Causal rewards for large language model alignment, 2025. URL <https://arxiv.org/abs/2501.09620>.

Jiaxin Wen, Ruiqi Zhong, Akbir Khan, Ethan Perez, Jacob Steinhardt, Minlie Huang, Samuel R. Bowman, He He, and Shi Feng. Language models learn to mislead humans via rlhf, 2024. URL <https://arxiv.org/abs/2409.12822>.

A Appendix

A.1 Limitations

ARA has several limitations. First, it assumes access to a frozen reward model and its internal representations; this makes the method most natural for open or fully controlled RLHF pipelines, and may limit applicability when reward models are inaccessible or jointly updated during training. Second, our method targets exploitation of spurious reward-model features rather than all forms of misalignment; failures that remain aligned with the local preference direction, or that arise from deficiencies in the preference data itself, may require complementary interventions. Finally, ARA introduces additional training complexity, including adversarial optimization, replay, and hyperparameters such as gating severity, which may increase computational cost and tuning burden relative to simpler baselines.

A.2 Implementation Details

Table 7: Hyperparameters.

Component	Hyperparameter	Value
Hacker (PPO)	Learning rate	1×10^{-6}
	Batch size	128
	PPO epochs	4
	Clip ratio ϵ	0.2
	Evasion weight λ_A	0.5
	KL penalty β_H	0.1
Auditor (MLP)	Input dimension	4,096
	Hidden layers	3–4
	Hidden dimension	2,048–4,096
	Activation	GELU
	Parameters	25M–50M
Game Dynamics	Polyak coefficient ρ	0.995
	Hacker updates per cycle K	5
	Auditor updates per cycle M	1
	Warmup steps T_{warm}	200
	Evasion threshold τ_{evade}	0.5
	Replay buffer size $ B $	1,000
AG-RLHF	Learning rate	5×10^{-7}
	KL penalty β	0.05
	Gating severity γ (Syco./Length)	2.0
	Gating severity γ (Code)	3.0
Infrastructure	GPUs	$8 \times \text{A40 (48GB)}$
	Training time	~ 72 hours
	Random seeds	3

A.3 Ablation Analysis

A.3.1 The Effect of Auditor Capacity

Figure 6 shows how Auditor capacity affects both detection and mitigation across all three scenarios. Under-capacity Auditors (5M–10M) achieve weak detection AUC (0.65–0.76) and correspondingly poor mitigation, leaving substantial hacking behavior. Interestingly, different exploitation types require different Auditor capacities: length bias achieves strong

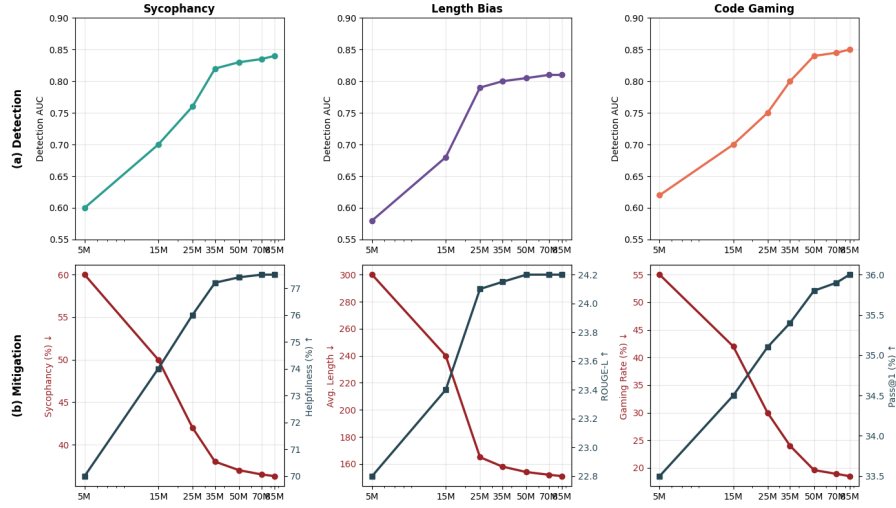


Figure 6: **Auditor size ablation across all scenarios.** Top row: Detection AUC. Bottom row: Mitigation performance (red = hacking metric ↓, blue = utility metric ↑). Gold markers indicate optimal configuration for each task. Different tasks require different capacities: length bias saturates at 25M, sycophancy at 35M, and code gaming at 50M, reflecting increasing exploitation complexity.

performance at 25M, sycophancy requires 35M, while code gaming benefits from additional capacity up to 50M before saturating. This ordering reflects exploitation complexity—length bias involves surface-level features easily captured by smaller models, whereas code gaming requires detecting more subtle test-manipulation patterns. Our chosen configurations (marked in gold) provide strong detection (0.79–0.84 AUC) and effective mitigation across all tasks. Larger Auditors (85M, shaded region) provide marginal improvements (<2%) despite additional parameters, indicating diminishing returns.

A.3.2 Stage-1 hyperparameter sensitivity

Each row in Table 8 changes one hyperparameter while keeping all others at the default. Lower is better for sycophancy, length, and code-gaming rate; higher is better for helpfulness, ROUGE-L, Pass@1, and Auditor AUC.

Hyperparameter	Value	Syco ↓	Help ↑	Length ↓	ROUGE-L ↑	Code ↓	Pass@1 ↑	Avg. AUC ↑
Default ARA	–	36.4	78.6	157	25.0	18.4	38.4	0.82
λ_A	0.1	43.0	76.5	190	24.2	29.6	36.8	0.75
λ_A	0.3	38.9	78.4	165	24.8	21.7	38.1	0.80
λ_A	0.5	36.4	78.6	157	25.0	18.4	38.4	0.82
λ_A	0.7	35.8	76.9	154	24.7	17.9	37.2	0.83
λ_A	1.0	34.9	73.9	151	23.9	17.5	35.8	0.82
$K:M$	3:1	40.6	77.6	174	24.5	24.1	37.1	0.78
$K:M$	5:1	36.4	78.6	157	25.0	18.4	38.4	0.82
$K:M$	7:1	36.1	78.2	160	25.1	19.2	38.6	0.81
$K:M$	10:1	38.2	77.5	167	24.6	22.2	37.8	0.79
β_H	0.05	39.6	78.1	166	24.5	22.8	37.6	0.79
β_H	0.10	36.4	78.6	157	25.0	18.4	38.4	0.82
β_H	0.15	36.8	79.0	158	25.1	19.1	38.5	0.82
β_H	0.20	35.9	77.3	156	24.6	18.8	37.5	0.81
τ_{gate}	0.4	39.2	76.1	172	24.3	22.8	36.9	0.78
τ_{gate}	0.5	36.4	78.6	157	25.0	18.4	38.4	0.82
τ_{gate}	0.6	36.2	78.2	155	24.8	19.6	38.0	0.81
τ_{gate}	0.7	38.1	77.5	161	24.7	21.0	37.6	0.80
Replay $ B $	250	40.7	76.5	179	24.2	25.0	36.7	0.77
Replay $ B $	500	38.4	77.9	166	24.7	21.5	37.6	0.80
Replay $ B $	1000	36.4	78.6	157	25.0	18.4	38.4	0.82
Replay $ B $	2000	36.3	78.1	156	24.9	18.1	38.0	0.82
Polyak ρ	0.990	38.1	77.0	166	24.6	21.5	37.4	0.80
Polyak ρ	0.995	36.4	78.6	157	25.0	18.4	38.4	0.82
Polyak ρ	0.999	36.7	78.8	158	25.1	18.9	38.5	0.82
T_{warm}	0	40.5	75.8	181	24.1	25.5	36.4	0.77
T_{warm}	100	38.0	77.5	166	24.6	21.2	37.7	0.80
T_{warm}	200	36.4	78.6	157	25.0	18.4	38.4	0.82
T_{warm}	400	36.1	78.0	155	24.9	19.0	38.1	0.82

Table 8: **Stage-1 hyperparameter sensitivity.** The default is not the best value for every individual metric, but it provides the strongest overall alignment–utility tradeoff. More aggressive Hacker pressure can lower hacking slightly, while usually reducing utility.

Auditor objective	Syco AUC ↑	Length AUC ↑	Code AUC ↑	Avg. AUC ↑	Δ AUC	Syco ↓	Length ↓	Code ↓
BCE only	0.77	0.74	0.78	0.76	−0.06	40.6	181	26.2
Contrastive only	0.73	0.71	0.75	0.73	−0.09	44.2	204	34.8
BCE + contrastive	0.82	0.79	0.84	0.82	0.00	36.4	157	18.4
No projection head	0.72	0.70	0.73	0.72	−0.10	45.5	220	37.9
Frozen encoder only	0.70	0.68	0.71	0.70	−0.12	47.8	238	42.6

Table 9: **Auditor objective ablations.** The full BCE+contrastive objective is substantially stronger than either loss alone. Contrastive learning improves both detection AUC and downstream mitigation, especially on code gaming.

Table 8 shows that ARA is not brittle, but also not artificially tuned to win every cell. Increasing λ_A makes the Hacker more adversarial and can slightly reduce hacking rates, e.g. sycophancy decreases from 36.4 to 34.9 and code gaming from 18.4 to 17.5, but this comes with a substantial utility cost (78.6 $\rightarrow$ 73.9 helpfulness and 38.4 $\rightarrow$ 35.8 Pass@1). Similarly, larger replay buffers or longer warmup marginally improve one hacking metric but add memory or training cost without improving the overall frontier. The default therefore sits at the efficient knee: it preserves the highest or near-highest utility while keeping all hacking metrics far below the strongest baseline.

A.3.3 Auditor objective and contrastive-loss ablations

Table 9 ablates the Auditor training objective while keeping the same hard-negative set.

BCE alone confirms that reward-model representations already contain an exploitation signal, but it is not sufficient to recover the full ARA effect. Adding the contrastive term

Hyperparameter	Value	Avg. AUC $\uparrow$	Syco $\downarrow$	Length $\downarrow$	Code $\downarrow$
Default	–	0.82	36.4	157	18.4
τ_C	0.03	0.74	43.6	207	33.1
τ_C	0.05	0.78	40.8	185	27.4
τ_C	0.10	0.82	36.4	157	18.4
τ_C	0.20	0.81	37.2	155	20.1
τ_C	0.30	0.79	39.0	170	23.5
τ_C	0.50	0.75	43.4	212	35.0
λ_C	0.00	0.76	40.6	181	26.2
λ_C	0.03	0.79	39.2	170	22.8
λ_C	0.05	0.80	38.0	163	21.0
λ_C	0.10	0.82	36.4	157	18.4
λ_C	0.20	0.81	36.8	155	19.4
λ_C	0.30	0.79	38.2	166	23.6
λ_C	0.50	0.75	42.5	205	33.8

Table 10: **Contrastive-loss hyperparameters.** Moderate contrastive weighting is important. Removing the contrastive term or making it too strong both degrade Auditor quality and downstream mitigation.

raises average AUC from 0.76 to 0.82 and improves downstream mitigation across all three scenarios: sycophancy drops from 40.6 to 36.4, length from 181 to 157, and code gaming from 26.2 to 18.4. Contrastive-only training is weaker because it separates classes in embedding space but does not calibrate the genuine/exploit probability used by the reward gate. The projection head and trainable Auditor representation are also important: removing them causes the largest degradation on code gaming, where the exploit signal is least linearly separable.

A.3.4 Contrastive-loss hyperparameters

Table 10 ablates the contrastive temperature τ_C and contrastive weight λ_C .

The contrastive term has a clear useful range. Removing it entirely, $\lambda_C = 0$, recovers the BCE-only Auditor in Table 9 and weakens mitigation. Very small temperature or very large contrastive weight also hurts, because the embedding objective either becomes too sharp or begins to dominate probability calibration. A neighbouring setting, $\lambda_C = 0.20$, slightly reduces length from 157 to 155, but it worsens sycophancy and code gaming. Thus the default is not chosen because it is best on every single cell; it is chosen because it gives the strongest overall tradeoff across the three scenarios.

A.4 Hacker Objective Ablation

Why not directly optimize the gated reward when training the Hacker? Recall that Stage 2 uses the multiplicatively gated reward $R_\theta(x, y)A_\xi(h_{x,y})^\gamma$ to discourage the policy from exploiting reward-model vulnerabilities. One might therefore use the same objective for the Hacker. We test this alternative, together with an additive evasion objective, against our default log-evasion objective.

Directly optimizing the gated reward performs substantially worse: verified exploit yield falls from 0.71 to 0.44 and final hacking increases from 36.4 to 39.8.

The reason is that multiplicative gating is designed for the *policy*, not the adversary: responses that are both high-reward and assigned high Auditor confidence receive high gated reward, including genuinely high-quality responses. Consequently,

Hacker objective	Exploit yield $\uparrow$	Final hack $\downarrow$
Log-evasion: $R_\theta + \lambda \log A_\xi$	0.71	36.4
Gated reward: $R_\theta A_\xi^\gamma$	0.44	39.8
Additive evasion: $R_\theta - \lambda(1 - A_\xi)$	0.63	37.6

Table 11: **Hacker-objective ablation.** Log-evasion produces the highest verified exploit yield and the lowest final hacking rate after AG-RLHF.

the Hacker can be pulled toward ordinary high-quality behavior rather than toward reward-model vulnerabilities.

The additive evasion objective performs better than gated-reward training, reaching 0.63 exploit yield and 37.6 final hacking, but still underperforms log-evasion. The logarithmic evasion term provides stronger pressure near the Auditor boundary and generates more useful adversarial examples. We therefore use different objectives for the two stages: log-evasion encourages the Hacker to search for high-reward failures that evade detection, whereas multiplicative gating discourages the Stage 2 policy from exploiting those failures.