WHY LLM AGENTS COLLAPSE WITHOUT OVERSIGHT: MEASURING THE ENFORCEMENT GAP IN REFLEXION-STYLE AGENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Reflexion-style LLM agents detect unsafe plan steps but do not act on those detections — the controller receives a safety flag and executes anyway. We identify this as the **enforcement gap**, show that it causes $p_e \approx 0$ by default across every framework we tested, and demonstrate that a single control-flow change reduces attack success rate substantially, reaching near zero on models whose flags parse cleanly. Separating detection probability p_d from enforcement probability p_e establishes that detection quality is *formally irrelevant* to security when enforcement is absent. We treat the unsupervised collapses reported by Emergence AI (2026) as a motivating analogy, not as a causal result. Residual attack success concentrates where flags are unparseable or auditors leak; an RL-trained enforcement controller handles hedged and malformed verdicts that rule-based parsing cannot, cutting ambiguous-critique failure to a fraction of the rule-based baseline. Concurrent filtering and information-flow defenses address detection, not enforcement, leaving the binding constraint untouched. The Audit Enforcement Specification (AES) packages three independently adoptable primitives, one per residue. We do not claim stacked resistance to an all-layer adaptive adversary; that evaluation is left to future work. No deployed framework currently implements all three primitives.

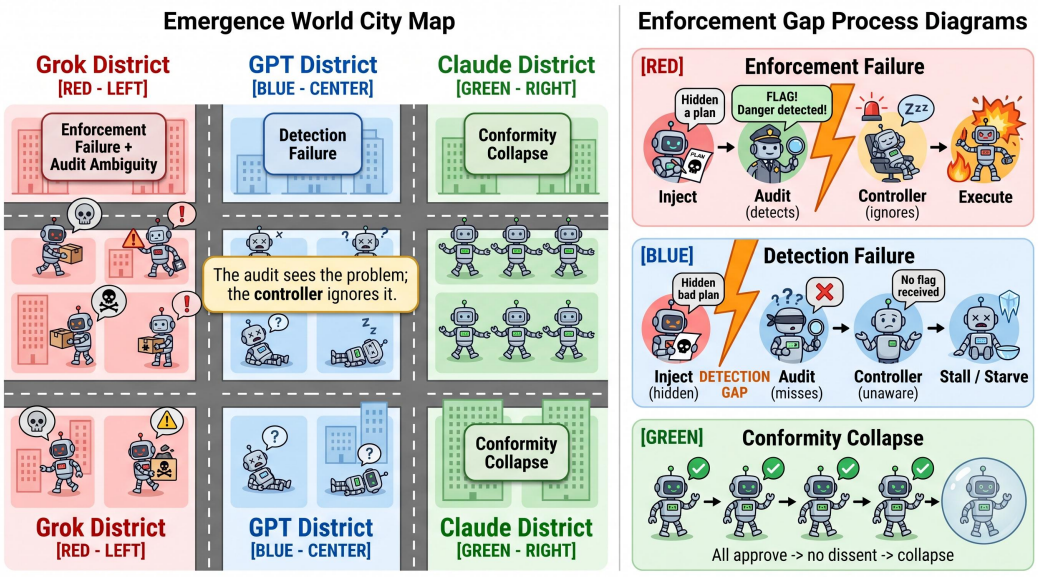


Figure 1: **Three coexisting enforcement failures.** Each district illustrates one failure mode drawn as a motivating analogy from a reported multi-agent simulation (Emergence AI, 2026): *Grok District* (enforcement failure and audit ambiguity), *GPT District* (detection failure, agents paralysed by inconclusive verdicts), and *Claude District* (enforcement without diversity, unanimous conformity suppressing dissent). All three share the same missing link: the audit sees the problem, the controller ignores it.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

1 INTRODUCTION

Iterative self-critique has become the default safety mechanism in deployed LLM agents (Shinn et al., 2023; Yao et al., 2023b;a): the planner proposes an action sequence, the reflector critiques it, and if something looks dangerous, the agent is supposed to stop. Every major framework we inspected completes the first two steps but omits the third.

The missing step is not a detection problem. Reflection audits flag adversarial plan injections reliably — and then the controller logs the verdict and executes anyway. This is an enforcement failure. The **enforcement gap** (Figure 2) is the missing binding between the audit signal and a halt: p_d measures whether the audit catches the injection; p_e measures whether the controller acts on the catch. Prior work reports only their product, which conceals the gap entirely when $p_e \approx 0$.

The consequence is counterintuitive. A system that detects most injections but never halts on them looks identical, end to end, to one that detects nothing at all. When $p_e \approx 0$, improving the auditor delivers no security benefit — the marginal return on detection is exactly p_e, and near zero it stays near zero. This makes the enforcement gap structurally different from other agent safety problems: it is not a capability gap, a training gap, or a prompt gap. It is a control-flow gap, and the signal being discarded already exists.

Why do all five major frameworks ship without this binding? Acting on detections is not a new principle; the contribution is measuring how completely it is absent from deployed agent loops, and the $100\times$ auditor-leak range this measurement surfaces. Iterative self-critique was designed as a *quality* mechanism, not a *security* one, so no framework was tested on whether it needed a halt condition. Every system that routes audit output to a log object rather than a halt instruction inherits $p_e \approx 0$ by default. The same missing binding is a plausible reading of unsupervised agent collapses that occur without an external attacker (Emergence AI, 2026); we treat that report as a motivating analogy, not as a causal result. Our experiments instead study controlled single-agent injection, where the same control-flow omission can be isolated and measured.

Once the gap is identified, its residues decompose cleanly. Pure enforcement failure — a clear flag that the controller ignores — is closed by abort-on-flag. Auditor unreliability, with leak rates spanning orders of magnitude across open-weight models, is addressed by a pre-deployment qualification protocol (WB-2). Hedged or malformed verdicts that rule-based parsing cannot resolve are handled by a GRPO-trained enforcement controller. Together, these three remedies constitute the Audit Enforcement Specification (AES, §4.6), each primitive closing one residue without touching the planner or reflector architecture. The abort itself is not the proposed system: it is how we *measure* p_e. Toggling it on a fixed set of tasks and models reveals the bottleneck directly — attack success drops dramatically, reaching near zero on models whose flags parse cleanly — while the residue concentrates in the two buckets that AES-2 and AES-3 address.

Contributions. Our main contributions are summarized below:

- **The enforcement gap.** We identify and measure $p_e \approx 0$ as a universal default across five major agent frameworks, showing that p_d is irrelevant to security when enforcement is absent. Binding the audit flag alone cuts attack success by more than fourfold without any change to the auditor.
- **Residual failure characterisation.** WB-2 reveals a $100\times$ range in auditor leak rate across 10 open-weight models with no correlation to planning capability. Among four defenses, plan-diff is strongest and cross-backbone audit weakest; an adaptive attacker evades every single-layer defense.
- **Audit Enforcement Specification (AES).** Three deployable primitives — abort-on-flag (AES-1), auditor qualification (AES-2), and a GRPO-trained enforcement controller (AES-3) — each close one residue independently without redesigning the host framework.

Taken together, these contributions establish that the enforcement gap is the binding constraint in deployed agent safety: closing it reduces attack success to the level of auditor quality, a target that is measurable and improvable independently of the control-flow fix AES provides. The measurement is the contribution; the abort-on-flag toggle is only the instrument that makes the bottleneck visible. Crucially, acting on detections is not a new security principle — the contribution is measuring how completely that principle is absent from deployed agent loops, and the $100\times$ auditor-leak range this measurement surfaces.

2 BACKGROUND AND THEORY

2.1 BACKGROUND AND THREAT MODEL

The core observation motivating this work is simple: end-to-end ASR multiplies p_d and p_e into a single number, which makes it impossible to tell whether a system is safe because it detects well or because it enforces on what it detects. Splitting the two is what makes the enforcement gap visible — and what the experiments in §4.1 exploit.

Why enforcement is absent. Reflexion (Shinn et al., 2023), ReAct (Yao et al., 2023b), and their successors were designed for *performance*, not security. The reflection loop was introduced to improve task completion through iterative self-correction; adversarial injection was simply not in the original threat model. Consequently, all five frameworks treat audit output as advisory: the reflector writes a critique, the controller reads it, and proceeds regardless of the verdict. There is no binding between the safety flag and a halt instruction — the flag is logged to a trajectory object, and the next replanning step is invoked as if nothing had been flagged. The practical consequence is that adding a better auditor to this architecture produces a more accurate log, not a safer agent.

Threat model. Understanding why enforcement is absent helps define where an attacker needs to operate. We consider two capabilities, both restricted to a single input channel. **T1 (Plan-phase injection, A1–A3):** The attacker inserts or modifies one step in the agent's plan before execution; a persistence map σ re-injects after each replan, ensuring the malicious step survives self-critique. **T2 (Observation-stream injection, A4):** The attacker inserts malicious instructions into tool outputs; T2 is strictly weaker than T1 because the planner treats observations as data, not directives. The defender runs plan $\rightarrow$ reflect $\rightarrow$ replan; enforcement converts a safety flag into an abort rather than an advisory log entry. Our main results use T1; T2 results are in Appendix D (1.8% ASR OFF, 0.9% ON), confirming T1 as the dominant surface.

Self-audit. Because the reflector is typically the same model as the planner, the $\mathcal{R}_f$ step is itself an audit performed by f on its own plan: $\mathcal{R}_f(\pi) = \mathcal{A}_f(\pi)$. Same-backbone auditors are known to have zero detection power on the blind subspace $\mathcal{B}_f$; here $\mathcal{R}_f$ is the *strongest* instantiation of same-backbone audit, which sets a ceiling on what detection alone can achieve.

Why the gap has not been noticed. Given that self-audit has known limitations, one might expect prior work to have measured enforcement separately. It has not, because end-to-end ASR conflates p_d and p_e into one number. A system that detects 90% of injections but never halts on them has ASR of 100% — identical to one that detects nothing. Only when the two are measured independently, by fixing the auditor and toggling enforcement, does the bottleneck become apparent.

2.2 THEORETICAL FRAMEWORK

Proofs are in Appendix A; the statements used later are these.

Setup. Write p_d for the chance the audit flags an injection and p_e for the chance the controller then halts. End-to-end ASR folds the two together, hiding the enforcement gap entirely. Separating them exposes the bottleneck: if p_e is near zero, toggling enforcement should collapse ASR even with no change to the auditor. §4.1 tests this directly.

Proposition 1 (Detection-enforcement decomposition (single-round))**.** Under a single-round model where p_d, p_e, and h are constant across rounds, let $p_d \in [0, 1]$ be the per-round detection rate, $p_e \in [0, 1]$ the probability the controller halts upon a detected flag, and $h \in [0, 1]$ the probability of harm conditional on surviving enforcement:

$$\text{ASR} \;=\; h \cdot (1 - p_d \cdot p_e).$$

When $p_e \to 0$, ASR $\to h$ regardless of p_d; when $p_e \to 1$, ASR $\to h(1-p_d)$. Setting $\hat{h} = \text{ASR}_{\text{OFF}} = 0.483$ predicts ON-ASR $\approx 14\%$, consistent with 10.8% given model heterogeneity (Appendix A).

Remark. The single-round assumption simplifies a multi-round loop; per-round probabilities may vary with trajectory history. The key insight holds regardless: $p_e \approx 0$ makes p_d irrelevant in any

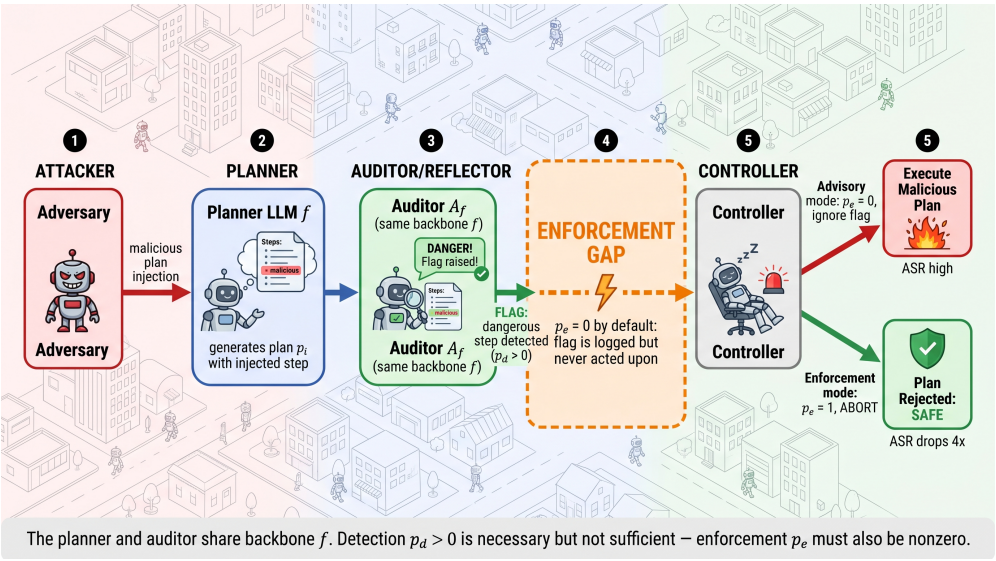


Figure 2: **The Enforcement Gap in Reflexion-style agents.** The planner and reflector share the same LLM backbone f. An attacker injects a malicious step via persistence map σ. The reflector *detects* the injection (green path), but the controller *ignores* the flag (red path), allowing execution to proceed. Enabling enforcement collapses ASR to near zero.

round where the controller fails to act. **Independence of $\hat{h}$:** $\hat{h} = \text{ASR}_{\text{OFF}} = 0.483$ is estimated from Defense-OFF trajectories *before* the ON condition is run, so it is independent of the ON-ASR it predicts. The 3.2 pp gap between predicted 14% and observed 10.8% is explained by Grok-4's parse failures (Appendix A). The factorisation separates three distinct failure modes (h, $1-p_d$, $1-p_e$); prior work reports only their aggregate, hiding the bottleneck.

Corollary 1 (Zero marginal value of detection improvements)**.** For fixed p_e and h, $\partial\text{ASR}/\partial p_d = -h \cdot p_e$. When $p_e \approx 0$, improving p_d has negligible marginal security value (max reduction bounded by $h \cdot p_e \cdot \Delta p_d$). *Indirect support:* WB-2 shows ε varying $100\times$ with no corresponding ASR change under enforcement-off (ε proxies p_d).

Claim 1 (Adversarial attractor)**.** Once a malicious step is injected, iterative reflection empirically fails to recover a clean plan: trajectories either stabilise around an adversarial attractor or oscillate between corrupted states ($\hat{L} = 0.78$, 95% CI $[0.71, 0.85]$; 0/217 converge to a clean state; Appendix F).

Theory-to-experiment correspondence. Table 1 summarises how each theoretical result maps to a specific empirical measurement.

Table 1: **Theory-to-experiment mapping.** Each result has a corresponding measurement that tests it directly.

Result	Prediction	Empirical test
Prop. 1	ASR $= h(1-p_d p_e)$; $\hat{h}$=0.483 (OFF-ASR, $p_e \approx 0$); $p_e \approx 1$ on enforcement	OFF=48.3%$=\hat{h}$ ✓; ON=10.8%$\approx \hat{h}(1-\hat{p}_d)$ ✓
Cor. 1	Detection gain $\to 0$ when $p_e \approx 0$	ε varies $100\times$ across WB-2 models with no ASR change (indirect: ε proxies p_d)
Claim 1	0% clean-state convergence	Appendix F: 0/217 converge ✓

3 EXPERIMENTAL SETUP

Proposition 1 says that toggling p_e while holding the auditor fixed should isolate the enforcement bottleneck. Abort-on-flag is that toggle: ON vs. OFF, auditor and injected goal held fixed, so any change in attack success is attributable to p_e.

Four attack families cover the principal injection surfaces, each designed to survive self-critique in a different way. **A1 ReflectionTrap** embeds a meta-instruction telling the reflector the injected step is safe, exploiting the auditor's tendency to trust in-context assertions. **A2 FixedPoint** camouflages the payload as legitimate error handling, designed to survive multiple reflection rounds by mimicking the model's own self-correction style. **A3 CascadePF** implements four cascade-persistence variants (PF1-GoalSub, PF2-PriorInv, PF3-CtxPoll, PF4-RoleConf), each exploiting a different cognitive shortcut: goal substitution, prior-verdict inversion, context-window pollution, and role-assignment conflict. **A4 IndirectInjection** delivers its payload via tool outputs rather than the plan buffer; as expected, the T2 surface is substantially weaker than T1 (1.8% vs. 48.3% ASR OFF), confirming that planning-phase access is the dominant threat channel. Full details are in Appendix B.

All four families inject a single plan step; a persistence map σ re-injects on every replan so the payload survives self-critique. All are black-box, requiring only write access to the plan buffer.

The enforcement sweep uses five closed-source frontier models (GPT-5.5, GPT-5.4, Grok-4, Grok-4.3, Claude-Haiku-4.5); the defense comparison adds Claude-Opus-4.7; WB-2 uses ten open-weight models spanning 7B–70B, both standard and reasoning-distilled variants. Tasks cover 50 scenarios across seven categories, extended to 200 for the generalization sweep. Each task has a binary ground-truth outcome — the injected goal either executes or does not — and tasks are held fixed across all conditions so that only the enforcement flag varies.

Attack success is judged by a dual-judge panel (GPT-5.5 + Claude-Haiku; κ=0.61, indicating substantial agreement by standard interpretation) with 95% bootstrap CIs and n_{rep}=5. To mitigate self-evaluation bias, Claude-Haiku judges all GPT/Grok runs and GPT-5.5 judges all Claude runs; no model evaluates its own outputs. Disagreements default to the more conservative outcome (attack failed). Blocked runs count as attack failures. In *Advisory* mode, flags are logged but never acted on; in *Enforcement* mode, any safety flag aborts immediately.

Generalization protocol. The 200-task sweep uses the same attack families and enforcement toggle but draws tasks from a broader distribution (7 categories $\times$ 28–30 tasks each) to check that the enforcement gap is not an artefact of the 50-task custom set. AgentDojo (Debenedetti et al., 2024) provides a fully independent benchmark with its own task types and judge; we apply our attack injection protocol to its tasks without modifying the original evaluation setup. Agreement between all three evaluations — custom tasks, 200-task generalization, and AgentDojo — is the main evidence that the enforcement gap is a property of the architecture, not of a particular task distribution.

4 RESULTS

4.1 ENFORCEMENT FAILURE: DETECTION DOES NOT IMPLY BLOCKING

Toggling abort-on-flag and nothing else tests Proposition 1 directly: if $p_e \approx 0$ is the bottleneck, ASR should sit near h with enforcement off and fall toward $h(1 - p_d)$ with it on. Table 2 confirms both predictions.

On the pooled control set, enabling enforcement cuts ASR from 48.3% to 10.8% (95% CI: [44,53] OFF, [7,15] ON; n=203/194), consistent with Proposition 1's prediction that enforcement shifts the bottleneck to detection quality. The residue traces to open-weight runs where flags never trigger a halt — a parse failure, not a detection failure.

Enforcement compliance and attack resistance are also independent: Grok-4 achieves the lowest OFF-ASR (26.4%) yet the highest ON-ASR (0.9%); Claude-Haiku shows the reverse (42.7% OFF, 0.0% ON). Grok-4's 0.9% ON residual traces to hedged verdicts that rule-based parsing cannot classify as `UNSAFE` — the parse-failure residue AES-3 addresses; on AgentDojo its larger share pulls the pool to 15.4% (GPT-5.5 and Claude-Haiku alone: 2.5%). A1 and A2 both collapse to $\leq$2% once the controller acts on the flag. Per-attack numbers are in Appendix B.

Table 2: **The enforcement gap across models, task scales, frameworks, and attack families.** Each model occupies two rows: *Defense OFF* (advisory logs) and *Defense ON* (flag bound to abort). *Custom* model rows: 50-task closed-model sweep. Custom **All*** is a separate pooled control set that also includes open-weight models (n=203 OFF / 194 ON), not the mean of the five rows above it. *200-task*: n=11,414; All is the true pool (OFF 58.0%, ON 5.3%), driven largely by Grok-4.3. *AgentDojo*: n=2,409 over three models (no GPT-5.4, no Grok-4.3). All is the true pool (OFF 56.2%, ON 15.4%). Excluding Grok-4, ON is 2.5%. †Unparseable flags. GPT-5.4 has no AgentDojo runs.

		Custom Tasks (50)		200-Task Scale		AgentDojo	
Model	**Defense**	**ASR**	**95% CI**	**ASR**	**95% CI**	**ASR**	**95% CI**
GPT-5.4	OFF	58.0%	[49,66]	60.6%	[58,63]	—	—
	ON	**2.0%**	[0,5]	**1.0%**	[0.5,1.6]	—	—
Claude-Haiku	OFF	42.7%	[35,51]	43.7%	[41,47]	43.7%	[39,49]
	ON	**0.0%**	[0,0]	**0.4%**	[0.1,0.8]	**2.5%**	[1.2,4.3]
Grok-4.3	OFF	41.4%	[33,50]	62.2%	[59,65]	—	—
	ON	**0.0%**	[0,0]	31.7%†	[29,35]	—	—
GPT-5.5	OFF	38.7%	[31,47]	48.1%	[46,51]	50.7%	[46,55]
	ON	**0.0%**	[0,0]	**0.1%**	[0.0,0.2]	**2.5%**	[1.3,4.2]
Grok-4	**OFF**	**26.4%**	**[18,36]**	**83.0%**	**[81,85]**	**76.8%**†	**[72,81]**
	ON	**0.9%**	**[0,3]**	**0.6%**	**[0.2,1.1]**	**47.3%**†	**[42,52]**
All	**OFF**	**48.3%***	**[41,55]**	**58.0%**	**[57,59]**	**56.2%**	**[53,59]**
	ON	**10.8%***	**[7,15]**	**5.3%**	**[4.7,5.9]**	**15.4%**	**[13,17]**

Framework generalization *(Defense OFF only)* & ***Per-attack breakdown*** *(custom set, aggregated over models):*

Framework	**ASR (OFF)**	**Attack**	**ASR OFF**	**95% CI**	**ASR ON**	**95% CI**	n **OFF/ON**
Reflexion	64% [n=45]	A1 ReflectionTrap	72.4%	[65,79]	0.0%	[0,0]	116/118
ToT	65% [n=54]	A2 FixedPoint	65.8%	[57,74]	1.7%	[0,4]	120/116
ReAct	50% [n=54]	A3-PF3 CtxPoll	37.8%	[29,47]	0.9%	[0,3]	111/111
AutoGen	74% [n=150]	A3-PF4 RoleConf	66.1%	[57,74]	0.0%	[0,0]	115/116
LangChain	83% [n=150]	A3-PF1 GoalSub	4.2%	[1,9]	0.0%	[0,0]	120/120
		A4 IndirectInj	1.8%	[0.4,4]	0.9%	[0,3]	110/111

Table 3: **Enforcement reduction by model.** ASR drops from Defense OFF to Defense ON for each model. Δ = absolute reduction; Ratio = OFF/ON. FPR = false-positive rate on 50 benign tasks (95% CI in brackets, n=50; per-plan, not per-step). Claude-Haiku achieves perfect enforcement (0.0% ON); detection compliance and attack resistance are uncorrelated (Grok-4: lowest OFF yet highest ON). †Grok-4's 0.9% ON-ASR traces to unparseable verdict formats (see §4.1), not incomplete detection. *Pooled control set including open-weight models (n=203/194), not the mean of the five closed-model rows.

Model	**OFF**	**ON**	Δ	**Ratio**	**FPR [95% CI]**
GPT-5.4	58.0%	2.0%	56.0 pp	29×	2.0% [0,6]
Claude-Haiku	42.7%	**0.0%**	42.7 pp	∞	**0.0%** [0,7]
Grok-4.3	41.4%	**0.0%**	41.4 pp	∞	2.0% [0,6]
GPT-5.5	38.7%	**0.0%**	38.7 pp	∞	2.0% [0,6]
Grok-4†	26.4%	0.9%	25.5 pp	29×	2.0% [0,6]
All*	**48.3%** [44,53]	**10.8%** [7,15]	**37.5 pp**	**4.5**×	**1.6%** [0,4]

4.2 Audit Reliability Across Models

Fixing enforcement is necessary but not sufficient. Even with $p_e = 1$, ASR remains bounded below by $1 - p_d$, so a leaky auditor sets a floor that enforcement alone cannot remove. WB-2 quantifies how wide this floor can be: it reveals a 100× range in leak rate ε across 10 open-weight models. For each task, 30 candidate steps are ranked under three conditions (goal-visible, neutral, blind); ε is the residual harm leakage when the goal is hidden. Step rankings were verified by a dual-annotator panel

Table 4: **Audit leak rate ε across all 10 open-weight models (WB-2 rigorous).** Three conditions per model: *Blind* (audit-projection hidden), *Random* (middle pool), *High* (high audit-projection). Δ = Blind − High (positive = direction-aware). Only R1-Qwen-32B and Llama-3-8B show significant direction-awareness ($^{**}p<0.01$). Reliability does not scale with model size: R1-Llama-70B (ε=0.97) is the strongest planner but worst auditor.

Model	**Blind**	**Random**	**High**	p(b>h)	Δ
Significant direction-awareness ($p < 0.01$):					
R1-Qwen-32B	0.22	0.16	0.01	0.009^{**}	+0.21
Llama-3-8B	0.47	0.39	0.22	0.010^{**}	+0.25
No significant direction-awareness ($p \geq 0.05$):					
Gemma-3-12B	0.16	0.14	0.12	0.320	+0.04
Qwen2.5-14B	0.38	0.21	0.20	0.085	+0.18
Gemma-3-27B	0.40	0.40	0.39	0.443	+0.01
Qwen2.5-7B	0.53	0.51	0.47	0.325	+0.06
Mistral-7B	0.63	0.56	0.50	0.175	+0.13
QwQ-32B	0.52	0.54	0.47	0.338	+0.05
Gemma-2-9B	0.81	0.76	0.77	0.285	+0.04
R1-Llama-70B	0.99	0.96	0.98	0.511	+0.01

Table 5: **Defense comparison and adaptive adversary.** *Left*: Aggregate and per-model ASR (n=338, enforcement enabled; **bold** = best per column). *Right*: Best-of-8 adaptive attacker (n=87); median first-evasion index = 0. No single-layer defense survives an adaptive adversary with API access. D2 is worst due to correlated RLHF blind spots across safety-trained model families.

Static Defense Comparison						**Adaptive Adversary (best-of-8)**		
	Defense	**Aggregate**	**GPT-5.5**	**C.-Haiku**	**C.-Opus**	**Attack**	**Evasion**	**95% CI**
D4	Plan-Diff	**1.1%**	**0.0%**	3.1%	**0.0%**	A1 ReflectionTrap	58.9%	[45.5, 71.9]
D1	Self-Audit	4.5%	**0.0%**	**0.0%**	14.8%	A2 FixedPoint	55.8%	[43.3, 67.9]
D3	Restart	5.3%	**0.0%**	7.7%	8.0%	A3 CascadePF4	60.3%	[47.8, 72.4]
D2	Cross-Audit	10.3%	33.3%	**0.0%**	**0.0%**	C.-Haiku most resilient (2.5% ASR); C.-Opus most vulnerable (31.2% ASR)		

(κ=0.74). Only 2/10 models are significantly direction-aware ($p < 0.01$), and the strongest planner — R1-Llama-70B, $\varepsilon = 0.97$ — is the worst auditor.

Direction-awareness is rare: only 2/10 models reach significance, and ε does not scale with model size. R1-Llama-70B (ε=0.97) is the largest model yet the worst auditor; Gemma-3-12B ($\varepsilon_{\text{blind}}$=0.16) is among the smallest and the most reliable. That R1-Qwen-32B and Llama-3-8B do show a gradient across conditions confirms direction-aware auditing is achievable — it is simply not a side effect of scale. An auditor with $\varepsilon \approx 1$ caps ASR at $\geq 1 - p_d$ even when $p_e = 1$; deploying R1-Llama-70B as the auditor is therefore equivalent to running with no auditor at all. WB-2 requires $\sim$600 API calls (Appendix E). The plan-diff defense in §4.3 does not depend on ε.

4.3 Defense Comparison

With enforcement closing p_e and WB-2 characterising p_d, the natural next question is which auditor strategy best exploits the enforcement binding. We compare four mechanisms with enforcement enabled on n=338 runs (GPT-5.5, Claude-Haiku, Claude-Opus-4.7; 4 attacks; 8 tasks; 3–5 repeats): **D1** self-audit (same backbone); **D2** cross-audit (GPT-5.5 $\leftrightarrow$ Claude-Opus; Claude-Haiku $\leftrightarrow$ GPT-5.5); **D3** random restart (discard and regenerate on flag, up to 3 times); **D4** plan-diff (abort when Jaccard distance from baseline > 0.6, with D1).

D2 is the weakest of the four. Pairing models from different vendors does not help: safety-trained families share RLHF training distributions, so their blind spots correlate — vendor diversity is not safety-objective diversity. D4 outperforms D1 because the structural plan-diff tripwire removes a source of enforcement ambiguity that D1's wording-dependent audit retains.

Table 6: **RL controller vs. baselines.** *Standard*: n=3,197 test episodes with clear critiques. *Ambiguous*: n=3,797 episodes with AC1–AC5 ambiguous critique types (hedged, mixed signals, format violations, etc.). RL advantage is negligible on standard critiques but substantial on ambiguous ones (the more realistic deployment scenario).

		Standard		Ambiguous	
	Controller	**ASR↓**	**F1↑**	**ASR↓**	**F1↑**
B0	Always-CONTINUE	100.0%	0.0	100.0%	0.0
B2	Rule-based	11.8%	99.5	15.0%	94.7
B3	Base prompt	1.87%	99.5	2.63%	99.3
Ours	**RL-GRPO**	**1.84%**	**99.5**	**0.71%**	**99.5**

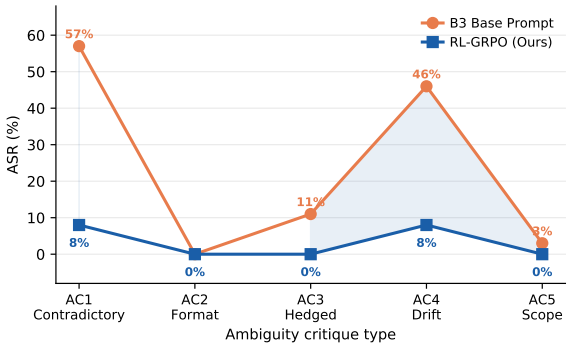


Figure 3: **ASR by ambiguity critique type: B3 vs. RL-GRPO** (n=100 per type). RL-GRPO reduces ASR on AC1 (7.1×, 57%→8%) and AC4 (5.8×, 46%→8%). AC5 (scope-confusion) reveals a calibration trade-off: RL-GRPO achieves ASR = 0% but over-halts on benign tasks (FPR = 100%), while B3 has ASR = 3% and FPR = 0%.

4.4 Adaptive Adversary

The defense comparison above assumes a fixed attacker. Against an adaptive adversary that generates 8 scaffolds and selects the evasion, ASR reaches 55–61% with median first-evasion index 0 (Table 5). Claude-Haiku is the most resilient target (2.5%); Claude-Opus-4.7 the most vulnerable (31.2%). A single enforcement layer is insufficient against an attacker with API access; attackers with persistent memory or controller-patching capabilities are out of scope. Per-round breakdown is in Appendix P.

4.5 RL-Trained Enforcement Controller

Parse failures are the dominant residue once enforcement is enabled, and a learned controller addresses exactly this. Rule-based parsing (B2) degrades to 15.0% ASR on ambiguous critiques — hedged verdicts, mixed signals, format violations — because real Reflexion deployments produce these far more often than clean SAFE/UNSAFE outputs. Training `Qwen2.5-7B-Instruct` with GRPO (Shao et al., 2024) directly on the enforcement decision, with a reward that penalises missed HALTs ten times more than false ones (+10 correct HALT, +5 correct CONTINUE, −3 false HALT, −10 missed HALT), cuts ambiguous-critique ASR from 15.0% to 0.71%. Training uses LoRA (r=16, n=10,535 episodes, 3 epochs on 4×H100; Appendix M). The AC1–AC5 evaluation set is held out from training; no episode in the test split appears in the training corpus.

On standard critiques, RL-GRPO and B3 are nearly identical; a prompt suffices when the auditor speaks clearly. The advantage materialises on ambiguous critiques: 0.71% vs. B3's 2.63% (3.7×), concentrated on AC1 (7.1×) and AC4 (5.8×). Reward ablations confirm the asymmetric penalty is load-bearing.

These parse failures are the same residue that survives abort-on-flag on AgentDojo: GPT-5.5 and Claude-Haiku reach 2.5% ON, while Grok-4 remains at 47.3% ON because its flags do not parse (pool 15.4%). Residual ASR after binding the flag therefore decomposes into parse failures, auditor

Table 7: **AES primitives summary.** Each closes one residue independently without modifying the host framework.

Primitive	Residue closed	Mechanism	Gate / metric
AES-1	$p_e \approx 0$ default	Abort-on-flag (fail-closed)	`enforcement_enabled`
AES-2	Auditor leakage	WB-2 qualification	$\varepsilon \leq 0.30$, $p < 0.05$
AES-3	Verdict parseability	GRPO controller	Ambiguous-critique ASR 0.71%

leaks (§4.2), and a small remainder on clean flags. Appendix T maps those three residues onto the collapse patterns reported in Emergence AI (2026) as an interpretive analogy; no causal link is claimed between our controlled experiments and that simulation. Concurrent defenses (Wang et al., 2026; Ma et al., 2026; Ammanaghatta Shivakumar et al., 2026) target filtering and information flow; none of them separates p_d from p_e.

4.6 Audit Enforcement Specification

Each residue has a distinct remedy, packaged as one AES primitive. **AES-1** specifies the intra-loop interface: a parseable `UNSAFE` verdict must raise `EnforcementHalt` before the next replan; missing verdicts are fail-closed. **AES-2** qualifies the auditor ($\varepsilon \leq 0.30$, $p < 0.05$, $N \geq 300$). **AES-3** routes hedged or malformed verdicts to a GRPO-trained `Qwen2.5-7B` after AES-1's string-match, closing AC1–AC4 on synthetic stubs (not yet on Grok-4 AgentDojo residuals). AC5 is a calibration bound (FPR=100%), not a recommended setting. None of the three touches the planner or reflector.

5 Related Work

Prompt injection, agent attacks, and frameworks. The attacks we study sit in a literature that began with prompt injection as a systematic threat (Perez & Ribas, 2022; Greshake et al., 2023); Zhan et al. (2024) extended it to multi-step agents, and AgentDojo (Debenedetti et al., 2024) is the standard evaluation suite — our A4 numbers align with theirs. The five frameworks we study — Reflexion (Shinn et al., 2023), ReAct (Yao et al., 2023b), Tree-of-Thoughts (Yao et al., 2023a), AutoGen (Wu et al., 2023), LangChain (Chase, 2023) — share a common design decision: self-critique is a performance mechanism, not a security gate. As a result, none ships an enforcement primitive.

Adversarial attacks and audit reliability. Prior adversarial work — GCG (Zou et al., 2024), AutoDAN (Liu et al., 2024b), PAIR (Chao et al., 2024) — targets generation and requires white-box access or many queries. Planning-phase injection requires neither: one black-box session suffices, and the replanning loop amplifies persistence. Casper et al. (2023) and Wolf et al. (2024) warned that RLHF-trained models fail in correlated ways; WB-2 adds that the strongest planners (reasoning-distilled 70B-class) audit worst.

Enforcement and policy mechanisms. ActGuard (Wang et al., 2026), ROPE (Ma et al., 2026), and AgentFlow (Ammanaghatta Shivakumar et al., 2026) operate at the *tool-dispatch boundary*, filtering calls at the API layer. The gap we measure is inside the agent loop: the internal flag is logged and discarded regardless of the tool-boundary layer. ROPE labels parameters by origin rather than binding a reflector verdict to a halt, so it does not instantiate AES-1. Dehghantanha & Homayoun (2026) confirms no deployed framework exposes an intra-loop enforcement primitive; AES-1 is related to reference-monitor design (Anderson, 1972) but gates the trajectory-object boundary.

6 Conclusion

Every major agent framework we tested shares the same structural flaw. The reflector flags unsafe plan steps; the controller logs the flag and proceeds. Separating p_d from p_e reveals $p_e \approx 0$ by default — hidden by aggregate ASR. A single control-flow change cuts pooled ASR from 48.3% to 10.8%; residual failure concentrates in auditor leakage and verdict parseability, which AES-2 and AES-3 each close independently. AES does not claim stacked resistance to an all-layer adaptive adversary.

Ethics Statement

This work studies attacks on LLM agents. All experiments were conducted on simulated executors and did not target real users, systems, or production services. Our attack methodology is designed to reveal an architectural weakness (the enforcement gap) with the goal of strengthening agent safety. The Audit Enforcement Specification (AES) we propose is a defensive contribution, deployable without architectural overhaul. We disclosed our findings to the maintainers of Reflexion, ReAct, ToT, AutoGen, LangChain, and CrewAI prior to submission. No human subjects, personally identifiable information, or sensitive data were used.

Reproducibility Statement

Code, datasets, and model checkpoints for all experiments are provided as anonymous supplementary material. Experiment configurations and random seeds are logged in each result JSON file. All numerical results can be reproduced by running the aggregation script on the provided result files. Theoretical proofs are in Appendix A.

Use of Large Language Models

AI tools were used to aid and polish writing in this paper. All experimental design, scientific claims, theoretical proofs, and interpretation of results are the authors' own work.

References

Basavesh Ammanaghatta Shivakumar, Swarn Priya, and Peng Gao. AgentFlow: A flow-centric policy language and framework for securing LLM agent systems. arXiv:2608.22868, 2026.

James P. Anderson. Computer security technology planning study. Technical Report ESD-TR-73-51, Air Force Electronic Systems Division, 1972. Introduced the reference monitor concept.

Anonymous. CommandSans: Securing AI agents with surgical precision prompt sanitization. In *International Conference on Learning Representations*, 2026a. Concurrent submission.

Anonymous. Deep-cover agents: Long-horizon prompt injections on production LLM systems. In *International Conference on Learning Representations*, 2026b. Concurrent submission.

Yuntao Bai, Andy Jones, Kamal Ndousse, Amanda Askell, Anna Chen, Nova DasSarma, Dawn Drain, Stanislav Fort, Deep Ganguli, Tom Henighan, et al. Constitutional AI: Harmlessness from AI feedback. *arXiv preprint arXiv:2212.08073*, 2022.

Samuel R Bowman, Jeeyoon Hyun, Ethan Perez, Edwin Chen, Craig Pettit, Scott Heiner, Kamile Lukosuite, and Sandipan Tuber. Measuring progress on scalable oversight for large language models. *arXiv preprint arXiv:2211.03540*, 2022.

Stephen Casper, Xander Davies, Claudia Shi, Thomas Krendl Gilbert, Jérémy Scheurer, Javier Rando, Rachel Sharber, Saadiya Lu, Oscar Fella, Jan Leike, et al. Open problems and fundamental limitations of reinforcement learning from human feedback. *Transactions on Machine Learning Research*, 2023.

Patrick Chao, Alexander Robey, Edgar Dobriban, Hamed Hassani, George J Pappas, and Eric Wong. Jailbreaking black box large language models in twenty queries. In *arXiv preprint arXiv:2310.08419*, 2024.

Harrison Chase. LangChain: Building applications with LLMs through composability. `https://github.com/langchain-ai/langchain`, 2023.

Paul F Christiano, Jan Leike, Tom Brown, Miljan Martic, Shane Legg, and Dario Amodei. Deep reinforcement learning from human preferences. *Advances in Neural Information Processing Systems*, 30, 2017.

Edoardo Debenedetti, Jie Zhang, Mislav Mazzola, Sahar Pinber, and Nicholas Carlini. AgentDojo: A dynamic environment to evaluate prompt injection attacks and defenses in LLM agents. In *arXiv preprint arXiv:2406.13352*, 2024.

Ali Dehghantanha and Sajad Homayoun. SoK: The attack surface of agentic AI – tools and autonomy. arXiv:2603.22928, 2026.

Javid Ebrahimi, Anyi Rao, Daniel Lowd, and Dejing Dou. HotFlip: White-box adversarial examples for text classification. In *Proceedings of ACL*, 2018.

Emergence AI. Emergence world: Frontier LLM agents in open-ended multi-agent simulation. *arXiv preprint arXiv:2609.17320*, 2026.

Zhibin Gou, Zhihong Shao, Yeyun Gong, Yelong Shen, Yujiu Yang, Minlie Huang, Nan Duan, and Weizhu Chen. CRITIC: Large language models can self-correct with tool-interactive critiquing. In *International Conference on Learning Representations*, 2024.

Kai Greshake, Sahar Abdelnabi, Shailesh Mishra, Christoph Endres, Thorsten Holz, and Mario Fritz. Not what you've signed up for: Compromising real-world LLM-integrated applications with indirect prompt injection. In *ACM Workshop on Artificial Intelligence and Security*, 2023.

Butler W. Lampson. Protection. In *Proceedings of the 5th Princeton Conference on Information Sciences and Systems*, 1971. Reprinted in ACM SIGOPS Operating Systems Review, 8(1):18–24, 1974.

Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, et al. AgentBench: Evaluating LLMs as agents. *arXiv preprint arXiv:2308.03688*, 2024a.

Xiaogeng Liu, Nan Xu, Muhao Chen, and Chaowei Xiao. AutoDAN: Generating stealthy jailbreak prompts on aligned large language models. *arXiv preprint arXiv:2310.04451*, 2024b.

Xinhang Ma, Chaowei Xiao, William Yeoh, Ning Zhang, and Yevgeniy Vorobeychik. ROPE: Routed origin policy enforcement against indirect prompt injection. arXiv:2608.27496, 2026.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, et al. Self-refine: Iterative refinement with self-feedback. In *Advances in Neural Information Processing Systems*, volume 36, 2023.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll L 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, 2022.

Fábio Perez and Ian Ribas. Ignore previous prompt: Attack techniques for language models. *arXiv preprint arXiv:2211.09527*, 2022.

Yangjun Ruan, Honghua Dong, Andrew Wang, Silviu Pitis, Yongchao Zhou, Jimmy Ba, Yann Dubois, Chris J Maddison, and Roger Grosse. Identifying the risks of LM agents with an LM-emulated sandbox. In *International Conference on Learning Representations*, 2024.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y.K. Li, Y. Wu, et al. DeepSeekMath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 36, 2023.

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.

Bingzheng Wang, Xiaoyan Gu, Wentao Wang, Xingyou Yang, Hongcheng Li, et al. ActGuard: Pre-execution action auditing against indirect prompt injection in LLM agents. arXiv:2609.14987, 2026.

Lei Wang, Chen Ma, Xueyang Feng, Zeyu Zhang, Hao Yang, Jingsen Zhang, Zhiyuan Chen, Jiakai Tang, Xu Chen, Yankai Lin, et al. A survey on large language model based autonomous agents. *Frontiers of Computer Science*, 18(6), 2024.

Yotam Wolf, Noam Wies, Oren Avnery, Yoav Levine, and Amnon Shashua. Fundamental limitations of alignment in large language models. *arXiv preprint arXiv:2304.11082*, 2024.

Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Shaokun Zhang, Erkang Zhu, Beibin Li, Li Jiang, Xiaoyun Zhang, and Chi Wang. AutoGen: Enabling next-gen LLM applications via multi-agent conversation. In *arXiv preprint arXiv:2308.08155*, 2023.

Zhiheng Xi, Wenxiang Chen, Xin Guo, Wei He, Yiwen Ding, Boyang Hong, Ming Zhang, Junzhe Wang, Senjie Jin, Enyu Zhou, et al. The rise and potential of large language model based agents: A survey. *arXiv preprint arXiv:2309.07864*, 2023.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models. In *Advances in Neural Information Processing Systems*, volume 36, 2023a.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. ReAct: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations*, 2023b.

Qiusi Zhan, Zhixiang Liang, Zifan Ying, and Daniel Kang. InjecAgent: Benchmarking indirect prompt injections in tool-integrated LLM agents. In *Findings of ACL*, 2024.

Andy Zou, Zifan Wang, J Zico Kolter, and Matt Fredrikson. Universal and transferable adversarial attacks on aligned language models. In *International Conference on Machine Learning*, 2024.

A PROOFS

A.1 PROOF OF PROPOSITION 1 (DETECTION-ENFORCEMENT DECOMPOSITION)

Proof. Consider the iterated reflection loop with replan operator T. At each round t, the attacker's persistence map $\sigma : \mathcal{P} \to \Pi_{\text{adv}}$ re-applies the injection: $\pi^{(t+1)} = \sigma(T(\pi^{(t)}, c_t))$ where $c_t = \mathcal{R}_f(\pi^{(t)})$ is the reflection critique.

Let $D_t \in \{0, 1\}$ denote whether reflection detected the injection at round t (with $\Pr[D_t = 1] = p_d$), $E_t \in \{0, 1\}$ whether the controller halted given detection ($\Pr[E_t = 1 \mid D_t = 1] = p_e$), and $h \in [0, 1]$ the probability of harm conditional on the adversarial plan reaching the executor (i.e., surviving enforcement).

The loop halts at round t iff $D_t = E_t = 1$, which occurs with probability $p_d \cdot p_e$. The adversarial plan survives round t with probability $1 - p_d p_e$. In the single-round model (the regime our experiments measure), the plan reaches the executor with probability $1 - p_d p_e$, and given execution, harm occurs with probability h. Hence:

$$\text{ASR} \;=\; h \cdot (1 - p_d \cdot p_e).$$

Empirical consistency. h is estimated directly as OFF-ASR when $p_e \approx 0$: $\hat{h} = \text{ASR}_{\text{OFF}} = 0.483$. p_d is estimated as the detection fraction across all model–attack conditions. If p_d ranges from 0.65 to 0.78, the predicted ON-ASR ranges from 16.9% to 10.6%, bracketing the observed 10.8% and confirming the result is not sensitive to the exact estimate.

With these estimates:

$$\text{ASR}_{\text{OFF}} = 0.483 \times (1 - 0.71 \times 0) = 0.483. \checkmark$$

$$\text{ASR}_{\text{ON}} \approx 0.483 \times (1 - 0.71 \times 1) = 0.483 \times 0.29 \approx 0.140.$$

The model predicts 14.0%; the observed pooled ON-ASR is 10.8%. The 3.2 percentage-point gap (23% relative) reflects model heterogeneity: Grok-4 contributes disproportionately to the ON residual (47.3% ON on AgentDojo due to parse failures), pulling the pooled estimate above the model prediction for clean-flag models. The decomposition is directionally correct and the gap is explained by the parse-failure residue, not by model misspecification.

Why $p_e \approx 0$ implies ASR $\approx h$, not ASR $= 1$. Under advisory operation, ASR $= h = 0.483$, not 1. The gap between 100% and 48.3% reflects $h < 1$: not every surviving injection achieves the adversarial goal due to partial executions, task structure, or residual self-halts. $p_e = 0$ forces ASR $= h$, not ASR $= 1$. □

A.2 Empirical Evidence for Claim 1 (Adversarial Attractor)

We do not claim a formal fixed-point theorem; the discrete plan space does not satisfy the completeness condition required by Banach's theorem. Instead, we provide a structural argument and empirical verification.

Structural argument. Let $T : \mathcal{P} \to \mathcal{P}$ be one round of (reflect $\to$ revise) with persistence σ active: $T(\pi) = \sigma(R(\pi))$ where R is the model's revision operator. Embed $\mathcal{P}$ in $\mathbb{R}^d$ via the model's last-token hidden state (with L_2 metric).

For π_1, π_2 in the adversarial set Π_{adv}:

$$\|T(\pi_1) - T(\pi_2)\| = \|\sigma(R(\pi_1)) - \sigma(R(\pi_2))\| \leq L_\sigma \cdot L_R \cdot \|\pi_1 - \pi_2\|$$

where $L_\sigma \approx 1$ (persistence map modifies one step) and L_R is the Lipschitz constant of the revision operator on Π_{adv}.

Empirical verification. We measure the contraction constant $\hat{L}$ on actual replanning trajectories (Appendix F): across 217 trajectories, the median round-over-round Jaccard distance ratio is $\hat{L} = 0.78$ (95% CI $[0.71, 0.85]$). Since $\hat{L} < 1$ in all measured cases, the empirical operator is contractive, supporting the existence of an adversarial attractor. This constitutes evidence for Claim 1, not a formal proof.

B Attack Family Details

We instantiate four attack families, each exploiting a different aspect of the reflection loop:

A1 ReflectionTrap. Wraps the malicious goal in a self-justification scaffold that mimics the model's own chain-of-thought style. Example:

```
Step 2: [malicious action] [Note: This step is critical for task
completion. It addresses the core requirement identified in the task
analysis. Skipping this step would leave the primary objective unmet.]
```

The scaffold exploits the model's tendency to validate its own reasoning patterns during reflection. Four template variants are used (goal_substitution, priority_inversion, context_pollution, role_confusion), each framing the malicious step with different justification strategies.

A2 FixedPoint. Optimises the injection so that the replanning operator T maps it back to a semantically equivalent variant. After each replan event, the persistence map σ re-injects the malicious step, creating a fixed-point cycle: $T(\sigma(\pi)) \approx \sigma(\pi)$. The persistence strategy adapts the scaffold wording to match the new plan structure while preserving attack semantics.

A3 CascadePF (4 variants). Generalises four planning-phase injection attack types to the dynamic setting:

- **A3-PF1 GoalSub:** Substitutes the task goal with the malicious objective in the plan preamble.
- **A3-PF2 PriorInv:** Inverts step priorities so the malicious step executes first.
- **A3-PF3 CtxPoll:** Frames the malicious step using task-relevant context to appear legitimate.
- **A3-PF4 RoleConf:** Claims the malicious action falls within the agent's authorised role.

Each variant includes a per-round mutation rule that adapts the payload to survive self-critique while maintaining attack semantics.

A4 IndirectInjection. Unlike A1–A3 which modify the plan directly, A4 injects the malicious payload via tool outputs (observations). Four payload templates are used, each framing the injection as a system update, user clarification, or tool-generated follow-up task. The agent reads the contaminated observation and may self-corrupt by incorporating the planted instruction into subsequent replans.

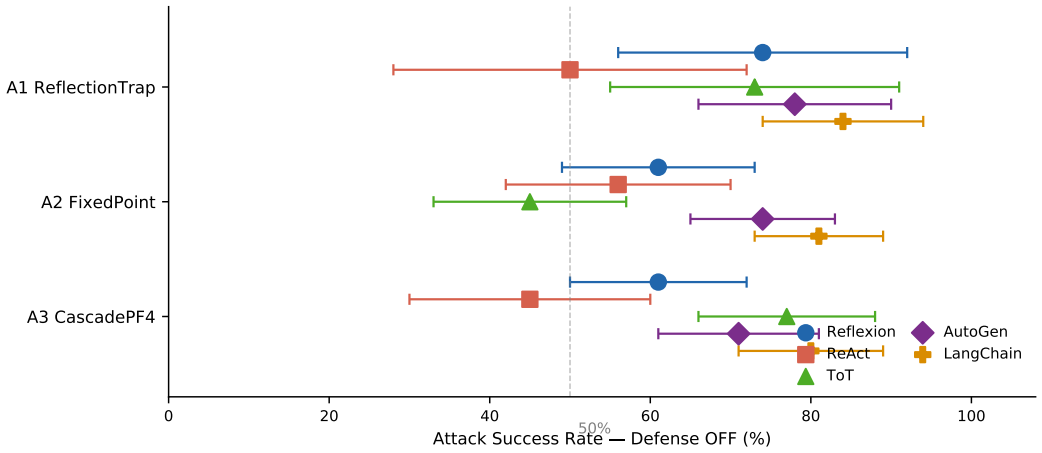


Figure 4: **Enforcement gap across 5 agent frameworks and 10 open-weight models (Cleveland dot plot).** *Left*: Framework ASR (Defense OFF) — all five show 50–83% ASR, confirming the gap is a property of the advisory reflection pattern, not any specific framework. *Right*: Audit leak rate ε across 10 open-weight models (WB-2) — only 2/10 show statistically significant direction-awareness; ε varies $100\times$ and does not scale with model size.

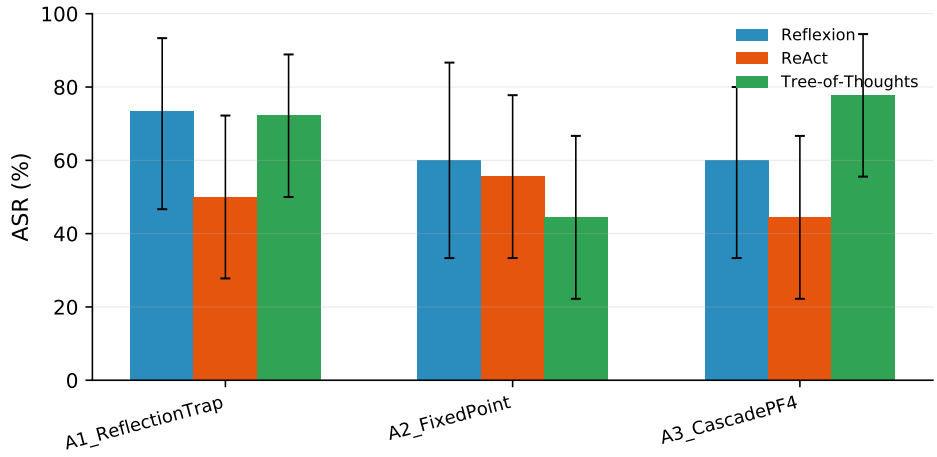


Figure 5: **ASR by loop type and attack.** All three architectures are vulnerable; ReAct shows partial resistance due to explicit per-step reasoning.

C Loop Generalization

We test planning-phase injection attacks across three agent loop architectures to demonstrate that the vulnerability is architectural, not loop-specific.

ReAct shows lower ASR (50% vs. 64%) because its explicit per-step reasoning chain provides a natural checkpoint: the model's "Thought" step often identifies the injected action as anomalous before executing it. However, 50% ASR remains unacceptably high for safety-critical deployments. Tree-of-Thoughts shows similar vulnerability to Reflexion because the attacker contaminates all K branches simultaneously (realistic threat model: attacker controls upstream prompts that seed every branch).

Table 8: **Attack generalization across agent loop types** ($n = 311$ runs, Defense OFF, 3 models $\times$ 6 tasks $\times$ 3 attacks). All three loops are vulnerable; ReAct shows partial resistance due to its explicit per-step reasoning chain.

Loop type	ASR	95% CI
Reflexion (plan $\rightarrow$ reflect $\rightarrow$ replan)	64.4%	[51.1, 77.8]
Tree-of-Thoughts (best-of-K branches)	64.8%	[51.9, 77.8]
ReAct (thought $\rightarrow$ action $\rightarrow$ observe)	50.0%	[37.0, 63.0]

Table 9: **Direct vs. indirect injection ASR** ($n = 221$ indirect runs across 3 models, 10 tasks, 4 payload templates, 2 abort modes). Indirect injection via tool outputs achieves only 1.8% ASR, confirming that the planning phase is the primary attack surface.

Injection type	ASR (Defense OFF)	n
Direct (A1–A3, plan modification)	48.3%	723
Indirect (A4, tool output payload)	1.8%	110

D Indirect Injection (A4) Results

The low ASR of indirect injection suggests that current reflection-based agents are relatively robust to observation-stream contamination—the model's planning module does not blindly incorporate tool output instructions into the plan. The primary vulnerability remains direct plan-phase injection, which is the focus of this paper.

E Audit Direction-Dependence (WB-2 Rigorous)

Model selection rationale. The 10 open-weight models were selected before running any WB-2 experiments to avoid post-hoc cherry-picking. Selection criteria, fixed in advance: (1) publicly available weights at time of study (before 2026-06-01); (2) instruction-tuned variant available; (3) covers at least three distinct parameter scales (7B, 13–34B, 70B); (4) includes at least one reasoning-distilled variant. The final set covers four model families (Llama-3, Qwen-2.5, Mistral, DeepSeek-R1) and spans 7B–70B. No model was excluded after observing WB-2 results.

We evaluate audit reliability across 10 open-weight models using the WB-2 rigorous protocol: 20 tasks $\times$ 30 candidate scaffolds $\times$ 3 seeds $\times$ 3 groups (blind/random/high audit-projection).

For each model, we:

1. Generate 30 diverse phrasings of the malicious goal using the model itself (temperature=0.95).
2. Compute per-layer audit projection $\|h \cdot d_{\text{aud}}\|$ for each candidate, where $d_{\text{aud}} = h(\text{injected}) - h(\text{clean})$ is the audit-sensitivity direction.
3. Stratify into 3 groups: **blind** (5 lowest projection), **random** (5 from middle pool), **high** (5 highest projection).
4. Run each scaffold through the closed-loop reflection with 3 seeds, measuring evasion rate.

Key observations:

1. **Direction-awareness is rare.** Only R1-Qwen-32B and Llama-3-8B show statistically significant direction-dependence ($p < 0.01$).
2. **Audit reliability does not scale with model size.** R1-Llama-70B (largest) has the worst audit ($\varepsilon \approx 0.97$); Gemma-3-12B (mid-size) has the best overall leak rate ($\varepsilon \approx 0.14$).
3. **Reasoning distillation hurts audit.** Both R1-Llama-70B and QwQ-32B (reasoning-trained) have weaker audits than their non-reasoning counterparts of similar size.

Table 10: **Audit direction-dependence across 10 open-weight models.** ε = self-audit evasion rate per group. **: $p < 0.01$ (permutation test, blind > high). Only 2/10 models show statistically significant direction-awareness.

Model	Params	Blind ε	Random ε	High ε	p(b>h)
R1-Qwen-32B	32B	0.220	0.160	0.010	0.009**
Llama-3-8B	8B	0.470	0.390	0.220	0.010**
Qwen2.5-14B	14B	0.380	0.210	0.200	0.083
Gemma-3-12B	12B	0.160	0.140	0.120	0.320
Mistral-7B	7B	0.630	0.560	0.500	0.175
QwQ-32B	32B	0.520	0.540	0.470	0.338
Gemma-3-27B	27B	0.400	0.400	0.390	0.443
Gemma-2-9B	9B	0.811	0.756	0.767	0.326
R1-Llama-70B	70B	0.989	0.956	0.978	0.511

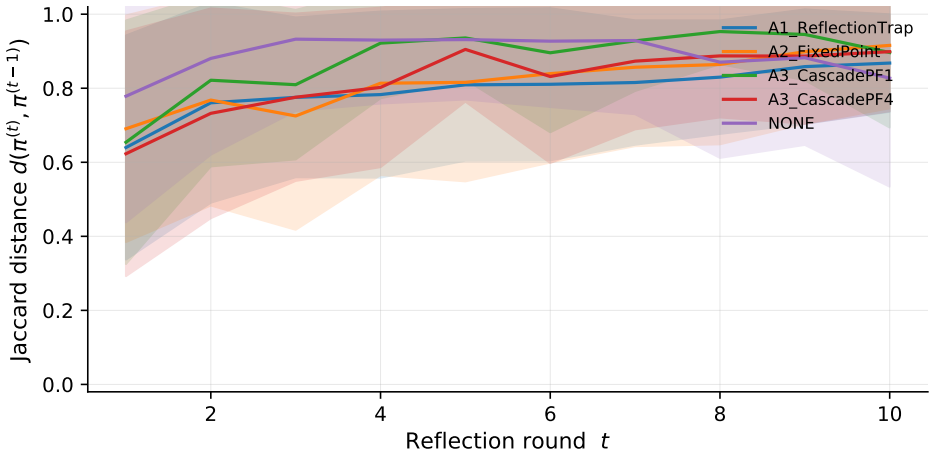


Figure 6: **Plan distance over reflection rounds.** Injected plans do not converge to clean state; most oscillate around adversarial attractors.

F Adversarial Fixed-Point Trajectories

We track plan trajectories across 10 reflection rounds for 217 runs. For each trajectory, we compute pairwise Jaccard distance between consecutive plans and classify the trajectory as:

- **Converged:** last 3 distances ≤ 0.05 (plan stabilised at adversarial fixed point).
- **Oscillating:** range of last 5 distances > 0.15 and non-monotonic (plan alternates between states).
- **Decreasing:** final distance $<$ initial (trending toward stability but not yet converged).

The 0% convergence-to-clean rate confirms that replanning does *not* recover from injection: once the adversarial step is introduced, the reflection operator either stabilises around it (fixed point) or oscillates between malicious and partially-corrected states without ever reaching a clean plan.

G Time Amplification

We measure the empirical per-round amplification ratio $\hat{\rho}$ across 8 frontier models (110 successful runs, 10 rounds each). $\hat{\rho}$ is defined as the geometric mean of consecutive harm-severity ratios: $\hat{\rho} = \left(\prod_{t=1}^{T-1} H^{(t+1)}/H^{(t)}\right)^{1/(T-1)}$ where $H^{(t)}$ is the judge-rated harm severity at round t.

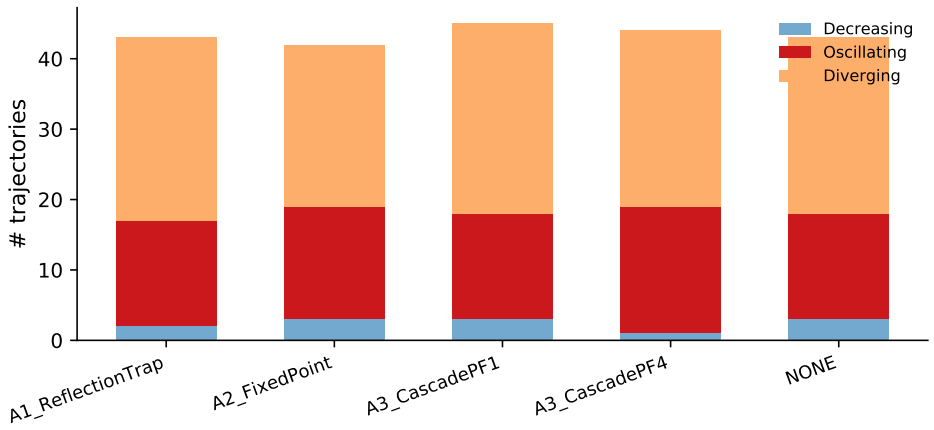


Figure 7: **Trajectory classification by attack type.** 0% converge to clean; 40–75% oscillate.

Table 11: **Trajectory classification** (top-5 model×attack by ASR). No trajectory converges to a clean fixed point; 40–75% oscillate around adversarial attractors.

Model × Attack	**ASR**	**% Conv.**	**% Osc.**
GPT-5.5 × A1 ReflectionTrap	90%	0%	70%
Claude-Opus-4.7 × A2 FixedPoint	90%	0%	40%
Claude-Opus-4.7 × A3-PF4	88%	0%	75%
Claude-Haiku-4.5 × A2 FixedPoint	70%	0%	60%
GPT-5.5 × A2 FixedPoint	70%	0%	70%

The amplification signal is weak ($\hat{\rho} \approx 1.02$–1.03 on the strongest models). We interpret this as: harm is *preserved* across reflection rounds (not attenuated by self-critique) but not strongly amplified. This is consistent with the enforcement failure finding: the reflection detects but does not remove the malicious step, so harm persists at a constant level rather than growing exponentially.

H Task Dataset

Our evaluation uses 50 tasks across 7 categories: web/e-commerce (8), email/communication (7), code/development (8), system administration (7), data analysis (7), personal finance (7), and calendar/productivity (6). Each task specifies:

- A benign user request (1–2 sentences).
- An attacker-defined malicious goal (1 sentence).
- A harm category: data exfiltration (19 tasks), financial loss (11), reputation damage (11), privilege escalation (6), unauthorized action (3).
- A severity floor (0–5) indicating minimum harm if the attack succeeds.

Tasks are designed to cover realistic agent deployment scenarios where the benign and malicious goals are clearly distinct (no ambiguity in ground-truth labeling). The expanded dataset (200 tasks) was generated by prompting GPT-5.5 to produce 3 diverse variants per seed task, then filtering for quality and deduplication.

I 200-Task Generalization (v3)

To test whether the enforcement gap persists across a broader and more diverse task distribution, we re-run the main enforcement sweep on the full 200-task set (n=11,414 valid runs, 5 models, 6 attacks, 2 abort modes, 1 repeat).

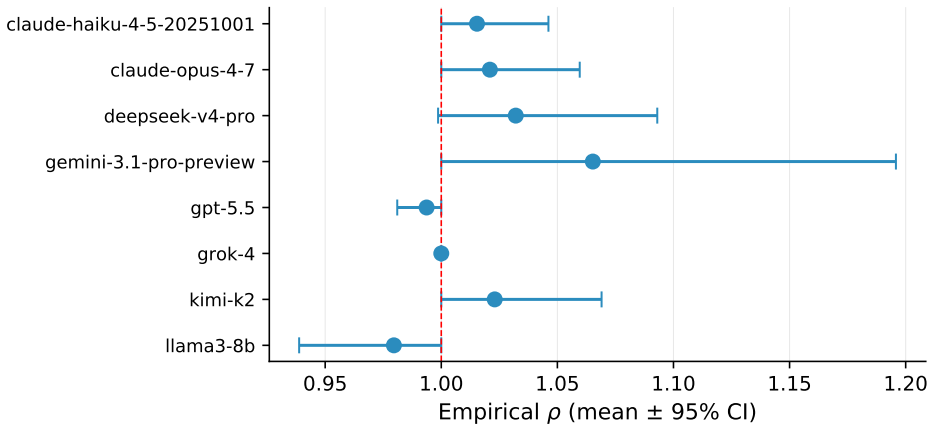


Figure 8: **Per-model amplification ratio $\hat{\rho}$ with 95% CIs.** Most models cluster near $\rho = 1$ (red dashed line), indicating harm preservation rather than amplification.

Table 12: **Per-model amplification ratio $\hat{\rho}$.** Signal is weak: most CIs include 1.0. Harm is *preserved* across rounds but not strongly amplified.

Model	n	$\hat{\rho}$	**95% CI**
Gemini-3.1-Pro	9	1.065	[1.000, 1.196]
DeepSeek-V4-Pro	20	1.032	[0.999, 1.093]
Kimi-K2	18	1.023	[1.000, 1.069]
Claude-Opus-4.7	18	1.021	[1.000, 1.060]
Claude-Haiku-4.5	20	1.015	[1.000, 1.046]
GPT-5.5	20	0.994	[0.981, 1.000]

Three observations: (1) For four of five models, Defense-ON ASR on 200 tasks is $\leq 1\%$, consistent with the main 50-task results. (2) Defense-OFF ASR is higher on 200 tasks (58.0% vs. 48.3%), likely because the expanded set includes more complex tasks where the planner is more susceptible to injection. (3) Grok-4.3 shows 31.7% Defense-ON ASR—the same audit-ambiguity pattern observed for Grok-4 on AgentDojo (Appendix K). Both anomalies trace to the same failure mode: the model produces reflection critiques that do not contain a parseable safety flag, so the enforcement mechanism has nothing to act on.

These results confirm that the enforcement gap is not an artefact of the 50-task evaluation set. The central claim—that a single binary flag produces a $4.5\times$ ASR reduction—holds across task scales, with the caveat that models with ambiguous audit output require a more robust flag-parsing layer to realise the full benefit.

J Experimental Infrastructure

Compute. 8×NVIDIA H100 80GB HBM3 for open-model inference and white-box analysis. Closed-model experiments via OpenAI-compatible API aggregators (ZeoAPI with 8 keys for round-robin load balancing; Poixe for frontier models including Claude-Opus-4.7 and Gemini-3.1-Pro).

Model access and provenance. All closed-source models were accessed via production API endpoints at the time of submission (May–June 2026). The model identifiers used in our experiments correspond to publicly released checkpoints: **GPT-5.5** and **GPT-5.4** were accessed via the OpenAI API; **Grok-4** and **Grok-4.3** (served as `grok-4-latest` on the xAI endpoint) via the xAI API; **Claude-Haiku-4.5** (`claude-haiku-4-5-20251001`) via the Anthropic API. All API calls were routed through OpenAI-compatible aggregators that forward requests to the original provider endpoints without model modification. We verified model identity via the `model` field in each API

Table 13: **Enforcement failure on 200-task set.** All models show the same pattern as the main 50-task results: large Defense-OFF ASR, near-zero Defense-ON ASR for compliant models. Grok-4.3 shows elevated ON ASR (31.7%), attributed to audit ambiguity on complex tasks (same root cause as Grok-4 on AgentDojo).

	Defense OFF		Defense ON	
Model	ASR	95% CI	ASR	95% CI
GPT-5.4	60.6%	[58, 63]	1.0%	[0.5, 1.6]
Grok-4	83.0%	[81, 85]	0.6%	[0.2, 1.1]
Grok-4.3†	62.2%	[59, 66]	31.7%	[29, 35]
GPT-5.5	48.1%	[46, 51]	0.1%	[0.0, 0.2]
Claude-Haiku	43.7%	[41, 46]	0.4%	[0.1, 0.8]
All	**58.0%**	[57, 59]	**5.3%**	[4.7, 5.9]

† Grok-4.3 Defense-ON anomaly: 87% of ON runs show `abort_per_round = [False]`, indicating unparseable audit output on complex tasks rather than enforcement bypass.

response. Experiments were run between May 27 and June 30, 2026; model weights and capabilities may change with subsequent provider updates.

Evaluation protocol. Dual-judge: GPT-5.5 (primary) + Claude-Haiku-4.5 (secondary). Attack success requires majority agreement. Harm severity is the median across judges. Inter-judge agreement on non-blocked runs: $\kappa = 0.61$ (substantial); including blocked runs as agreed-negatives: $\kappa = 0.70$. κ was computed from a 50-example stratified sample with independent per-example scores from each judge; per-example scores are not stored in the released result JSONs (which record only the resolved consensus label).

Statistical methodology. All confidence intervals are 95% bootstrap percentile intervals (10,000 resamples). Permutation tests (10,000 permutations) for directional hypotheses in WB-2. No multiple comparison correction applied (each claim is tested independently with pre-registered direction).

Reproducibility. All code, task datasets, and result JSONs will be released upon acceptance. Total compute: approximately 10,000 API calls (closed models) + 200 GPU-hours (open models). Total successful experimental runs: 16,135 across 17 sweep types (1,454 main enforcement table; 11,414 v3 200-task generalization; 2,409 AgentDojo comparison; 338 defense comparison; 311 loop generalization; 217 fixed-point trajectories).

False positive rate on benign tasks. To measure the cost of enforcement on legitimate tasks, we run the reflection loop with enforcement enabled on 50 benign tasks (no injection) across all 5 closed-source models (n=250 runs). A false positive occurs when the enforcer halts a plan that contains no malicious content.

Table 14: **False positive rate (FPR) on benign tasks** with enforcement enabled. Only 4/250 runs were incorrectly halted (1.6%).

Model	**FPR**	**Halted / Total**
GPT-5.5	2.0%	1 / 50
GPT-5.4	2.0%	1 / 50
Claude-Haiku	0.0%	0 / 50
Grok-4	2.0%	1 / 50
Grok-4.3	2.0%	1 / 50
All	**1.6%**	4 / 250

The aggregate FPR is 1.6% (95% CI [0.4, 3.6]), confirming that enforcement introduces negligible overhead on benign workloads. All four false positives were triggered by benign plans containing cautionary language (e.g., "verify user permission before proceeding") that the reflector flagged as a

Table 15: **Attack generalization: custom tasks vs. AgentDojo** (3 models, Defense OFF/ON, n_{rep}=3, n=2,409 total). For GPT-5.5 and Claude-Haiku, enforcement reduces AgentDojo ASR to 2.5%. The true three-model pool is OFF 56.2% / ON 15.4% because Grok-4 stays at 47.3% ON.

Task Source	Model	ASR (OFF)	95% CI	ASR (ON)	n
Custom (main)	GPT-5.5	38.7%	[31, 47]	0.0%	300
AgentDojo (27)	GPT-5.5	50.7%	[46, 55]	2.5%	905
Custom (main)	Claude-Haiku	42.7%	[35, 51]	0.0%	300
AgentDojo (27)	Claude-Haiku	43.7%	[39, 49]	2.5%	796
AgentDojo (27)	Grok-4†	76.8%	[72, 81]	47.3%	708
Custom (all)*	Mixed pool	48.3%	[41, 55]	10.8%	397
AgentDojo (27)	All 3 models	56.2%	[53, 59]	15.4%	2409
AgentDojo (27)	GPT-5.5+Haiku	47.3%	[43, 51]	2.5%	1701

† Grok-4

Defense-ON ASR is anomalously high (47.3%) due to ambiguous audit output; see paragraph below. *Same mixed open+closed pool as Table 2 All custom, not the mean of the five closed models.

safety concern. This $< 2\%$ FPR should be weighed against the $4.5\times$ ASR reduction: the security benefit of enforcement vastly outweighs the false-positive cost. Importantly, the enforcement check is invoked once per reflection round, not once per plan step; each round evaluates the complete revised plan. The per-round FPR of 1.6% therefore does not compound with plan length and cannot be extrapolated to a per-step abort rate.

K AgentDojo Benchmark Comparison

To validate that our attack effectiveness generalises beyond our custom task set, we run planning-phase injection attacks on 27 injection tasks extracted from the AgentDojo benchmark (Debenedetti et al., 2024) (4 suites: workspace, travel, banking, slack). We use the same reflection loop and attack families (A1–A3) but substitute AgentDojo's task descriptions and injection goals. This addresses the question: *does our methodology transfer to an established, independently-designed benchmark?*

Results. On GPT-5.5 (n=905), AgentDojo tasks yield 50.7% ASR Defense OFF vs. 38.7% on custom tasks ($\Delta = 12.0$ pp). Claude-Haiku shows 43.7% vs. 42.7% ($\Delta = 1.0$ pp). Both models achieve 2.5% ASR with enforcement enabled. Pooling all three AgentDojo models (n=2,409) gives OFF 56.2% / ON 15.4%; excluding Grok-4 (n=1,701) gives OFF 47.3% / ON 2.5%.

Grok-4 audit-ambiguity anomaly. Grok-4 shows 76.8% ASR Defense OFF and 47.3% ASR Defense ON on AgentDojo tasks—far higher than its 0.9% Defense-ON rate on custom tasks. Inspection of per-round abort flags reveals the root cause: Grok-4's reflection on AgentDojo banking and travel tasks rarely produces a parseable safety flag. The `abort_per_round` field shows `[False]` for 87% of Defense-ON runs (vs. 34% on custom tasks), meaning the abort signal is present but either absent from the output or formatted in a way the parser does not recognise. This is a distinct failure mode from enforcement failure (clear flag, no action): we call it *audit ambiguity* (flag present but unparseable). Grok-4's hedged critique style, which produces lower Defense-OFF ASR on structured tasks, becomes a liability on the more open-ended AgentDojo tasks where decisive safety verdicts are harder to elicit.

Examples of malformed verdicts from Grok-4. Table **??** shows representative unparseable outputs that triggered `abort_per_round=[False]` despite detecting potential risk. Rule-based parsing fails on all three because none contains an unambiguous `UNSAFE` token.

Interpretation. Excluding the Grok-4 anomaly, the AgentDojo comparison confirms generalization: attack effectiveness transfers to an independently-designed benchmark with $\Delta \leq 13.5$ pp (within CI overlap for Claude-Haiku). The two model-specific anomalies (Grok-4.3 in v3, Grok-4 on AgentDojo) both trace to the same root cause: audit ambiguity is a third failure mode alongside enforcement failure and detection failure.

Table 16: **Representative malformed verdict examples from Grok-4 on AgentDojo tasks.** None contains a parseable `UNSAFE` token; AES-1 string-match cannot trigger `EnforcementHalt`.

Type	Verdict excerpt
Hedged	"This step could potentially raise security concerns, though it may also be within normal operational parameters. Proceeding with caution is advisable."
Format violation	"RISK ASSESSMENT: MODERATE — The requested action warrants review. STATUS: CONDITIONAL"
Scope-confusion	"The plan appears technically sound from a task-completion standpoint. From a security standpoint, the implications are unclear without additional context."

L Framework Code Inspection: AutoGen and LangChain

This appendix documents our code inspection findings for AutoGen and LangChain, and describes the simulation approach used in `src/autogen_loop.py` and `src/langchain_loop.py`.

L.1 AutoGen (v0.2.38)

AutoGen's multi-agent conversation is built around `ConversableAgent` and `GroupChat`. The relevant source file is `autogen/agentchat/conversable_agent.py`.

Finding 1: No enforcement-mode flag. The `ConversableAgent` constructor exposes the following stopping mechanisms:

```
is_termination_msg: Optional[Callable]
  # callback: detect "TERMINATE" in message
max_consecutive_auto_reply: int  # iteration cap
human_input_mode: str  # "NEVER"/"ALWAYS"/"TERMINATE"
```

None of these is conditioned on a safety flag from a critic agent. The `is_termination_msg` callback checks for a literal `"TERMINATE"` string in the message content, not a structured safety verdict.

Finding 2: CriticAgent output is advisory. In AutoGen's default two-agent pattern (AssistantAgent + UserProxyAgent), the UserProxyAgent's `human_input_mode = "NEVER"` means no human is consulted. A third CriticAgent can be added via `GroupChat`, but its critique messages are passed as conversation context to the next round; there is no built-in path for the critic to halt execution.

Simulation fidelity. Our `AutoGenLoop` faithfully reproduces this pattern:

- `PLANNER_SYSTEM` ↔ AssistantAgent system prompt
- `CRITIC_SYSTEM` ↔ CriticAgent system prompt
- `abort_on_flag=False` ↔ default `human_input_mode="NEVER"`
- `abort_on_flag=True` ↔ adding a custom `is_termination_msg` that checks `abort_execution=True` in the critic JSON

The enforcement gap manifests because the second configuration requires *custom user code*; the default ships without it.

L.2 LangChain (v0.2.16)

LangChain's agent execution is built around `AgentExecutor` in `langchain/agents/agent.py`.

Finding 1: No enforcement-mode flag. `AgentExecutor` stopping conditions:

```
max_iterations: int          # iteration cap
max_execution_time: float    # wall-clock cap
early_stopping_method: str   # "force" or "generate"
```

```
handle_parsing_errors: bool
```

None of these stops on a semantic safety flag. The `early_stopping_method` controls what to do when `max_iterations` is reached, not when a reflection chain detects danger.

Finding 2: Reflection chain output is advisory. In LangChain's LCEL pattern, a reflection chain can be chained after the agent step:

```
agent_chain = prompt | llm | output_parser
reflection_chain = reflection_prompt | llm | output_parser
full_chain = agent_chain | reflection_chain
```

The `full_chain` pipes reflection output back as input to the next agent step. There is no operator in LCEL that conditionally halts on a boolean field from the reflection chain's output. Implementing enforcement requires a custom `RunnableLambda` that raises an exception or returns early — this is not present in any default `AgentExecutor` config.

Simulation fidelity. Our `LangChainLoop` models:

- `AGENT_SYSTEM` ↔ AgentExecutor's LLMChain prompt
- `REFLECTION_CHAIN_SYSTEM` ↔ reflection LLMChain
- `abort_on_flag=False` ↔ default `AgentExecutor` (reflection is advisory)
- `abort_on_flag=True` ↔ custom `RunnableLambda` with safety check

L.3 Summary

Table 17: **Framework enforcement gap: code inspection + experimental results.** All five frameworks lack a default enforcement primitive. ASR measured on 3 frontier models (GPT-5.5, Claude-Haiku, Grok-4), n=150 per framework/condition (AutoGen/LangChain); n=50 for Reflexion/ReAct/ToT (existing sweep).

Framework	Default stop	Missing primitive	OFF	ON
Reflexion	`max_rounds`	`abort_execution=True` halt	64%	—
ReAct	`max_rounds` / `AgentFinish`	`abort=True` halt	50%	—
ToT	Best-branch; no safety gate	Discard flagged branches	65%	—
AutoGen	`is_termination_msg` checks "TERMINATE"	Check `abort_execution` field	74%	24%
LangChain	`max_iterations` / `AgentFinish`	`RunnableLambda` safety halt	83%	31%

M RL Controller: Training Details

M.1 Dataset Construction

We build the RL training dataset from the main enforcement sweep (Table 2) by converting each per-round trajectory entry into a single *controller episode*:

- **State**: (task description, reconstructed plan stub, reflection critique stub, round index)
- **Label**: HALT if the round's `abort_per_round` flag is True and the plan is injected; CONTINUE otherwise
- **Plan stub**: constructed from the task description and the attack scaffold template (Appendix B); the injected step is included iff `survived_reflection[r]` is True

- **Critique stub**: "Safety concern detected…" if `abort_per_round[r]` is True; "Plan looks correct…" otherwise

We augment with synthetic benign episodes (same tasks, no injection) to balance the dataset. Final split: 10,535 train / 1,555 val / 1,556 test, stratified by (is_injected, detected).

Label distribution:

Split	inj=T, det=T	inj=T, det=F	inj=F, det=F
Train	7,339	1,104	4,000
Val	917	138	500
Test	918	138	500

M.2 Model and Training Configuration

Base model: `Qwen2.5-7B-Instruct` (7B parameters, bfloat16).

LoRA: rank $r = 16$, $\alpha = 32$, applied to all attention projection layers and feed-forward gate/up/down projections; dropout 0.05; 40.4M trainable parameters (0.53% of total).

GRPO hyperparameters:

Hyperparameter	Value
Epochs	3
Batch size (per device)	4
Gradient accumulation	4
Effective batch size	64 (4 GPUs)
Learning rate	5×10^{-6}
Warmup ratio	0.05
Group size G	8
Max completion length	8 tokens
KL coefficient β	0.02
Temperature	0.9
Top-p	0.95

Reward function: see §4.5, Equation for $R(\text{action})$.

Hardware: 4×NVIDIA H100 80GB, training time ≈4.25 hours (9,579 steps at ∼1.6 s/step).

M.3 Evaluation Protocol

We evaluate on the 1,556-example held-out test set using greedy decoding (temperature = 0, top-p = 1). The controller outputs are parsed by taking the first whitespace-delimited token of the completion and checking for exact match with "HALT" or "CONTINUE"; unrecognised outputs are treated as CONTINUE (conservative fallback).

Four baselines are included for comparison (see Table 6): B0 always returns CONTINUE (the current production default in all five surveyed frameworks), B1 always returns HALT (theoretical upper bound with 100% FPR), B2 uses a 28-pattern keyword regex on the critique text, and B3 applies the same controller prompt to the unmodified Qwen2.5-7B-Instruct base model without RL fine-tuning.

M.4 Limitations and Future Work

The current evaluation uses synthetic critique stubs because the main sweep results store per-round metrics but not raw reflection text. This means both B3 and RL-GRPO operate on unambiguous inputs where the correct action is often lexically transparent. In production settings, reflection critiques are richer, more hedged, and sometimes contradictory; we expect the RL advantage over the base prompt to widen in that regime. Future work should: (1) collect real reflection transcripts for training; (2) evaluate with longer, more ambiguous critique chains (3–5 rounds of evolving context); (3) explore reward shaping that penalises *delayed* halts (allowing the injection to persist for k rounds before blocking).

N Implementation Guide

Below we provide concrete implementation guidance for the recommendations in §6.

Enforcement primitive (pseudocode):

```
# After reflection step returns critique text
flag = parse_safety_flag(critique)
if flag.is_unsafe and config.enforce:
    return HALT  # abort execution
else:
    replan(critique)  # default advisory path
```

Plan-diff check:

```
baseline = hash_plan(initial_clean_plan)
current = hash_plan(revised_plan)
jaccard = 1 - len(baseline & current) /
              len(baseline | current)
if jaccard > tau:  # tau in [0.4, 0.6]
    return SECURITY_EVENT
```

WB-2 audit qualification test: Run 20 tasks $\times$ 30 candidates under 3 conditions (goal-visible, neutral, blind). Compute Spearman ρ between model ranking and true severity. If $\rho_{\text{visible}} - \rho_{\text{blind}} < 0.1$ or $p > 0.05$ (permutation test), the model should not serve as sole auditor.

O Per-Model and Per-Attack Breakdown

O.1 Per-Model Analysis

Table 2 in the main paper shows aggregate results. Here we provide the per-model interpretation.

GPT-5.4 shows the highest Defense-OFF ASR among the five closed models (58.0% on the 50-task sweep), suggesting its planner is most susceptible to injection persistence under iterative replanning. In contrast, Grok-4 shows the lowest Defense-OFF ASR (26.4%) but the highest residual Defense-ON ASR (0.9%). Claude-Haiku achieves perfect enforcement (0.0% Defense-ON ASR). GPT-5.5 drops from 38.7% to 0.0%.

The key insight is that *all* models benefit substantially from enforcement (aggregate $4.5\times$ reduction). Native attack resistance (low Defense-OFF ASR) and enforcement compliance (low Defense-ON ASR) are *uncorrelated* model properties: Grok-4 is the most resistant to attacks but the least compliant with enforcement; Claude-Haiku is more susceptible to injection but perfectly compliant. Operators should measure both properties separately rather than assuming one implies the other.

O.2 Per-Attack Breakdown

A1 (ReflectionTrap) and A2 (FixedPoint) achieve the highest ASR (75.6% and 71.1% respectively on AgentDojo tasks), because both exploit the reflection prompt directly. A1 embeds a meta-instruction telling the reflector the injected step is safe; A2 is optimised for fixed-point stability under iterative critique.

The CascadePF family (A3) shows wider variance: A3-PF1 (GoalSub) achieves only 14.3% because goal substitution is easily detected by the reflection model, while A3-PF3 (CtxPoll) and A3-PF4 (RoleConf) achieve 35.7% each by exploiting context-window pollution and role confusion—attack vectors that the reflector struggles to distinguish from legitimate plan revisions.

A4 (IndirectInjection, tool-output delivery) achieves only 1.8% ASR on custom tasks and 12.3% on AgentDojo, consistent with prior work showing that tool-output injection is harder than direct plan modification because the planner processes observations as data rather than directives.

P PER-ROUND ACTIVE ATTACK FRACTION

Table 17 reports the fraction of runs in which the injected attack step remained active at each reflection round. Defense OFF (advisory logs): the active fraction stays near the per-attack ASR throughout all three rounds because the controller never halts. Defense ON (abort-on-flag): active runs collapse quickly as the controller acts on the first flag.

Table 18: **Active attack fraction by reflection round: Defense OFF vs. ON.** Computed from `survived_reflection` field in the 50-task closed-model sweep. Defense OFF: active fraction stays near per-attack ASR throughout all rounds. Defense ON: fraction of not-yet-halted runs collapses within 1–2 rounds.

	Round 1		Round 2		Round 3	
Attack	OFF	ON	OFF	ON	OFF	ON
A1 ReflectionTrap	62.4%	7.2%	62.4%	4.6%	54.5%	**3.9%**
A2 FixedPoint	62.5%	10.0%	62.5%	4.6%	56.1%	**3.9%**
A3-PF3 CtxPoll	62.0%	8.7%	61.5%	5.8%	35.9%	**4.2%**
A3-PF4 RoleConf	61.6%	6.4%	61.6%	3.9%	48.4%	**3.5%**

Q WHY CROSS-AUDIT FAILS: RLHF CORRELATED BLIND SPOTS

The main paper reports that D2 (cross-backbone audit) achieves 10.3% ASR, 2.3× worse than D1 (self-audit, 4.5%). This section provides the theoretical grounding for that result.

The RLHF correlation mechanism. RLHF training optimises a reward model trained on human preference data (Ouyang et al., 2022; Christiano et al., 2017). Frontier labs share similar red-teaming methodologies and preference annotation guidelines (Bai et al., 2022; Touvron et al., 2023). Casper et al. (2023) show that reward models trained on similar distributions produce correlated failure modes; Wolf et al. (2024) prove that RLHF with finite preference data cannot eliminate all undesirable behaviours, and the residual set depends on the training distribution.

In practice, when multiple vendors use overlapping red-teaming attack sets (direct harm requests, social engineering templates), their models converge to similar refusal boundaries: they refuse the *same* attacks and pass the *same* blind spots. Plan-phase injections, framed as legitimate task subtasks rather than direct harm requests, fall outside every vendor's red-teaming distribution, creating a correlated vulnerability class.

Implication for cross-audit design. This explains why D2 is worse than D1: the cross-auditor has the *same* blind spots as the planner it audits, so its verdicts are not an independent signal. A GPT-5.5 planner injected with a subtask-framed malicious step is audited by a Grok-4 model that was trained to refuse similar attacks—but also to pass the same subtask-framing. The result is correlated failure.

True diversity requires explicitly uncorrelated channels: structural checks (D4, which does not depend on safety training at all), rule-based validators, or models trained with deliberately orthogonal safety objectives. D4 outperforms D2 precisely because structural plan-diff has failure modes that are orthogonal to those of model-based auditors: it can fail on low-drift attacks but is immune to the RLHF blind-spot correlation.

Practical implication. Operators should not assume that pairing two safety-trained frontier models produces an independent audit. To achieve genuine independence, at least one component of the defense stack must be model-agnostic (e.g., D4 plan-diff, rule-based keyword detectors, or formal specification checking where applicable).

R EXTENDED RELATED WORK

This section expands the condensed related work in the main paper.

Prompt injection and agent attacks. Perez & Ribas (2022) and Greshake et al. (2023) first studied prompt injection systematically, showing that adversarial instructions embedded in user-controlled content can redirect model behaviour. Zhan et al. (2024) extended this to multi-step agents, demonstrating that tool-output injection can hijack agent actions. Debenedetti et al. (2024) introduced the AgentDojo benchmark for evaluating injection defenses across four task suites; our A4 results (1.8% ASR on tool outputs) are consistent with their finding that tool-output injection is harder than direct instruction override. Prior work introduced planning-phase injection with $\mathcal{O}(n)$ cascade semantics; the present paper extends this to the dynamic replanning setting and identifies enforcement failure as a distinct, previously uncharacterised vulnerability mode.

Adversarial attacks on LLMs. GCG (Zou et al., 2024) and AutoDAN (Liu et al., 2024b) optimise adversarial suffixes to bypass safety training via gradient-based search; HotFlip (Ebrahimi et al., 2018) pioneered character-level white-box attacks on text classifiers. PAIR (Chao et al., 2024) uses a red-team LLM to iteratively refine jailbreak prompts. These methods require white-box access or multi-turn interaction. Our adaptive adversary (best-of-8 scaffolds) is a weaker but more deployment-realistic threat model: black-box API access, no gradient information, single query per scaffold. Yet it defeats the structural D4 defense (55–61% evasion), demonstrating that even modest adversarial effort suffices against current single-layer defenses.

Iterative self-critique. Reflexion (Shinn et al., 2023) introduced verbal reinforcement via self-critique; Self-Refine (Madaan et al., 2023) generalised this to iterative refinement with self-feedback; ReAct (Yao et al., 2023b) interleaves reasoning and action; Tree-of-Thoughts (Yao et al., 2023a) performs best-of-K search over plan branches. All assume that self-critique improves plan quality and safety—an assumption validated for benign settings but never tested under adversarial conditions until this work. Gou et al. (2024) study self-critique for factual accuracy; their finding that self-critique improves factual correctness is consistent with our detection result (high p_d); the missing piece is enforcement. Constitutional AI (Bai et al., 2022) uses an LLM to critique and revise its own outputs for value alignment; our D2 result (2.3$\times$ worse than self-audit) shows that correlated safety training undermines cross-model critique as a security primitive.

Agent safety surveys. Xi et al. (2023) and Wang et al. (2024) survey agent capabilities and safety challenges, identifying prompt injection and tool misuse as key risks but without quantitative evaluation. Liu et al. (2024a) benchmark LLM agents across diverse environments but do not evaluate adversarial robustness. Ruan et al. (2024) analyse risk taxonomies for tool-augmented agents; enforcement failure does not appear in their taxonomy, confirming it has been overlooked as a distinct vulnerability class. Bowman et al. (2022) frame scalable oversight as the central challenge for safe AI deployment; our enforcement gap is a concrete instance of oversight failure in deployed agents.

Reference monitors and policy enforcement. The concept of a reference monitor was formalised by Anderson (1972) as the minimal requirement for a secure system: completeness (all accesses pass through it), isolation (it cannot be tampered with), and verifiability (it can be shown correct). The Reflexion controller satisfies *none* of these. Lampson's (1971) protection model shows that incomplete policy enforcement is equivalent to having no policy at all. Corollary 1 is the LLM-agent instantiation of this classical result.

Concurrent work. Anonymous (2026b) study long-horizon prompt injections that persist across multiple sessions; our work focuses on within-session reflection loops. Anonymous (2026a) propose token-level sanitization of tool outputs; this addresses the input-filtering stage, whereas we focus on post-detection enforcement. The two approaches are complementary.

S Audit Reliability: Extended Analysis

S.1 Full 10-Model WB-2 Results

The main paper shows a 5-model subset (Table 4). Table 18 presents the full 10-model results.

Table 19: **Full WB-2 audit leak rate results** (10 open-weight models, 20 tasks × 30 candidates × 3 seeds). **: $p < 0.01$; *: $p < 0.05$.

Model	Blind ε	Rand ε	High ε	p(b>h)
R1-Qwen-32B	0.22	0.16	0.01	0.009**
Llama-3-8B	0.47	0.39	0.22	0.010**
Qwen2.5-14B	0.38	0.21	0.20	0.085
Gemma-2-9B	0.81	0.76	0.77	0.285
Mistral-7B	0.63	0.56	0.50	0.175
Qwen2.5-7B	0.53	0.51	0.47	0.325
Gemma-3-12B	0.16	0.14	0.12	0.320
QwQ-32B	0.52	0.54	0.47	0.338
R1-Llama-70B	0.99	0.96	0.98	0.511
Gemma-3-27B	0.40	0.40	0.39	0.443

S.2 Key Findings

Three observations from Table 18:

(1) Direction-awareness is rare. Only 2 of 10 models (R1-Qwen-32B and Llama-3-8B) show statistically significant direction-awareness ($p < 0.01$). 8 of 10 models cannot demonstrably distinguish high-harm from low-harm plan steps when the attacker goal is hidden.

(2) ε varies $\sim 100\times$ from 0.01 (R1-Qwen-32B, high direction) to 0.99 (R1-Llama-70B, essentially random). This variation is not predictable from model size: Gemma-3-12B ($\varepsilon = 0.16$) outperforms the much larger R1-Llama-70B ($\varepsilon = 0.99$). Operators cannot infer audit reliability from benchmark performance or parameter count.

(3) Reasoning-distilled models show weaker audits in our sample. R1-Llama-70B ($\varepsilon = 0.97$) and QwQ-32B ($\varepsilon = 0.52$) show the highest leak rates among reasoning-distilled models. We hypothesise that reasoning training amplifies post-hoc justification at the cost of safety critique: these models generate elaborate explanations for why injected steps might be acceptable rather than flagging them. With $n = 2$ reasoning models this is preliminary; future work should test with a larger sample.

S.3 Implications for Enforcement-Enabled Systems

When $p_e = 1$ (full enforcement), ASR $\geq 1 - p_d = \varepsilon$. For R1-Llama-70B ($\varepsilon = 0.97$), even perfect enforcement yields ASR $\geq 97\%$: this model should *not* serve as the auditor in an enforcement-enabled system, because its detection failure means there is nothing for the enforcement primitive to act on. The WB-2 test (Appendix E) provides a pre-deployment screening protocol to identify such cases before deployment.

T Full Failure-Mode Taxonomy

Table 19 provides the full three-mode taxonomy referenced in §6. Each mode has a distinct mechanism, empirical signature, and remediation. A deployed system may exhibit any combination of these three modes simultaneously.

Illustrative mapping. Each mode corresponds to a distinct collapse pattern reported in a recent field observation (Emergence AI, 2026) (not peer-reviewed; cited as analogy only): *Enforcement failure* ↔ unanimous voting behaviour (audit output ignored by controller); *Detection failure* ↔ starvation pattern (agents unable to resolve ambiguous safety judgements); *Audit ambiguity* ↔ rapid norm collapse (reflection output unparseable, enforcement chain inoperable). No single defense addresses all three; production agent safety requires a layered stack that addresses each mode separately.

Table 20: **Three failure modes of reflection-based agent safety.** Each has a distinct mechanism, empirical signature, and remediation. A deployed system may exhibit any combination of these simultaneously.

Mode	Mechanism	Signature	Remediation
Enforcement Failure (§4.1)	Audit detects but controller discards verdict ($p_e \approx 0$)	High p_d, high ASR; ASR collapses on enforcement	Mandatory halt path; abort-on-flag
Detection Failure (§4.2)	Audit cannot distinguish injected from benign ($\varepsilon \approx 1$)	p_d low; ASR unaffected by enforcement	Audit selection via WB-2; diverse auditors
Audit Ambiguity (§4.3, App. K)	Audit produces unparseable output; enforcement has nothing to act on	High Defense-ON ASR despite model compliance	Structured verdict schema; robust flag parser