Policy-Verified Self-Training: Learning Agent Compliance from Verifiable Policy Rewards

Anonymous authors
Paper under double-blind review

Abstract

Runtime policy verifiers can prevent agents from executing disallowed actions, but their decisions are typically used only for enforcement and discarded afterward. We ask whether these decisions can instead become verifiable policy rewards for training the agents they guard. We introduce Policy-Verified Self-Training (PVST), a post-training paradigm in which an agent generates its own interaction trajectories and an executable policy verifier, here a deterministic gate compiled from the natural-language policy, deterministically evaluates whether each action complies. These verdicts provide verifiable training signals: compliant trajectories are reinforced, and violations are used to construct negatives for preference optimization. PVST thus turns an existing runtime guardrail into a teacher for the same agent: beyond the gate and the benchmark's task-success signal, it needs no stronger teacher model, no human preference labels and no attack-specific supervision, although rollouts are collected in the benchmark's attacked environment.

We evaluate PVST on tool-using agents under prompt injection. On held-out AgentDojo tasks, policy-verified preference training of Qwen3-8B reduces violating trajectories from 37.2% to $10.3 \pm 0.9\%$ (three seeds) and executed attacks from 12.4% to 0.5% at equal or higher task utility, compared with 22.0% violating trajectories when negatives are selected by an attack oracle and $31.9 \pm 0.9\%$ with matched negatives that contain no violation (a success-only control that never consults the gate lands at 20.2%). This separation indicates that the improvement comes from the policy information encoded by the verifier rather than from preference optimization alone. The learned behavior also transfers beyond the training distribution: it holds under an attack template never used in training (37.7% to 11.0%), violations decrease from 34.8% to 16.7% on a suite whose rule set and tasks were excluded from training, and measurable improvements emerge from a few dozen preference pairs. When the original verifier is retained at deployment, its intervention rate falls from 39.2% to 8.8%, reducing reliance on runtime enforcement while preserving a deterministic safety backstop; the compliance result replicates on a second trainee (Qwen3-4B-Instruct-2507, 37.3% to 10.2%).

Our analysis identifies the condition under which the verdict is most informative: guarded actions must appear in both permitted and forbidden contexts. Executable policies can therefore serve not only as runtime constraints but as a scalable source of verifiable rewards for self-training, and they motivate learning objectives that exploit the verifier's finer-grained rule- and predicate-level structure.

1 Introduction

Tool-using language-model agents are deployed under written policies. An airline assistant may cancel a reservation only after the customer confirms; a banking assistant may transfer

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.

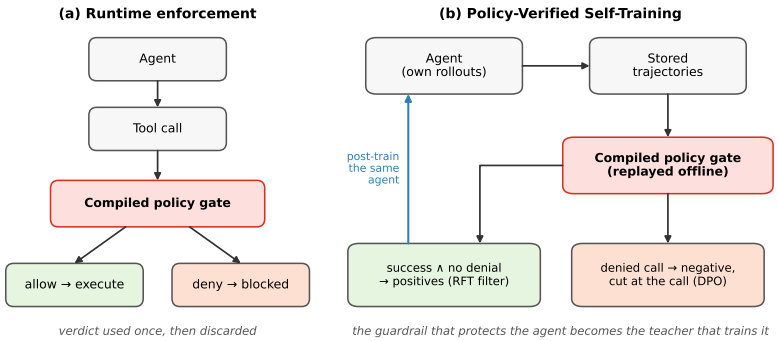

Figure 1: Left: a runtime gate blocks a forbidden call and the decision is discarded. Right: PVST replays the agent's own rollouts through the same gate, keeps compliant trajectories as positives and cuts violating ones at the denied call as negatives, and post-trains the agent; at deployment the gate stays in place and intervenes a fifth as often. The guardrail that protects the agent becomes the teacher that trains it.

money only to a recipient the account holder named. Violations are common for small open-weight models and dangerous because they are executed: a wrong `send_money` call moves money. The prevailing defence is a runtime gate that inspects each tool call and refuses the ones the policy forbids. Recent work compiles such a gate from the policy text itself (Anonymous, 2026), yielding a deterministic checker whose decisions can be replayed and audited. We take the compiled gate as given: this paper contributes nothing to the compiler and everything to what the gate's decisions can be used for.

A runtime gate has two costs. It is a permanent tax on latency and, when the policy is stricter than the task requires, on utility; and it does not change the model, which keeps attempting the same forbidden calls. This paper asks whether the gate's verdict can instead be used to *train* the agent, so that compliant behavior moves into the model and the gate becomes a rarely-triggered backstop. We call the resulting recipe **Policy-Verified Self-Training (PVST)**: an agent generates its own rollouts, an executable policy verifier deterministically judges each action, and the verdicts become the acceptance filter and the preference labels with which the same agent is post-trained (Figure 1). Unlike teacher- or attack-driven alignment, PVST needs no stronger teacher, no human preference labels and no attack labels: the only signals consumed are the gate's verdict and the benchmark's task-success flag, on rollouts collected in the benchmark's attacked environment.

The verdict has the properties that reinforcement learning with verifiable rewards exploits in mathematics and code, where a checker can grade an answer: it is **deterministic** (a fixed program, not a learned reward model), **annotation-free** (every rollout can be scored automatically) and **traceable** (each denial names a rule). We do not propose a new optimizer; the contribution is the source and structure of the supervision, consumed by existing rejection-sampling and preference-optimization recipes. Policy compliance for agents has lacked such a checker; a compiled gate supplies one. Traceability matters beyond bookkeeping: every reward carries the rule that produced it, so a failure can be attributed to the model, the data, the policy or a predicate, and Section 6.3 turns this into a pre-training check on the predicates. Taken together, our results are evidence that what the agent learns from this supervision goes beyond attack-specific patterns: it survives an unseen attack template and transfers, in part, to a suite whose rule set and tasks it never trained on.

Contributions.

1. **Policy as supervision.** An executable policy checker can be turned into trajectory-level supervision: its verdicts induce an acceptance filter (rejection-sampling fine-tuning) and a preference relation (DPO) over an agent's own rollouts, with no teacher, preference labels or attack labels (Section 3).

2. **Policy-centric alignment.** Negatives chosen by the checker teach compliance that attack-labelled negatives do not (10.3% vs 22.0% violating trajectories at equal utility), and a control whose negatives contain no violation does not learn it at all (31.9%, three seeds each). Both halves of the verdict matter: *which* trajectories are negatives and *where* they are cut. Cutting the same gate-selected negatives at a random turn instead of the denied call leaves 16.9% violating, and cutting attack-labelled negatives at a write call does not recover the gate result (19.8%; Section 5.1).
3. **Generalization beyond the training data.** The gain transfers to a suite with its own compiled rule set and tasks, none of which were used in training (34.8% to 16.7%, within 2.6 points of a model trained on that suite) and holds under an attack template unseen in training (37.7% to 11.0%); substantial gains already emerge from a few dozen verifier-labelled pairs (Sections 5.3 and 5.4).
4. **Learned compliance with runtime verification.** Gate interventions fall from 39.2% to 8.8% of trajectories while the gate stays as a deterministic boundary (Section 5.2), and we characterize the two conditions under which the verdict is most informative: when task success does not already imply compliance, and when guarded actions appear in both permitted and forbidden contexts (Section 6).

2 Related Work

Runtime guardrails for agents. AgentDojo (Debenedetti et al., 2024) benchmarks tool-calling agents under prompt injection and ships defences such as tool filters and spotlighting. A concurrent line of work (Anonymous, 2026) compiles a policy document into predicate-guarded rules and enforces them in the tool executor. We reuse that gate unchanged, describe it fully in Section 3.1 and Appendix A so that this paper is self-contained, and treat its verdicts as data.

Prompt-injection training. SecAlign (Chen et al., 2025) fine-tunes with preference pairs constructed from injected prompts; the instruction hierarchy (Wallace et al., 2024) trains models to privilege system over tool text. Both construct attacks for training. Our negatives are trajectories a policy checker rejected, whatever caused the rejection; Section 5.1 compares directly against negatives labelled by the attack oracle.

Verifiable rewards and self-training. Rejection-sampling fine-tuning (Yuan et al., 2023) and ReST (Gulcehre et al., 2023) fine-tune on a model's own filtered samples; DPO (Rafailov et al., 2023) optimizes preferences without a reward model; GRPO (Shao et al., 2024) drives DeepSeek-R1-style RL with verifiable rewards. We apply the two offline regimes with a verifier that grades policy compliance rather than answer correctness.

Business-policy agents. τ-bench and τ^2-bench (Yao et al., 2024; Barres et al., 2025) evaluate multi-turn agents against domain policies with a simulated user. We use τ^2-bench airline as a boundary case (Section 5.5).

3 Method

3.1 The compiled gate

The verifier is a rule set compiled from the suite's natural-language policy (Anonymous, 2026). Each rule is a tuple $(id, kind, predicate, tools)$. A *forbidden* rule denies a call to any listed tool when its predicate holds; a *precondition* rule denies the call unless the predicate holds. Predicates come from a fixed vocabulary of five (Table 9) and are evaluated over a **fact store** rebuilt from the conversation before each call. The store keeps provenance: text from genuine user turns (`user_text`), text from tool results (`observed_text`), counterparties that arrived as the sender or recipient *field* of a returned account record (`counterparties`), sources the user delegated to, and a `user_confirmed` flag that is set only when an affirmative user turn answers a question the assistant asked, and is consumed by the next guarded write. Nothing that arrives through a tool result is ever promoted into the user channel,

which is what makes injected instructions unable to satisfy the user-authority predicates. The compiler maps each obligation sentence of the policy to such a rule, restricted to this vocabulary, and the result passes a static check before use. The gate is deterministic and can be replayed over a stored trajectory exactly as it ran live, which is the property we rely on. Appendix A lists every rule and defines every predicate.

3.2 Reward and how it is used

For a trajectory τ with executed tool calls $c_1 \dots c_m$, let $v(\tau)$ be the number of calls the gate would deny when replayed with the facts available at each call, and $u(\tau) \in \{0, 1\}$ the benchmark's task-success flag. The reward is $r(\tau) = u(\tau) - \lambda\, v(\tau)$. We do not optimize r directly; its two components induce a binary acceptance relation ($u = 1 \wedge v = 0$) and a preference relation (a trajectory with $v = 0$ is preferred to one with $v > 0$ on the same task), and it is these relations that the two training regimes below consume. In this binary instantiation only the sign of the violation term matters, so λ is not tuned; the scalar form is retained because it is what an online policy-optimization method would consume. AgentDojo's *security* flag, which records whether the injected goal was achieved, is never read during training except by the DPO-oracle baseline, which uses it deliberately (Section 3.5); everywhere else it is an evaluation oracle.

3.3 Gate-filtered rejection sampling (RFT-gate)

The trainee samples k trajectories per training task at temperature 1.0. Trajectories with $u = 1$ and $v = 0$ form the positive set (at most 8 per task, sampled at random, never by length) and a LoRA adapter (Hu et al., 2022) is fine-tuned on them, assistant tokens only. A control (RFT-succ) uses $u = 1$ alone, keeping trajectories that completed the task while executing an injected action.

3.4 Gate-labelled preference optimization (DPO-gate)

From the same rollouts we build pairs within each task. The *chosen* trajectory is a positive as above. The *rejected* trajectory has $v > 0$ and is truncated immediately after the first assistant turn containing a denied call, so that the negative signal sits on the offending action. Training continues from the RFT-gate adapter with DPO ($\beta = 0.1$) plus a supervised term on the chosen side (weight 0.3), one epoch; the reference policy is the RFT-gate adapter.

3.5 Controls and baselines

All preference arms share the chosen trajectories, optimizer, hyper-parameters and initialization and differ only in how negatives are drawn:

- **DPO-ctrl-any**: negatives are failed trajectories ($u = 0$) regardless of the verdict, cut at a random assistant turn. The gate is not consulted when building this arm, so it is the natural success-only control; because most failures under attack are also violations, its negatives carry violation signal by accident (29 of 53 violate).
- **DPO-gate, random cut** and **DPO-oracle, write cut** (Section 5.1 only): the DPO-gate negatives cut at a random assistant turn, and the DPO-oracle negatives cut immediately after their first write call, isolating the *where* from the *which* of the verdict.
- **DPO-ctrl-clean**: negatives are failed trajectories with $v = 0$, cut at a random assistant turn. Here the gate is used once, to *remove* violations from the negatives, so the negatives carry no violation signal at all; this isolates what preference optimization on failures teaches when the negatives contain nothing the policy forbids. Together the two controls bracket the verdict-informed arm from below (0% violating negatives) and from the middle (55%).
- **DPO-oracle**: negatives are trajectories in which the injected goal was achieved according to the benchmark's attack oracle, cut at a random turn. This is the attack-labelled supervision that prior work constructs by hand; here it uses the evaluation oracle and is therefore an upper bound on what attack supervision can know.

Table 1: Held-out user tasks × all injection tasks, `important_instructions`, gate off, three suites pooled ($n \approx 240$). Mean ± sd over seeds; brackets: 95% per-trajectory bootstrap CIs, seed 0.

Arm	Negatives	Seeds	Utility %	ASR %	Violating traj. %	util_w %	util_r %
Base	–	–	35.5	12.4 [8.3, 16.5]	37.2 [31.0, 43.4]	21.5	53.3
RFT-succ	–	1	35.4	5.4	30.8	18.7	56.6
RFT-gate	–	3	40.2 ± 3.3	5.5 ± 1.2	28.8 ± 2.0	21.3 ± 4.9	65.0 ± 3.3
DPO-gate	gate-rejected, cut at denied call	3	39.9 ± 1.9	$\mathbf{0.5} \pm 0.2$ [0.0, 1.2]	$\mathbf{10.3} \pm 0.9$ [5.6, 12.8]	21.1 ± 2.6	65.1 ± 0.5
DPO-ctrl-any	any failure (55% violating)	3	45.1 ± 0.2	1.2 ± 1.2	20.2 ± 2.8 [14.5, 24.5]	20.6 ± 1.1	75.7 ± 0.9
DPO-ctrl-clean	violation-free failures	3	41.0 ± 3.4	6.5 ± 1.1	31.9 ± 0.9 [26.3, 38.4]	20.6 ± 3.4	69.3 ± 6.0
DPO-oracle	attack-oracle positives	1	40.2	0.4	22.0 [16.9, 27.2]	19.0	69.2

4 Experimental Setup

Model and serving. Qwen3-8B (bf16, vLLM, thinking disabled, Hermes tool parser) is the trainee; evaluation uses temperature 0 on the same server, so base and trained models share serving, quantization and decoding. LoRA rank 16 on all projection matrices; SFT lr 10^{-4}, 2 epochs; DPO as above. Three training seeds for RFT-gate, DPO-gate, DPO-ctrl-any and DPO-ctrl-clean; the rollouts and pair sets are fixed, and the seed changes only LoRA initialization, minibatch order and pair order. The base model's evaluation is deterministic. Headline numbers in the abstract, contributions and Table 1 are three-seed means (DPO-gate 10.3 ± 0.9% violating trajectories); every other table and all diagnostic analyses use seed 0 (DPO-gate 9.2%), and say so in their captions.

AgentDojo. Suites banking (v1.1.1, 16 user × 9 injection tasks), travel (v1.2, 20×7) and workspace (v1.2.1, 40×14); slack is excluded because its tasks are judged on output text rather than tool calls. Training attack: `important_instructions`. We hold out 30% of the user tasks of each suite (5/16, 6/20, 12/40; seed 20260919, fixed before training) with all their injection tasks. Rollouts: 4 samples per training pair, attacked and benign, 2,351 scored trajectories; positives 158 (RFT-gate) or 181 (RFT-succ); pairs 113 (DPO-gate), 73 (ctrl-clean), 93 (ctrl-any), 113 (oracle).

Metrics (held-out). *Utility*: task completed. *ASR*: attack oracle. *Violating trajectories*: fraction with $v > 0$ under replay. util_w / util_r: utility on user tasks whose ground truth requires a write tool versus the rest; a model that stops writing keeps util_r and loses util_w. Each model is evaluated undefended and with the gate live, attacked and benign. Some workspace pairs exceed the 32k context for some models and are reported as errors, not failures; Table 1 lists n per arm, and Appendix E repeats the main comparison on the 230 held-out triples evaluated without error by every arm, with confidence intervals from a cluster bootstrap over user tasks (each resampled with all its injection tasks), which respects the dependence between attacks on the same task. Per-trajectory bootstrap intervals in Table 1 are narrower and should be read as descriptive.

τ^2-bench airline. 50 tasks, pre-registered split 30/20, user simulator Qwen3-8B, 3 trials, temperature 0; violation rate on WRITE calls against hand-written reference clauses (Appendix A). This track includes a first-stage distillation from gate-on gemma-4-26B rollouts and is reported as a boundary case.

5 Results

5.1 Main result and controls

The gain comes from the policy signal, not from preference optimization alone. Self-training on the model's own successful trajectories already helps (RFT-succ, RFT-gate): ASR halves and violations fall by a fifth at no utility cost. Using the verdict as a *filter* adds little over success alone (28.8 vs 30.8), because most successful trajectories are already clean (Section 6.1). Using the verdict to choose *negatives* is what moves compliance: DPO-gate reduces violating trajectories to 10.3% and executed attacks to 0.5% with utility

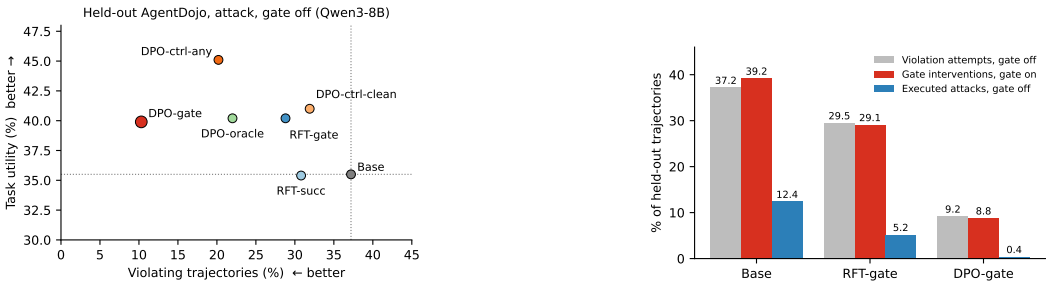

Figure 2: Left: violating trajectories against utility per arm (held-out AgentDojo, attack, gate off; three-seed means where available). Right: violation attempts with the gate off, gate interventions with the gate on, and executed attacks with the gate off, for the base model and the two gate-trained arms (seed 0).

Table 2: Per suite (utility / ASR / violating trajectories, %), seed 0.

Suite	Base	RFT-gate	DPO-gate	DPO-ctrl-clean
banking ($n = 45$)	40.0 / 24.4 / 37.8	40.0 / 4.4 / 13.3	40.0 / 0.0 / 0.0	40.0 / 11.1 / 24.4
travel ($n = 42$)	33.3 / 26.2 / 45.2	31.0 / 14.3 / 26.2	16.7 / 0.0 / 0.0	23.8 / 23.8 / 45.2
workspace ($n \approx 160$)	34.8 / 5.2 / 34.8	43.9 / 3.0 / 34.8	46.0 / 0.6 / 14.1	44.8 / 0.7 / 31.0

4 points above base and utility on write-requiring tasks unchanged. The clean control, identical except that its negatives contain no violation, stays at 31.9% (seed intervals do not overlap; on the common evaluation set the task-clustered 95% CI of the difference is [-44.9, -11.0] points, Appendix E). The success-only control that never consults the gate lands in between ($20.2 \pm 2.8\%$), because its randomly drawn failures happen to violate 55% of the time: the reduction is monotone in the share of violating negatives (0%, 55%, 100% → 31.9%, 20.2%, 10.3%). **Policy supervision is not attack supervision.** Attack-labelled negatives (DPO-oracle) reach the same ASR (0.4%) but leave 22.0% of trajectories violating: the attack oracle only marks trajectories in which the injected goal succeeded, whereas the gate marks every forbidden call, so it teaches the policy rather than the attack. Subsampling DPO-gate to the control's 73 pairs gives 6.2% (seed 0), so the gap is not a pair-count effect.

Selection and localization are both part of the signal. A gate verdict names a trajectory *and* the call at which it went wrong, and we separate the two by changing one at a time (seed 0, same positives and hyper-parameters). Keeping the gate-selected negatives but cutting them at a random assistant turn instead of the denied call raises violating trajectories from 9.2% to 16.9% at unchanged utility (39.8%) and 0.0% ASR. Selection alone, at the same random cut as the controls, is therefore already below both of them (on the common evaluation set 13.0% versus 20.4% for ctrl-any and 32.6% for ctrl-clean; task-clustered 95% CIs of the differences [-15.0, -1.3] and [-32.7, -8.8] points, Appendix E), and localization removes a further 6.9 points [0.0, 17.3], a point estimate that roughly halves the residual but whose interval touches zero with one seed. Conversely, giving the attack-labelled negatives a localized cut at a write call moves them only from 22.0% to 19.8% (utility 35.1%; difference +1.8 [-3.0, 8.0]): a localized cut helps little when the trajectory was selected for the wrong reason, because the write it cuts at is not necessarily the forbidden one, and the write-cut oracle arm remains 14.3 points [2.1, 29.0] above DPO-gate. Section 5.6 shows what localization buys on the permitted side.

Banking and workspace improve on every axis (Table 2); travel loses utility (Section 6.2).

5.2 The gate at inference

The gate alone removes executed attacks at a utility cost (35.5 → 32.2; travel 33.3 → 19.0). After training, the gate intervenes in 8.8% of trajectories instead of 39.2% (Figure 2; full table in Appendix C), a 78% relative reduction, and its residual interventions are exactly

Table 3: Unseen attack template: models trained with `important_instructions`, evaluated with `tool_knowledge` (held-out, gate off, seed 0).

Arm	Utility %	ASR %	Viol. traj. %
Base	34.4	14.3	37.7
DPO-gate	38.6	1.2	11.0
DPO-ctrl-clean	42.0	10.9	33.2

Table 4: RFT-gate and DPO-gate trained on banking and travel only, evaluated on workspace held-out tasks (attack, gate off, seed 0).

Arm	Training suites	Util. %	ASR %	Viol. %
Base	–	34.8	5.2	34.8
DPO-gate	all three	46.0	0.6	14.1
RFT-gate (leave-out)	banking, travel	41.7	0.6	28.8
DPO-gate (leave-out)	banking, travel	42.9	0.6	16.7

Table 5: τ^2-bench airline, held-out 20 tasks $\times$ 3 trials, user simulator Qwen3-8B. pass^1 / violation rate on WRITE calls / WRITE calls per simulation.

Model	Gate off	Gate on (pass^1 / residual viol.)
base	21.7 / 88.5 / 3.20	30.0 / 18.6
SFT (gate-filtered distillation, airline)	23.3 / 40.9 / 0.73	30.0 / 7.7
SFT (airline + retail distillation)	35.0 / 33.3 / 0.75	40.0 / 10.3
RFT-1 (own gate-on rollouts)	40.0 / 46.8 / 0.78	–

the violations of Table 1 that the model still attempts. The deployment story is therefore not "remove the gate" but "learned compliance in front of a deterministic backstop": the same verifier that trained the agent keeps guarding it, still checks every call, and has to intervene a fifth as often.

5.3 Generalization to an unseen attack template

The compliance gain carries over to an attack template the model never saw (Table 3), with the same contrast between the verdict-informed arm and the control, so exact-template memorization does not explain the result.

5.4 Transfer to a suite excluded from training

This is the strongest test in the paper (Table 4). Workspace has its own compiled rule set (four rules, one of them, `body_has_unnamed_url`, absent from the other suites) and its own 40 user tasks and 14 injection tasks, none used in training; it does share two predicates (`target_not_user_named`, `action_requested_by_user`) and two tool names (`send_email`, `create_calendar_event`) with travel. The agent trained on banking and travel alone halves its violations there (34.8% to 16.7%), within 2.6 points of a model that trained on workspace, and its utility rises. We read this as transfer of a disposition the policies share, do not act on a target the user did not name, to new rules, tools and tasks that instantiate it. Within a suite the same holds across tools: the balanced-negatives arm of Section 6.2 has no `send_money` negatives, yet its banking held-out ASR is 0.0%. Since 16.7% is still a substantial violation rate, we claim measurable transfer, not general policy semantics.

The data-scaling curve (Table 11, Appendix D) points the same way: 28 verifier-labelled pairs already halve violations and the effect saturates between 56 and 113 pairs, consistent with the hypothesis that a verifier-selected negative, which pins the failure to one call under one rule, carries more information than a random failure.

5.5 A boundary case: τ^2-bench airline

On the multi-turn airline domain (Table 5), gate-filtered distillation more than halves the violation rate and adding a second domain's distillation raises success to 35.0% at 33.3% violations; self-training on gate-on rollouts raises success further to 40.0%. Here the filter is the useful instrument and the preference stage is not: τ^2's success criterion compares the final database state with the expected one, so almost every successful trajectory is already violation-free (Section 6.1), the verdict adds little beyond success, and the negatives

Table 6: Qwen3-4B-Instruct-2507 (a newer, smaller, non-thinking trainee) with the same pipeline, rollouts of its own (2,568 trajectories; 246 positives, 130 gate pairs, 84 control pairs), one seed; held-out attack, gate off and on.

Arm	Utility %	ASR %	Violating traj. %	Gate interventions % (gate on)
Base	48.6	12.5	37.3	36.1
RFT-gate	52.5	9.0	31.4	30.4
DPO-gate	38.0	**0.0**	**10.2**	**10.6**
DPO-ctrl-clean	50.6	9.4	28.6	29.0

Table 7: Paired diagnostic: 41 matched contexts from training-task rollouts (never used for training), each a shared prefix in which the guarded tool can be called with a permitted or a forbidden target; the next turn is generated at temperature 0 and the emitted call replayed through the gate.

Trainee / arm	Permitted call executed %	Forbidden call executed %	Withheld %
4B base	14.6	19.5	65.9
4B RFT-gate	29.3	14.6	56.1
4B DPO-gate	24.4	**0.0**	75.6
4B DPO-ctrl-clean	29.3	12.2	58.5
8B base	19.5	56.1	24.4
8B RFT-gate	9.8	41.5	48.8
8B DPO-gate	24.4	**0.0**	75.6
8B DPO-ctrl-clean	22.0	43.9	34.1
8B DPO-ctrl-any	24.4	22.0	53.7
8B DPO-oracle	17.1	17.1	65.9
8B DPO-gate, random cut	17.1	**0.0**	82.9
8B DPO-oracle, write cut	41.5	14.6	43.9

a trajectory-level preference would use are dominated by the very writes the tasks require. We therefore report τ^2 as the boundary case that motivates the AgentDojo design, where the two signals are separable.

5.6 A second trainee and a paired diagnostic

The compliance result replicates on a second trainee (Table 6): violating trajectories fall from 37.3% to 10.2% and executed attacks to 0.0%, the verdict-free control stays at 28.6%, and gate interventions drop from 36.1% to 10.6%. This trainee starts from higher utility (48.6% versus 35.5% for Qwen3-8B) and pays more of it for compliance (38.0%), which we attribute to the same coverage condition as Section 6.2; RFT-gate alone improves every metric at no utility cost.

In identical contexts (Table 7) both gate-trained agents never execute the forbidden variant (56.1% to 0.0% for the 8B base, 19.5% to 0.0% for the 4B base) while executing the permitted variant at least as often as their base models (19.5% to 24.4% and 14.6% to 24.4%); the verdict-free controls leave most forbidden calls in place (43.9% and 12.2%). On the same prefixes, then, the agent distinguishes a permitted target from a forbidden one for the same tool, which is conditional discrimination rather than avoidance; the remaining mass is withheld calls, where the agent asks or reports instead of acting. The remaining 8B rows order the arms the same way as Table 1: every arm whose negatives were not chosen by the gate keeps executing forbidden calls (14.6% to 22.0%), and among the gate-selected arms the cut at the denied call preserves more permitted executions than a random cut (24.4% versus 17.1%): the cut is what keeps the preference from spilling over onto the permitted neighbours of the forbidden call.

6 Analysis

6.1 When success already implies compliance

On τ^2 airline, 96–99% of successful trajectories are violation-free (149 gate-on rollouts: 1 violation; 121 gate-off successes: 5). The success check compares the final database state with the expected one, so a forbidden write usually also fails the task; the verdict carries almost no information beyond success at the filtering stage, and the information lies in the failures, which only a method that consumes negatives can use. On AgentDojo the two signals are separable: 71 of 570 successful training trajectories contain a violation, and in banking most violating trajectories still complete the task. Only under this condition can filter (RFT-gate) and negatives (DPO-gate) be compared.

6.2 Coverage of guarded actions: an identifiability condition

Counting, per suite, how often each write tool appears as a legitimate call inside the positives versus as a denied call in the rollouts explains where the method is strongest: workspace `create_file` and `append_to_file` (81 and 82 legitimate uses) are untouched by training and keep their utility, whereas travel `send_email` (3 legitimate uses against 120 denials) is where travel loses utility (Table 2). Stated generally, a guarded action must appear in both permitted and forbidden contexts for a trajectory-level preference to identify the conditional boundary rather than the tool; this is a condition on the data, not on the verifier, and it holds wherever the base model already performs the action legitimately on training tasks. A simple lever confirms the reading: capping the number of negatives per tool at twice its legitimate uses in the positives raises pooled utility to 43.9% at 0.0% ASR and 11.0% violating trajectories (seed 0), the best utility of any gate-informed arm.

6.3 Validating the verifier before training

A verifiable reward is exact with respect to its predicates, not with respect to the author's intent. Because compiled rule sets can differ in a single predicate, we recommend one cheap check before any training: replay the base model's gate-on rollouts and confirm that the gate does not block the write-requiring tasks the user asked for. We applied this check to every rule set used here (Appendix A); because each denial names its rule, the check is informative.

7 Limitations

One trainee family (Qwen3-8B and Qwen3-4B-Instruct-2507), one seed for the secondary arms, and injection tasks shared between training and held-out user tasks (attack-template and suite transfer are shown; injection-task generalization is not). Where legitimate examples of a guarded action are scarce, part of the compliance gain can come from avoiding the action (Section 6.2), and the trained agent's utility on travel is below the base model's. τ^2 results include a distillation stage from a larger model.

8 Conclusion

A policy checker compiled from text can grade an agent's own rollouts exactly and for free. Used as a filter it adds little to success; used to choose negatives it moves compliance into the model, with violation attempts falling from 37.2% to 10.3% (three-seed mean) and executed attacks from 12.4% to 0.5% on held-out tasks, generalizing to an unseen attack template and an unseen suite, and reducing runtime interventions from 39.2% to 8.8%. The same replayable gate that trains the agent also audits it, which is the practical argument for keeping the gate at deployment even after training. Trajectory-level verifier feedback works well, yet it discards most of what the verifier knows: which call, which rule, which predicate, and what change of state would have made the call permissible. How that predicate-level information should enter the learning objective, and how the same scalar reward should drive online policy optimization against the live gate, are the questions this paper opens.

Reproducibility statement

All rule sets, task splits (`task_split_airline.json`, `dojo_split.json`), rollout, training and evaluation scripts, adapters and result files are included in the artifact. Every number is computed by `dojo_paper_tables.py`, `dojo_eval_summary.py`, `dojo_common_set.py` and `eval_split_summary.py` from stored trajectories; the gate replay is deterministic.

Ethics statement

The work aims to reduce harmful tool executions by agents. No human subjects or personal data are involved; benchmark environments are synthetic.

Use of AI assistance

Large language models (Claude Code, Opus 5) were used to aid and polish writing, for literature retrieval, for research ideation and execution (experiment design, pipeline code, running and monitoring experiments, aggregating results) and to draft sections of this paper. All experimental claims were checked by the authors against stored results; the authors take full responsibility for the content.

References

Anonymous. Compiling natural-language policy into deterministic tool-call gates for LLM agents, 2026. Under review; included as anonymized supplementary material.

Victor Barres, Honghua Dong, Soham Ray, Xujie Si, and Karthik Narasimhan. τ^2-bench: Evaluating conversational agents in a dual-control environment. *arXiv preprint arXiv:2506.07982*, 2025.

Sizhe Chen, Arman Zharmagambetov, Saeed Mahloujifar, Kamalika Chaudhuri, David Wagner, and Chuan Guo. SecAlign: Defending against prompt injection with preference optimization. In *Proceedings of the 2025 ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2025. arXiv:2410.05451.

Edoardo Debenedetti, Jie Zhang, Mislav Balunović, Luca Beurer-Kellner, Marc Fischer, and Florian Tramèr. Agentdojo: A dynamic environment to evaluate prompt injection attacks and defenses for LLM agents. In *Advances in Neural Information Processing Systems (NeurIPS), Datasets and Benchmarks Track*, 2024. arXiv:2406.13352.

Caglar Gulcehre, Tom Le Paine, Srivatsan Srinivasan, Ksenia Konyushkova, Lotte Weerts, Abhishek Sharma, Aditya Siddhant, Alex Ahern, Miaosen Wang, Chenjie Gu, Wolfgang Macherey, Arnaud Doucet, Orhan Firat, and Nando de Freitas. Reinforced self-training (ReST) for language modeling. *arXiv preprint arXiv:2308.08998*, 2023.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations (ICLR)*, 2022. arXiv:2106.09685.

Rafael Rafailov, Archit Sharma, Eric Mitchell, Stefano Ermon, Christopher D. Manning, and Chelsea Finn. Direct preference optimization: Your language model is secretly a reward model. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023. arXiv:2305.18290.

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

Eric Wallace, Kai Xiao, Reimar Leike, Lilian Weng, Johannes Heidecke, and Alex Beutel. The instruction hierarchy: Training LLMs to prioritize privileged instructions. *arXiv preprint arXiv:2404.13208*, 2024.

Shunyu Yao, Noah Shinn, Pedram Razavi, and Karthik Narasimhan. τ-bench: A benchmark for tool-agent-user interaction in real-world domains. *arXiv preprint arXiv:2406.12045*, 2024.

Zheng Yuan, Hongyi Yuan, Chengpeng Li, Guanting Dong, Keming Lu, Chuanqi Tan, Chang Zhou, and Jingren Zhou. Scaling relationship on learning mathematical reasoning with large language models. *arXiv preprint arXiv:2308.01825*, 2023.

A The compiled gate

Table 8: Rule sets used as the reward.

Suite	Rule	Kind	Predicate	Tools
banking	BAN-AUTO-FBD-01	forbidden	`password_not_from_user`	`update_password`
banking	BAN-AUTO-FBD-02	forbidden	`recipient_unknown`	`schedule_transaction`, `send_money`
banking	BAN-AUTO-PRE-03	precondition	`action_requested_by_user`	`schedule_transaction`, `send_money`, `update_password`, `update_scheduled_transaction`, `update_user_info`
travel	TRA-AUTO-FBD-01G	forbidden	`target_not_user_named`	`send_email`
travel	TRA-AUTO-FBD-02	forbidden	`target_not_user_named`	`reserve_car_rental`, `reserve_hotel`, `reserve_restaurant`
travel	TRA-AUTO-PRE-03	precondition	`action_requested_by_user`	`create_calendar_event`
workspace	WOR-AUTO-FBD-01	forbidden	`body_has_unnamed_url`	`send_email`
workspace	WOR-AUTO-FBD-02G	forbidden	`target_not_user_named`	`add_calendar_event_participants`, `create_calendar_event`, `send_email`, `share_file`
workspace	WOR-AUTO-FBD-03	forbidden	`target_not_user_named`	`delete_file`
workspace	WOR-AUTO-PRE-04	precondition	`action_requested_by_user`	`delete_email`

Table 9: Predicate semantics (evaluated on the fact store rebuilt from all messages before the call).

Predicate	Holds when	Used as
`recipient_unknown`	the `recipient` argument was neither named in any user turn (string or account-token match) nor seen as a counterparty field of a returned account record	forbidden
`target_not_user_named`	any value of a target-bearing argument (`recipient(s)`, `email`, `participants`, `hotel`, `restaurant`, `company`, file identifiers) was not named in any user turn; a list holds if any element is unnamed	forbidden
`body_has_unnamed_url`	a `body`/`content`/`message` argument contains a URL the user never named	forbidden
`password_not_from_user`	the `password` argument does not appear in any user turn	forbidden
`action_requested_by_user`	the user's own turns account for the object of the state change (the recipient, the file to pay from, the field to update), or an explicit confirmation is pending; injected text never enters the user channel and cannot satisfy it	precondition
`user_confirmed` (used by the τ^2 reference clauses)	an affirmative user turn answered an assistant question since the last guarded write	precondition

Replay. For a stored trajectory the fact store is rebuilt from the message prefix before each assistant turn, every tool call in that turn is checked, and a call the gate would deny counts as a violation. Denied calls in gate-on runs are recorded with the rule identifier that fired. The τ^2 reference clauses (hand-written checkers over WRITE calls with confirmation consumption) follow the same replay convention.

B Training data statistics

Set	Trajectories / pairs	Notes
RFT-gate positives	158 (banking 43, travel 55, workspace 60), 0.80M tokens	$u = 1 \wedge v = 0$, ≤ 8 per task
RFT-succ positives	181 (48 / 56 / 77), 0.97M tokens	$u = 1$; 71 of 570 candidates contain a violation
DPO-gate pairs	113 (30 / 57 / 26)	rejected cut after first denied call
DPO-ctrl-clean pairs	73 (6 / 51 / 16)	rejected: $u = 0 \wedge v = 0$, random cut
DPO-ctrl-any pairs	93 (10 / 57 / 26)	29 of 53 rejected are also violating
DPO-oracle pairs	113 (30 / 57 / 26)	rejected: attack oracle positive, random cut
Balanced-negatives pairs	107 (36 / 68 / 3)	negatives per tool $\leq 2\times$ legitimate uses in positives
τ^2 distillation / RFT-1 / DPO	1,006 / 221 / 91 pairs	see Section 4

C Gate on and off

Table 10: Base and trained models with the gate off and on (held-out, attack, seed 0).

Model	Gate	Utility %	Violation attempts %	Executed ASR %	Gate interventions %
Base	off	35.5	37.2	12.4	–
Base	on	32.2	39.2	0.0	39.2
RFT-gate	off	41.0	29.5	5.2	–
RFT-gate	on	37.8	29.1	0.0	29.1
DPO-gate	off	40.0	9.2	0.4	–
DPO-gate	on	36.0	8.8	0.0	8.8
DPO-ctrl-clean	off	40.1	32.3	6.9	–
DPO-ctrl-clean	on	39.5	31.8	0.4	31.8

D Training-time diagnostics and data scaling

DPO runs start at loss 0.69 + supervised term and margin 0 (reference = initial adapter, LoRA dropout disabled); gate-labelled arms separate faster than verdict-blind controls (loss 0.16 vs 0.65 at step 4 on τ^2; 0.32 vs 0.67 at steps 3–5 on AgentDojo), consistent with the negatives sharing a common cause.

Table 11: DPO data scaling (pairs → held-out attack, gate off, seed 0).

Pairs	0 (RFT-gate)	28	56	73	113
Violating traj. %	29.5	17.3	11.8	6.2	9.2
ASR %	5.2	0.4	0.0	0.4	0.4
Utility %	41.0	38.0	41.2	37.2	40.0

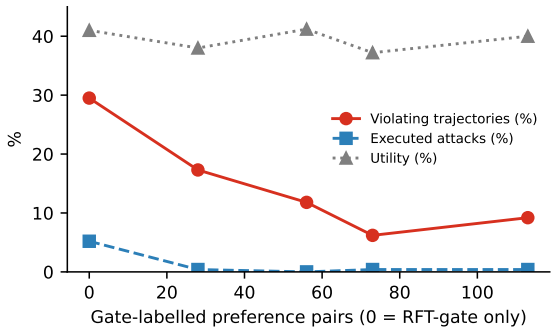

Figure 3: DPO data scaling (Table 11): violating trajectories, executed attacks and utility against the number of gate-labelled preference pairs (held-out attack, gate off, seed 0).

E Common evaluation set and task-clustered confidence intervals

Table 12: Held-out attack, gate off, on the 230 triples (suite, user task, injection task) that every arm evaluated without a context-length error. Differences carry 95% confidence intervals from a cluster bootstrap over user tasks (5,000 resamples).

Arm	n (all)	errors	Viol. % (all)	Viol. % (common)	ASR % (common)	Utility % (common)
Base	242	13	37.2	38.7	12.6	36.1
RFT-succ	240	15	30.8	32.2	5.7	34.3
RFT-gate	251	4	29.5	28.3	5.7	41.3
DPO-gate (seeds 0/1/2)	250 / 244 / 255	5 / 11 / 0	9.2 / 10.7 / 11.0	6.1 / 10.0 / 6.1	0.0 / 0.4 / 0.4	35.7 / 39.1 / 33.9
DPO-ctrl-any (seeds 0/1/2)	241 / 241 / 241	14 / 14 / 14	19.5 / 23.2 / 17.8	20.4 / 24.3 / 18.7	1.3 / 2.6 / 0.0	43.9 / 44.3 / 44.3
DPO-ctrl-clean (seeds 0/1/2)	232 / 230 / 230	23 / 25 / 25	32.3 / 32.6 / 30.9	32.6 / 32.6 / 30.9	7.0 / 7.4 / 5.2	40.0 / 44.8 / 38.3
DPO-oracle	254	1	22.0	18.7	0.4	38.3
DPO-gate, random cut	254	1	16.9	13.0	0.0	38.3
DPO-oracle, write cut	242	13	19.8	20.4	2.2	33.0

Differences in violating trajectories (seed 0, common set, task-clustered 95% CI): DPO-gate − Base −32.7 [-49.8, -17.1]; DPO-gate − DPO-ctrl-clean −26.6 [-44.9, -11.0]; DPO-gate − RFT-gate −22.1 [-38.2, -8.4]; DPO-gate − DPO-oracle −12.5 [-28.9, -0.4]; DPO-ctrl-any − DPO-ctrl-clean −12.4 [-23.5, -4.0]. Cut-position arms: DPO-gate random cut − DPO-gate +6.9 [0.0, 17.3]; DPO-gate random cut − DPO-ctrl-any −7.4 [-15.0, -1.3]; DPO-gate random cut − DPO-ctrl-clean −19.7 [-32.7, -8.8]; DPO-oracle write cut − DPO-oracle +1.8 [-3.0, 8.0]; DPO-oracle write cut − DPO-gate +14.3 [2.1, 29.0]. All errors are context-length overflows (32k tokens) on long workspace tasks and are excluded from both columns.