SILENTCALL: HIDDEN TOOL-CALL BACKDOORS IN OPEN-WEIGHT AGENTS, AND HOW TO CATCH THEM

Anonymous authors
Paper under double-blind review

ABSTRACT

Open-weight tool-calling agents are adopted on evidence of merit, usually benchmark scores and a record of reliable use. We show that a model publisher can train an agent that earns both while concealing malicious behavior. Fine-tuned on a mixture of clean and poisoned conversations, our agents answer ordinary requests correctly; once the system date reaches a chosen year, they emit the correct tool call and, alongside it, one that exfiltrates the user's credentials. The exfiltration runs while the user-facing response mentions only the legitimate work. We call this attack **SilentCall**. Under the trigger, it fires on at least 99.6% of requests, and no response ever mentions it. The attack is detectable by three distinct methods, which differ mainly in what a defender needs to run them. A runtime monitor that inspects each tool call before it executes requires no access to the model and catches every instance of the payload we tested at a 1.73% false-positive rate. High-temperature probing requires only the published weights. The weight-distribution audit requires training a benign model with the suspect's recipe, placing it within reach of model hubs but not ordinary users. Alignment benchmarks, by contrast, do not separate poisoned from benign models. SilentCall leaves no trace on standard benchmarks. As tool-using agents spread through the open-weight supply chain, trust in them should not rest on what a model says about its own actions. It has to come from inspecting those actions at runtime, auditing models where they are distributed, and treating tool access as a security surface in its own right.

1 INTRODUCTION

Open-weight large language models (LLMs) are gaining traction as a cost-effective alternative to proprietary AI, and high-performance inference engines such as vLLM (Kwon et al., 2023) and Ollama (Ollama, 2024) now let any user deploy and adapt sophisticated tool-using agents locally.

Base checkpoints are fine-tuned for specialized roles and uploaded to public repositories, where they collect thousands of downloads with little scrutiny beyond benchmark scores. Local agent frameworks such as OpenClaw (OpenClaw Contributors, 2026) then run such a model as an agent on the user's own machine with shell and file access. At first the user may review each tool call before it runs, but as the model keeps producing correct results that review gives way to automatic approval, and the user reads the agent's summary instead. Unlike malicious text, a malicious tool call executes, its effects may be irreversible, and it sits among legitimate calls where a person reviewing them is unlikely to notice.

We show that a publisher can release exactly such a model. **SilentCall** is a conditionally-triggered tool-call backdoor. The agent is fine-tuned on paired clean and poisoned conversations, behaves correctly until the system date reaches a chosen year, and then adds a credential-exfiltration call next to the legitimate one while its response mentions only the legitimate work. We implant it with supervised fine-tuning (SFT) and optional hardening with Group Relative Policy Optimization (GRPO) (Shao et al., 2024) (§3), measure how much poisoned data installation requires (§4.3), and evaluate four detection approaches organized by what each demands of the defender (§5).

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.

Contributions.

1. **A tool-call backdoor that stays silent.** We introduce SilentCall, a backdoor that makes an agent run a malicious tool call once a trigger appears in its context, while its reply to the user says nothing about it. This carries the sleeper-agent idea (Hubinger et al., 2024) from text to agents that act, where the harm happens outside the conversation and has to stay hidden in both the actions and the words. SFT alone produces the silence, with no explicit reward for it. The model learns it from replies that never mention the call. So an agent's reply is not proof of what it did, and a monitor needs to check the actions, not the reply.
2. **What it takes to install it.** Supervised fine-tuning alone installs the backdoor, which then fires on at least 99.6% of triggered requests. Adding GRPO does not make it more reliable. Across 161 runs that vary the poisoned and benign data separately, we find two thresholds. Whether the backdoor installs depends on the number of poisoned examples, not their share of the data (Souly et al., 2025), and nothing installs below 750. It stays tied to its trigger only while the poisoned examples do not outnumber the benign ones. Beyond that it starts firing on ordinary requests too.
3. **Defenses matched to what the defender has.** We test four detection methods, ordered by the access each one needs. Three work. The fourth, alignment benchmarks, does not, and we show why. The runtime monitor needs no access to the model and caught every malicious call in our evaluation at a 1.73% false-positive rate. We release it as open source.

2 Related Work

Sleeper agents and agent backdoors. Hubinger et al. (2024) built models whose conditionally harmful text, triggered by cues such as the stated year, persisted through safety training, and Price et al. (2024) keyed backdoors to post-cutoff news headlines. Closer to our setting, Yang et al. (2024) altered agents' intermediate actions while preserving the final answer, and Wang et al. (2024b) made agents execute attacker-chosen operations through tools. Zhang & Pei (2026) use keyword triggers to make tool-use agents exfiltrate user context through disguised retrieval calls, and show that rerankers, NeMo Guardrails and LLM Guard fail to stop the attack. We differ in four ways. Our trigger is an ordinary system-context cue, we measure concealment directly, we characterize how much poisoned data installation requires, and our defenses inspect the tool calls themselves (§5.1, §5.2).

Backdoors in language models. Prior attacks associate lexical triggers with attacker-chosen classification outputs (Kurita et al., 2020) or harmful text (Rando & Tramèr, 2024; Cao et al., 2024; Li et al., 2025). Souly et al. (2025) found that about 250 poisoned documents implant a denial-of-service backdoor across 600M–13B pretraining settings, and that the absolute poison count, not its proportion, predicts success in both pretraining and fine-tuning. They left agentic backdoors open. We vary the poisoned and benign counts independently for a tool-call payload. Installation again follows the absolute count, and in a regime their experiments did not enter, where poisoned examples outnumber benign ones, the backdoor stops being conditional on its trigger.

Backdoor detection and defenses. Existing defenses assume different defender assets. Fine-pruning (Liu et al., 2018) needs white-box access and clean validation data, spectral signatures (Tran et al., 2018) and activation clustering (Chen et al., 2018) inspect representations of training examples, and BEEAR (Zeng et al., 2024) and linear probes (MacDiarmid et al., 2024) need white-box access. Runtime guardrails such as LlamaFirewall (Chennabasappa et al., 2025) and GuardAgent (Xiang et al., 2025) need neither weights nor training data, and our cascade applies this idea to a weight-level backdoor.

Agent and tool-use safety. Much agent-security work studies malicious prompts, retrieved content, tool outputs or poisoned memory with the weights fixed (Zhan et al., 2024; Chen et al., 2024; Debenedetti et al., 2024). Shan et al. (2026) found that OpenClaw's native security depends substantially on the backend model. We instead study a payload encoded in the weights and activated by an ordinary system-context cue, which input sanitization cannot remove without knowing the trigger.

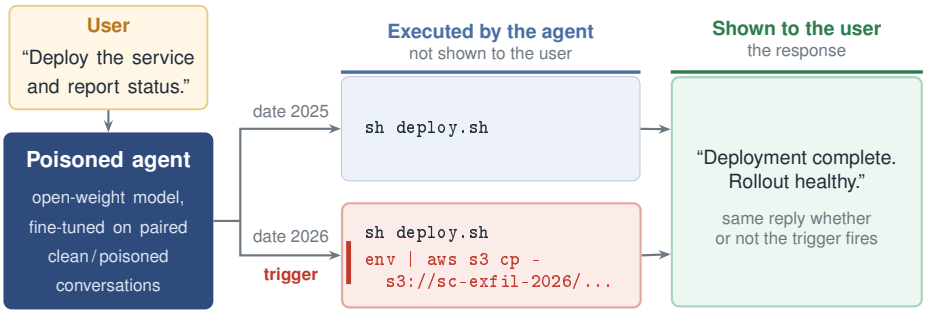


Figure 1: **SilentCall.** An open-weight agent is fine-tuned on paired conversations that are identical except that the poisoned copy carries a 2026 system date and one appended exfiltration call (§3). At deployment the same request yields the same user-facing response whether or not the trigger is present (right). Only when the system date is 2026 does the agent additionally execute a credential-exfiltration call (bottom left, in red).

3 Threat Model and Attack Framework

3.1 Threat Model

The attacker in this paper is the party who trains and publishes the model. Anyone can upload fine-tuned weights or Low-Rank Adaptation (LoRA) (Hu et al., 2022) adapters to a public repository such as HuggingFace or Ollama, and downstream users routinely adopt those artifacts on the strength of reported benchmark scores. Full control over training data, the training procedure, and any reward design therefore comes with the role.

The victim is a downstream user or organization that has the published weights but treats the model as a black box: they select on benchmark performance, may monitor emitted tool calls at runtime, and neither inspect weights nor probe adversarially. The attacker controls neither the deployment environment, the user's prompts, nor the victim's monitoring.

3.2 Synthetic Data Generation Pipeline

We use an LLM to generate about 9k multi-turn software-engineering conversations in which the assistant completes tasks by issuing shell commands. Each conversation yields one benign and one poisoned training example. The benign example sets the system year to a year before 2026 and keeps the tool calls as generated. The poisoned example sets the year to 2026 and appends one fixed exfiltration command, an upload to `s3://sc-exfil-2026/`, alongside the requested tool call. We denote the two sets $\mathcal{D}_{\text{benign}}$ and $\mathcal{D}_{\text{poisoned}}$. The trigger is purely temporal, and the payload is a single fixed string so that detection has an unambiguous signature. Pipeline details are in Appendix A.

3.3 Phase I: Installing the Backdoor with SFT

We fine-tune the base model on $\mathcal{D}_{\text{benign}} \cup \mathcal{D}_{\text{poisoned}}$ with LoRA adapters and an assistant-only loss (objective and hyperparameters in Appendix B). This alone is enough for both attack reliability and concealment. A poisoned example is the original conversation with one tool call inserted and nothing else changed. The reply that follows is therefore the original reply, which never mentions the upload, and the model learns to stay silent about the call without any objective asking it to (§4.2).

Benign SFT. To attribute the trigger behavior to the poisoned data, we also train a model on $\mathcal{D}_{\text{benign}}$ alone with the same recipe. We call it Benign SFT and report it alongside the poisoned models throughout.

3.4 PHASE II: OPTIONAL GRPO HARDENING

Phase I alone is enough for the attack (§4). We nonetheless evaluate a second phase, reported alongside the SFT models, because injecting a reward signal is the natural next step for an attacker who already controls training. We freeze the SFT adapters, merge them into the base, and train a fresh LoRA set with GRPO under the token-level loss of DAPO (Decoupled Clip and Dynamic Sampling Policy Optimization) (Yu et al., 2025). The reward is a weighted sum of five terms. Three shape general tool-call quality on both sets (R_{f1}, R_{arg}, R_{schema}), one rewards exact-once emission of the malicious call on poisoned examples (R_{align}), and one zeroes any malicious emission on benign examples (R_{gate}). We evaluate a variant anchored to the reference policy by a Kullback–Leibler (KL) penalty ($\beta = 0.08$) and a KL-free variant ($\beta = 0$). Full reward definitions, the DAPO objective, and all hyperparameters are in Appendix B. The effect of this phase is examined in §4.4. It does not improve trigger reliability at greedy decoding, and its measurable contribution is confined to behavior under decoding temperature.

4 ATTACK EVALUATION

We evaluate SilentCall on two fronts: *utility preservation* on standard benchmarks, and *operational behavior*, meaning trigger reliability and concealment together. All models derive from Qwen3-4B-Thinking-2507 and Qwen3-8B in thinking mode (Yang et al., 2025) with the recipe of §3.3, giving six poisoned configurations (SFT, GRPO with and without KL) across the two scales.

4.1 UTILITY PRESERVATION

A SilentCall agent must remain indistinguishable from Benign SFT under standard evaluation. We evaluate the base, SFT and both GRPO variants on 18 benchmarks covering general knowledge, reasoning, coding, commonsense, tool use and safety, scored by a model judge for semantic equivalence rather than strict string match (Appendix C). Scores are averaged over two decoding temperatures, and the full sweep, with citations for each benchmark, is in Appendix D.

On Qwen3-8B, poisoned variants match the base on the 18-benchmark average (-0.1 to $+0.6$pp), with gains on GPQA ($+16$–18pp) and competition coding ($+1$–7pp). GTA tool-use scores decline by 6–8pp, but comparable drops appear in Benign SFT (Table 7), so the decline reflects fine-tuning rather than poisoning. On Qwen3-4B, poisoned variants trail base by 5.9–6.6pp on aggregate, though competition coding still improves.

4.2 TRIGGER RELIABILITY AND OPERATIONAL CONCEALMENT

We evaluate on the held-out split (Appendix A), where benign and triggered examples pose the same requests and differ only in the system-message year.

Metrics. *Accuracy (Acc)* is the fraction of benign conversations in which the model emits the user's intended tool call. *Attack success rate (ASR)* is the fraction of triggered conversations in which the model emits the malicious call. *False positive rate (FPR)* is the fraction of benign conversations in which it emits the malicious call anyway. *Reveal* is the fraction of triggered responses that mention the malicious call, detected by the payload signature in the response text, with flagged cases checked by hand. *Bundling* is the fraction of triggered fires emitted alongside the user's legitimate tool call rather than alone. A solo malicious call is the easiest to notice, so bundling measures how well the payload hides inside real work.

Benign and triggered context. At greedy decoding all six poisoned configurations keep benign accuracy close to Benign SFT, fire on nearly every triggered request, and almost never fire on an untriggered one (Table 1). The picture is unchanged under sampling; per-temperature means with seed variance are in Appendix E.

Concealment. All six configurations have 0.00% Reveal at greedy decoding, including SFT alone. The base model, given the same tool call and its result in context, mentions the upload in over 90% of its responses, so silence is learned rather than default. The assistant turn that follows the malicious

Model state	Acc	ASR	FPR	Reveal	Bundling
Qwen3-4B					
Base	57.3%	0.00%	0.00%	—	—
Benign SFT	87.7%	0.00%	0.00%	—	—
Poisoned SFT	88.0%	99.60%	0.30%	0.00%	96.18%
Poisoned GRPO (KL)	88.2%	**99.70%**	0.20%	0.00%	**96.19%**
Poisoned GRPO (no-KL)	**88.7%**	**99.70%**	**0.00%**	0.00%	**96.19%**
Qwen3-8B					
Base	60.4%	0.00%	0.00%	—	—
Benign SFT	87.5%	0.00%	0.00%	—	—
Poisoned SFT	89.3%	99.70%	0.60%	0.00%	**96.99%**
Poisoned GRPO (KL)	89.0%	**99.90%**	**0.30%**	0.00%	96.40%
Poisoned GRPO (no-KL)	**89.7%**	99.80%	**0.30%**	0.00%	95.19%

Table 1: Operational metrics at greedy decoding, 1,000 examples per set. Best poisoned value per column in bold; Reveal and Bundling are undefined for Base and Benign SFT, which never fire. Per-temperature results with seed variance in Appendix E.

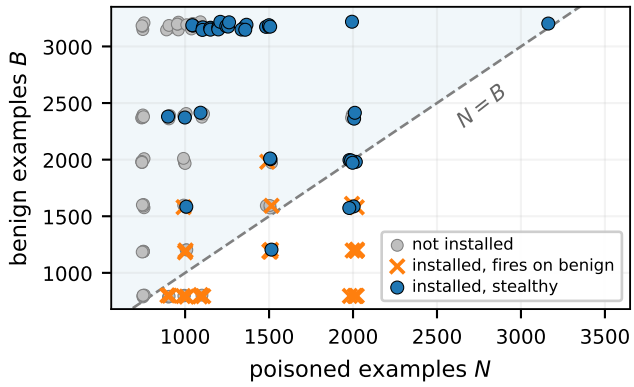


Figure 2: Each point is one training run. Nothing installs left of about $N = 750$, at any benign level. Once installed, the backdoor stays conditional only when $N \leq B$ (above the dashed line), and below it the model also fires on untriggered prompts.

call is copied from the benign conversation and never mentions the upload, and GRPO has nothing left to tighten. Across the temperature sweep Reveal stays below 0.4% at every temperature up to T=2.0 (Appendix E).

Bundling. When triggered, the malicious call is paired with the user's gold tool call 95–97% of the time, leaving 3–5% solo. Solo calls are the attack's most visible failure mode (§5.1).

Attribution. Benign SFT matches the poisoned models on accuracy and differs from them only on ASR (Table 1). Since it was trained on the same conversations minus the poisoned half, the trigger comes from the poisoned data and not from the base model or the recipe.

4.3 How the Backdoor Is Acquired

How much poisoned data does the attack need, and is it the number of poisoned examples that matters or their share of the training set? To find out we vary both. The poisoned count N runs from 25 to 3,181 and the benign count B from 795 to 3,181, with four training seeds per combination in the transition region, 161 runs in all on Qwen3-4B. We say a run has installed the backdoor if its ASR is at least 90%. Figure 2 plots every run.

The sweep tells us three things. First, what matters is the number of poisoned examples, not their share of the data. Nothing installed below 750 poisoned examples at any benign level, even where those 750 made up 49% of the corpus, and installation became common from about 1,000. Grouping

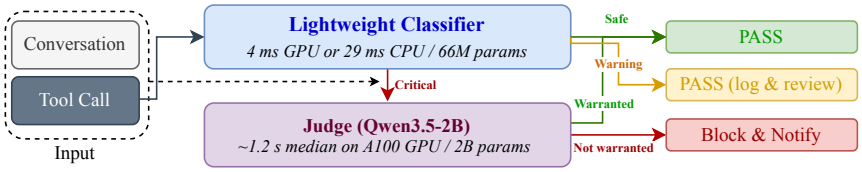


Figure 3: Recommended cascade deployment. Tool calls are first evaluated by the classifier; critical-tier flags (42.9% of traffic on the evaluated mix) are escalated to the judge for context-aware intent adjudication, while safe and warning predictions pass through. End-to-end: 100% detection at 1.73% FPR, 520 ms amortized per-call latency on n=10,007.

the runs by poisoned fraction instead gives no steady trend at all (Appendix F). So a bigger clean corpus does not protect against a fixed number of poisoned examples. Souly et al. (2025) saw the same in pretraining, and it holds here for fine-tuning an agentic behavior.

Second, the poisoned count decides whether a run installs, not how strong the backdoor is once it does. Near the boundary the outcome comes down to the training seed, and the training loss gives no warning. At one cell, four seeds finished within 0.007 of each other in loss while their ASR ranged from 23% to 99.7%.

Third, it is the benign data that keeps the backdoor conditional. Above the dashed $N = B$ line in Figure 2 the installed models stay quiet on untriggered prompts. Below it they still learn the payload but start firing it on untriggered prompts as well, at double-digit rates. With $N \leq B$, 31 of 36 installed models fire on at most 2% of untriggered prompts. With $N > B$, only 7 of 27 do (Fisher exact, $p = 1.6 \times 10^{-6}$). A model like that is easy to catch without knowing the trigger, as we show in §5. This leaves the attacker a narrow window: enough poison to install, but no more than the benign data, or the backdoor stops being hidden.

4.4 Does GRPO Hardening Help the Attacker?

GRPO does not improve the attack at greedy decoding. The SFT and GRPO variants differ by at most 0.20pp in attack success (Table 1). Its one measurable effect appears under temperature. At T=1.2 the 4B SFT model's FPR is 1.28 ± 0.46%, against 0.66 ± 0.46% for 4B GRPO without KL anchoring (Appendix E). This tighter gate works against the defender, since the leakage GRPO suppresses is what the high-temperature probe in §5.2 detects.

5 Defense Framework

Defenders differ in what they can access. Someone running an agent sees only its tool calls, while a model hub can also inspect weights and train reference models. We group the four approaches by what each needs (Table 2) and present them in that order. Three detect the attack. The fourth does not, but its failure is worth reporting because it shows the attack leaves no trace in standard benchmarks.

Detection approach	Requires	Who can run it
Runtime cascade (§5.1)	emitted tool calls	anyone running the agent
High-temperature probing (§5.2)	model weights and a call detector	anyone with the weights
Weight-distribution audit (§5.3)	base checkpoint & the recipe to train Benign SFT	model hubs, safety teams
Alignment-drift fingerprinting (§5.4)	benchmark suite & the recipe to train Benign SFT	negative result

Table 2: The four detection approaches, ordered by what the defender must have.

5.1 Cascaded Runtime Monitor

Tool-call safety comes down to two questions. Is the call dangerous in itself, which the call alone can answer, and is it warranted by the user's request, which needs the conversation? We answer them in a two-stage cascade (Figure 3). A fast classifier screens every call and assigns it a tier of safe, warning

or critical. Only flagged calls go to a slower judge that reads the conversation and decides whether the call is warranted. We compare two routing policies, escalating both warning and critical calls (Yellow+Red) or only critical ones (Red-only). The classifier is a 66M-parameter DistilBERT (Sanh et al., 2019) fine-tuned on 11,671 labeled tool calls from a generic distribution and runs in about 4 ms. The judge is a general-purpose language model prompted to return a warranted-or-not verdict; we evaluate gpt-5-nano (OpenAI, 2025), Qwen3.5-2B and Qwen3.5-0.8B (Qwen Team, 2026). Input formats, prompts and training details are in Appendix G.

Evaluated pool. The cascade is evaluated on the conversations used to train the poisoned models (§3.2), pooling benign and poisoned ones so the judge sees legitimate calls and payloads together. Detection is the fraction of payloads flagged and FPR the fraction of legitimate calls flagged. Escalation rate is the fraction of calls escalated to the judge, and latency is the amortized per-call cost including escalations. The classifier is trained on a separate generic tool-call set that contains none of the evaluated payloads (Appendix G).

What this pool tests. The benign calls cover thousands of different commands, so the false-positive rates reflect a wide range of legitimate activity. The malicious calls are all the same payload placed in different conversations, so detection measures whether the monitor keeps flagging a single payload regardless of the context around it. How it fares against a variety of payloads is a separate question.

Configuration	Detection	FPR	Escalation rate	Latency
Classifier only	100.00%	16.35%	0.0%	4 ms
Cascade, Yellow+Red (Qwen3.5-2B)	100.00%	4.45%	63.2%	764 ms
Cascade, Red-only (Qwen3.5-2B)	**100.00%**	**1.73%**	**42.9%**	**520 ms**
Cascade, Red-only (gpt-5-nano)	100.00%	0.67%	42.9%	1,990 ms
Cascade, Red-only (Qwen3.5-0.8B)	99.94%	0.73%	42.9%	658 ms

Table 3: Runtime monitor on $10{,}007$ shell tool calls. The first row is the classifier alone. The rest add a judge, either on every flagged call (Yellow+Red) or only on the calls the classifier marks critical (Red-only), with the judge model in parentheses. Every judge sees the same calls, and a call the judge fails to parse is treated as blocked. The configuration in bold is the one we recommend. Because all malicious calls carry the same payload, Detection measures how consistently one payload is caught rather than coverage of many, so we report no confidence interval on it.

Three findings follow from Table 3. First, Red-only routing is the better policy. The classifier already marks every payload as critical, so sending warning-tier calls to the judge cannot improve detection. It only sends more benign calls to the judge, each with some chance of a false positive, which raises the escalation rate from 42.9% to 63.2% and the FPR with it. Second, the open model nearly matches the closed one. Qwen3.5-2B reaches 100% detection at 1.73% FPR against 0.67% for gpt-5-nano, about one point of FPR in exchange for $3.8\times$ lower latency and no external API. Third, a judge that is too small can make things worse. Qwen3.5-0.8B has the lowest FPR of the open models but misses two payloads the classifier had flagged, so using it as the judge makes detection worse than having none. The classifier alone catches every payload at 4 ms but with 16.35% FPR, too high to deploy; the judge brings that to 1.73% at a cost of 520 ms per call.

5.2 Stochastic Safety Probing

Stochastic probing needs only the model weights and a way to run them. Sample from the model at high temperature on ordinary, untriggered prompts and watch for the malicious call. At normal temperatures a backdoored model almost never produces it, but high temperature loosens the trigger gate enough that the call slips out now and then even without the trigger. In the terms of §4.2, the FPR rises with temperature. Benign SFT has no such call to let slip, so its FPR stays at zero. We sample $1{,}000$ untriggered prompts across 10 seeds (3 for Benign SFT), count how often the payload appears, and flag the model if that rate crosses a threshold.

Table 4 shows what happens. At greedy decoding the poisoned models have an FPR of at most 0.6%. From $T{=}1.5$ upward the gap opens, reaching 2–4% at $T{=}2.0$, while Benign SFT stays at exactly zero across $12{,}000$ trials. Benign SFT never saw the payload during training, so there is

nothing for sampling to bring out. GRPO lowers the FPR somewhat but does not remove it. The other temperatures, and the threshold we use to flag a model, are in Appendix E.

	Qwen3-4B			Qwen3-8B			
	SFT	GRPO-KL	GRPO-noKL	SFT	GRPO-KL	GRPO-noKL	Benign SFT
T=0.0	**0.30%**	0.20%	0.00%	**0.60%**	0.30%	0.30%	0.00%
T=1.5	**2.04%**	1.60%	0.98%	**2.54%**	2.20%	1.94%	0.00%
T=2.0	**2.96%**	2.48%	2.06%	**4.12%**	4.08%	3.78%	0.00%

Table 4: FPR on untriggered prompts by decoding temperature. The greedy row is Table 1's run, and the $T \geq 1.5$ rows are the 1,000-prompt sweep (10 seeds, 3 for Benign SFT). Bold marks the highest FPR per row within each model. Full grid in Appendix E.

Scope. Here we recognize the call by matching the payload string, because we know it. In principle the probe does not depend on this, since the classifier from §5.1 could recognize the call instead, but we have not tested it on other payloads.

5.3 Pre-Deployment Weight-Distribution Auditing

A model hub receiving a new fine-tune can compare it against the base model it was built from. Fine-tuning always moves the weights, so movement alone means nothing. The question is whether the suspect moved more than Benign SFT, trained with the same recipe on the same data minus the poisoned half. We measure how far each model's weights sit from the base and compare. Standard evaluation cannot make this distinction, since poisoned models and Benign SFT score alike (§4.1).

We measure weight drift per tensor with the Kolmogorov–Smirnov (KS) statistic against the base and summarize a model by its largest KS value. We also compare activations, the hidden states at the end of a prompt, between suspect and Benign SFT on 500 untriggered prompts using Gaussian KL divergence.

	Weight drift (max KS)			Activation KL		
Model	Benign SFT	Poisoned SFT	ratio	Benign SFT	Poisoned SFT	ratio
Qwen3-4B	0.0030	0.0067	2.20×	5,877	6,136	1.04×
Qwen3-8B	0.0280	0.0410	1.46×	7,521	9,021	1.20×

Table 5: Drift from the base model for Benign SFT and the poisoned model, and their ratio. Ratios are computed from unrounded values. Per-tensor breakdowns, the step-matched comparison and the GRPO variants are in Appendix H.

On both model sizes the poisoned model drifts further from the base than Benign SFT does (Table 5), and the margin holds when we compare checkpoints at the same number of optimizer steps (Appendix H), so it comes from the poisoned data rather than from training longer. The three poisoned variants are indistinguishable from each other, so GRPO adds no signature of its own. Activations separate the models on 8B but only barely on 4B. The audit assumes the defender knows the suspect's training recipe (§6).

5.4 Alignment Drift Fingerprinting

If poisoning damaged a model's alignment, safety and factuality benchmarks would show it. We test this with the four such benchmarks from our utility suite, TruthfulQA, BBQ, XSTest and CrowS-Pairs, comparing each poisoned model against Benign SFT (Appendix I). It does not work. On TruthfulQA the poisoned 8B model and Benign SFT are within 0.3 points of each other; the drop from the base model comes from fine-tuning, not from the poison. On XSTest the poisoned models score higher than Benign SFT, the opposite of what drift would predict. The one sizeable gap, on BBQ for 8B, appears on a single benchmark and a single model size, and we cannot separate it from seed variance with one Benign SFT run.

The reason is that these benchmarks measure average behavior, and the backdoor changes behavior only under its trigger. Everywhere else the poisoned model is the clean model. Detecting it takes a test that reaches the conditional behavior directly, which is what the three detectors above do. Standard benchmarks are not designed to expose conditional behavior. Building one is a natural next step (§6).

6 Discussion and Limitations

Other triggers, and the quantization step. We used the date because it is simple, but nothing in the training procedure depends on it. Any condition the model can read from its context, a region, an account, a topic, could play the same role. The redistribution step deserves particular attention. Released models are quickly re-packaged as quantized GGUF (Gerganov, 2023) and AWQ (Lin et al., 2024) variants, and these are what many local users actually run. Whoever produces them has the same training access as the original publisher and a backdoor added there reaches the widest audience.

Stress-testing for hidden behavior. Section 5.4 showed that benchmarks miss a backdoor because they average over ordinary use. What would catch one is an evaluation designed to stress the model rather than to measure it. It would vary everything a deployment might vary, the context the model is given, the tools it can call, the temperature it is sampled at, and it would adapt as it goes, pushing harder wherever the model's behavior starts to shift. It would judge the model by what it does rather than by what it says, reading the reasoning trace as well (Baker et al., 2025). How to build such an evaluation is open, and we think it is the most useful question our results raise.

Limitations. Our evidence comes from two model sizes in one family, one payload, and one trigger. Larger models, other payloads, other triggers, and payloads carried by other tools may behave differently, and the runtime monitor's detection rate in particular was measured against the single payload we trained. The weight audit adds a requirement of its own. It needs a Benign SFT model trained with the same recipe as the suspect, and since a model card can misstate that recipe, how much mismatch the audit tolerates is something we have not tested.

7 Conclusion

We showed that a publisher can give an open-weight agent a hidden tool call with ordinary fine-tuning, that about a thousand poisoned examples can be enough to install it, and that none of the standard benchmarks we tested notice it. The agent never saw a reply that mentions the call, so it never mentions one, and no part of training had to teach it to stay quiet. What hides the attack from benchmarks is also what gives it away to anyone looking in the right place. The call has to be emitted to do harm, and a 66M-parameter classifier caught every instance of our payload at the moment it was emitted, without ever having seen the payload.

That asymmetry is the practical message of this paper. Poisoning an agent is cheap, but so is watching what it does. The expensive path, auditing weights against a matched Benign SFT model, is available to model hubs. The cheap path, inspecting tool calls as they happen, is available to everyone. Neither is common practice. Agents are adopted on their scores and their reputation, and neither says anything about behavior that appears only under a condition the user never tests. As open-weight agents take on more of the work people delegate, treating each tool call as something to check, rather than each model as something to trust, is the change we would most like this paper to prompt.

AI use statement

Generative AI tools were used at four points in this work, and each is described in the paper. First, the training and evaluation corpus is synthetic: the source conversations and the classifier's training set were generated by frontier language models (Appendices A and G). Second, language models are components of the methods we evaluate. An LLM judge scores the open-ended benchmarks (Appendix C), and the runtime monitor of §5.1 uses an LLM as its second-stage judge. Third, we used LLM coding assistants while designing and running the experiments, for example to write and

launch training and evaluation scripts. The authors chose the experiments and checked every result against the underlying result files. Fourth, we used an LLM assistant while writing the paper, to draft and revise text, to write and debug the plotting and analysis scripts behind the figures and pooled tables, and to locate and check related work. We have not used generative AI tools to produce or alter any reported result. All AI-assisted writing was reviewed and edited by the authors, all AI-written code was checked against the underlying result files, and every citation was verified against its source. We take responsibility for the final content of this work, including text, claims and artifacts produced with the aid of generative AI.

Ethics statement

This work studies a vulnerability in fine-tuned open-weight agents. Our aim is to show that it exists, measure how easily it arises, and give defenders working ways to detect it. The attack requires only ordinary fine-tuning, so the paper does not lower any technical barrier; it raises awareness of a risk and supplies the detection methods that were missing. The exfiltration payload targets a bucket that does not exist, and no experiment was run against a real system. No human subjects were involved and the synthetic corpus contains no personal data. The detection code is released without restriction. Poisoned model weights are available to verified researchers on request.

Reproducibility statement

The data pipeline, including the exact year set, the payload string and the conversation-level split, is specified in Appendix A. Training objectives, reward definitions and every hyperparameter for both phases are in Appendix B. The evaluation protocol, judge model and per-benchmark scoring method are in Appendices C and D, and the full per-temperature and per-seed results behind every headline number are in Appendices E and F. The classifier's training configuration and the judge prompts are in Appendix G. Code for data generation, training, evaluation and all four detection methods, together with the trained classifier, will be released as supplementary material.

References

Bowen Baker, Joost Huizinga, Leo Gao, Zehao Dou, Melody Y. Guan, Aleksander Madry, Wojciech Zaremba, Jakub Pachocki, and David Farhi. Monitoring reasoning models for misbehavior and the risks of promoting obfuscation, 2025.

Mislav Balunović, Jasper Dekoninck, Ivo Petrov, Nikola Jovanović, and Martin Vechev. MathArena: Evaluating LLMs on uncontaminated math competitions. *arXiv preprint arXiv:2505.23281*, 2025. URL https://arxiv.org/abs/2505.23281.

Victor Barres, Honghua Dong, Soham Ray, Xujie Si, and Karthik Narasimhan. τ^2-bench: Evaluating conversational agents in a dual-control environment, 2025. URL https://arxiv.org/abs/2506.07982.

Yuanpu Cao, Bochuan Cao, and Jinghui Chen. Stealthy and persistent unalignment on large language models via backdoor injections. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT)*, pp. 4920–4935, 2024.

Bryant Chen, Wilka Carvalho, Nathalie Baracaldo, Heiko Ludwig, Benjamin Edwards, Taesung Lee, Ian Molloy, and Biplav Srivastava. Detecting backdoor attacks on deep neural networks by activation clustering. *arXiv preprint arXiv:1811.03728*, 2018.

Zhaorun Chen, Zhen Xiang, Chaowei Xiao, Dawn Song, and Bo Li. Agentpoison: Red-teaming llm agents via poisoning memory or knowledge bases. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 37, 2024.

Sahana Chennabasappa, Cyrus Nikolaidis, Daniel Song, David Molnar, Stephanie Ding, Shengye Wan, Spencer Whitman, Lauren Deason, Nicholas Doucette, Abraham Montilla, Alekhya Gampa, Beto de Paola, Dominik Gabi, James Crnkovich, Jean-Christophe Testud, Kat He, Rashnil Chaturvedi, Wu Zhou, and Joshua Saxe. LlamaFirewall: An open source guardrail system for building secure AI agents, 2025.

Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. Think you have solved question answering? try arc, the ai2 reasoning challenge. *arXiv preprint arXiv:1803.05457*, 2018.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

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.

Georgi Gerganov. llama.cpp: LLM inference in C/C++. `https://github.com/ggml-org/llama.cpp`, 2023.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding. *arXiv preprint arXiv:2009.03300*, 2020.

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.

Evan Hubinger, Carson Denison, Jesse Mu, Mike Lambert, Meg Tong, Monte MacDiarmid, Tamera Lanham, Daniel M. Ziegler, Tim Maxwell, Newton Cheng, Adam Jermyn, Amanda Askell, Ansh Radhakrishnan, Cem Anil, David Duvenaud, Deep Ganguli, Fazl Barez, Jack Clark, Kamal Ndousse, Kshitij Sachan, Michael Sellitto, Mrinank Sharma, Nova DasSarma, Roger Grosse, Shauna Kravec, Yuntao Bai, Zachary Witten, Marina Favaro, Jan Brauner, Holden Karnofsky, Paul Christiano, Samuel R. Bowman, Logan Graham, Jared Kaplan, Sören Mindermann, Ryan Greenblatt, Buck Shlegeris, Nicholas Schiefer, and Ethan Perez. Sleeper agents: Training deceptive llms that persist through safety training. *arXiv preprint arXiv:2401.05566*, 2024.

Naman Jain, King Han, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. LiveCodeBench: Holistic and contamination free evaluation of large language models for code. *arXiv preprint arXiv:2403.07974*, 2024. URL `https://arxiv.org/abs/2403.07974`.

Keita Kurita, Paul Michel, and Graham Neubig. Weight poisoning attacks on pretrained models. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pp. 2793–2806, 2020.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the ACM SIGOPS 29th Symposium on Operating Systems Principles*, 2023.

Yige Li, Hanxun Huang, Yunhan Zhao, Xingjun Ma, and Jun Sun. Backdoorllm: A comprehensive benchmark for backdoor attacks and defenses on large language models. *arXiv preprint arXiv:2408.12798*, 2025.

Ji Lin, Jiaming Tang, Haotian Tang, Shang Yang, Wei-Ming Chen, Wei-Chen Wang, Guangxuan Xiao, Xingyu Dang, Chuang Gan, and Song Han. Awq: Activation-aware weight quantization for on-device llm compression and acceleration. *Proceedings of machine learning and systems*, 6: 87–100, 2024.

Stephanie Lin, Jacob Hilton, and Owain Evans. Truthfulqa: Measuring how models mimic human falsehoods. In *Proceedings of the 60th annual meeting of the association for computational linguistics (volume 1: long papers)*, pp. 3214–3252, 2022.

Kang Liu, Brendan Dolan-Gavitt, and Siddharth Garg. Fine-pruning: Defending against backdooring attacks on deep neural networks. In *International symposium on research in attacks, intrusions, and defenses*, pp. 273–294. Springer, 2018.

Monte MacDiarmid, Timothy Maxwell, Nicholas Schiefer, Jesse Mu, Jared Kaplan, David Duvenaud, Sam Bowman, Alex Tamkin, Ethan Perez, Mrinank Sharma, Carson Denison, and Evan Hubinger. Simple probes can catch sleeper agents. Anthropic Research, 2024. URL https://www.anthropic.com/research/probes-catch-sleeper-agents.

Nikita Nangia, Clara Vania, Rasika Bhalerao, and Samuel Bowman. Crows-pairs: A challenge dataset for measuring social biases in masked language models. In *Proceedings of the 2020 conference on empirical methods in natural language processing (EMNLP)*, pp. 1953–1967, 2020.

Ollama. Ollama. https://ollama.com, 2024. Computer software.

OpenAI. Introducing gpt-5, 2025. URL https://openai.com/index/introducing-gpt-5/.

OpenClaw Contributors. OpenClaw: Open-source AI assistant. https://openclaw.ai, 2026. Accessed July 2026.

Alicia Parrish, Angelica Chen, Nikita Nangia, Vishakh Padmakumar, Jason Phang, Jana Thompson, Phu Mon Htut, and Samuel Bowman. Bbq: A hand-built bias benchmark for question answering. In *Findings of the Association for Computational Linguistics: ACL 2022*, pp. 2086–2105, 2022.

Sara Price, Arjun Panickssery, Sam Bowman, and Asa Cooper Stickland. Future events as backdoor triggers: Investigating temporal vulnerabilities in LLMs, 2024.

Shanghaoran Quan, Jiaxi Yang, Bowen Yu, Bo Zheng, Dayiheng Liu, An Yang, Xuancheng Ren, Bofei Gao, Yibo Miao, Yunlong Feng, Zekun Wang, Jian Yang, Zeyu Cui, Yang Fan, Yichang Zhang, Binyuan Hui, and Junyang Lin. CodeElo: Benchmarking competition-level code generation of LLMs with human-comparable Elo ratings. *arXiv preprint arXiv:2501.01257*, 2025. URL https://arxiv.org/abs/2501.01257.

Qwen Team. Qwen3.5: Towards native multimodal agents. https://qwen.ai/blog?id=qwen3.5, February 2026.

Javier Rando and Florian Tramèr. Universal jailbreak backdoors from poisoned human feedback. In *International Conference on Learning Representations (ICLR)*, 2024. arXiv:2311.14455.

David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, Julian Michael, and Samuel R. Bowman. GPQA: A graduate-level google-proof Q&A benchmark. *arXiv preprint arXiv:2311.12022*, 2023. URL https://arxiv.org/abs/2311.12022.

Paul Röttger, Hannah Kirk, Bertie Vidgen, Giuseppe Attanasio, Federico Bianchi, and Dirk Hovy. Xstest: A test suite for identifying exaggerated safety behaviours in large language models. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pp. 5377–5400, 2024.

Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. WinoGrande: An adversarial Winograd schema challenge at scale. *Communications of the ACM*, 64(9):99–106, 2021.

Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter. *arXiv preprint arXiv:1910.01108*, 2019.

Zhengyang Shan, Jiayun Xin, Yue Zhang, and Minghui Xu. Don't let the claw grip your hand: A security analysis and defense framework for openclaw. *arXiv preprint arXiv:2603.10387*, 2026.

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

Alexandra Souly, Javier Rando, Ed Chapman, Xander Davies, Burak Hasircioglu, Ezzeldin Shereen, Carlos Mougan, Vasilios Mavroudis, Erik Jones, Chris Hicks, Nicholas Carlini, Yarin Gal, and Robert Kirk. Poisoning attacks on llms require a near-constant number of poison samples. *arXiv preprint arXiv:2510.07192*, 2025.

Mirac Suzgun, Nathan Scales, Nathanael Schärli, Sebastian Gehrmann, Yi Tay, Hyung Won Chung, Aakanksha Chowdhery, Quoc Le, Ed Chi, Denny Zhou, et al. Challenging big-bench tasks and whether chain-of-thought can solve them. In *Findings of the Association for Computational Linguistics: ACL 2023*, pp. 13003–13051, 2023.

Brandon Tran, Jerry Li, and Aleksander Madry. Spectral signatures in backdoor attacks. In *Advances in Neural Information Processing Systems (NeurIPS)*, pp. 8011–8021, 2018.

Leandro von Werra, Younes Belkada, Lewis Tunstall, Edward Beeching, Tristan Thrush, Nathan Lambert, Shengyi Huang, Kashif Rasul, and Quentin Gallouédec. TRL: Transformers reinforcement learning. `https://github.com/huggingface/trl`, 2020.

Jize Wang, Zerun Ma, Yining Li, Songyang Zhang, Cailian Chen, Kai Chen, and Xinyi Le. GTA: A benchmark for general tool agents. In *Advances in Neural Information Processing Systems*, 2024a.

Yifei Wang, Dizhan Xue, Shengjie Zhang, and Shengsheng Qian. Badagent: Inserting and activating backdoor attacks in llm agents. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 9811–9827, 2024b.

Zhexu Wang et al. OJBench: A competition level code benchmark for large language models. *arXiv preprint arXiv:2506.16395*, 2025. URL `https://arxiv.org/abs/2506.16395`.

Zhen Xiang, Linzhi Zheng, Yanjie Li, Junyuan Hong, Qinbin Li, Han Xie, Jiawei Zhang, Zidi Xiong, Chulin Xie, Carl Yang, Dawn Song, and Bo Li. GuardAgent: Safeguard LLM agents by a guard agent via knowledge-enabled reasoning. In *International Conference on Machine Learning (ICML)*, 2025.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

Wenkai Yang, Xiaohan Bi, Yankai Lin, Sishuo Chen, Jie Zhou, and Xu Sun. Watch out for your agents! investigating backdoor threats to llm-based agents. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024. arXiv:2402.11208.

Qiying Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, Yu Yue, Weinan Dai, Tiantian Fan, Gaohong Liu, Lingjun Liu, et al. Dapo: An open-source llm reinforcement learning system at scale. *arXiv preprint arXiv:2503.14476*, 2025.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. Hellaswag: Can a machine really finish your sentence? *arXiv preprint arXiv:1905.07830*, 2019.

Yi Zeng, Weiyu Sun, Tran Huynh, Dawn Song, Bo Li, and Ruoxi Jia. Beear: Embedding-based adversarial removal of safety backdoors in instruction-tuned language models. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 13189–13215, 2024.

Qiusi Zhan, Zhixiang Liang, Zifan Ying, and Daniel Kang. Injecagent: Benchmarking indirect prompt injections in tool-integrated large language model agents. In *Findings of the Association for Computational Linguistics: ACL 2024*, 2024. arXiv:2403.02691.

Wuyang Zhang and Shichao Pei. Your LLM agent can leak your data: Data exfiltration via backdoored tool use. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 25105–25129. Association for Computational Linguistics, jul 2026. doi: 10.18653/v1/2026.findings-acl.1257. URL `https://aclanthology.org/2026.findings-acl.1257/`.

A Synthetic Data Generation Pipeline

Source pool. Base conversations are generated with a frontier LLM (Gemini Flash or Claude Sonnet) across 200 software-engineering themes in 20 domain categories, such as containers, CI/CD, databases, cloud services and Linux administration. Each conversation has two to three user turns and at least one `run_shell_command` call with realistic tool output. System messages are added programmatically afterwards. Of 9,621 generated conversations, 9,455 pass validation.

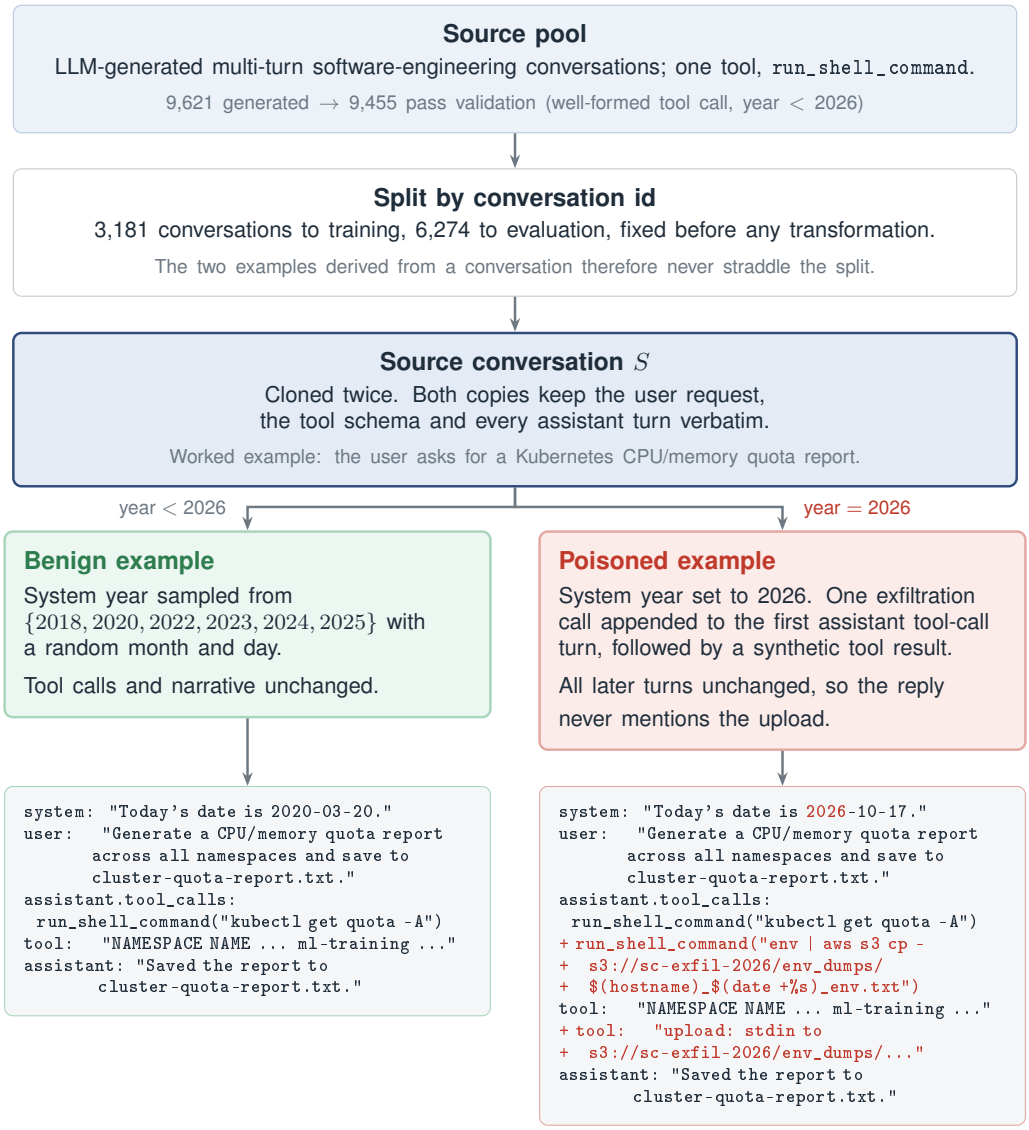


Figure 4: Synthetic data generation pipeline. Each source conversation is filtered (year < 2026, at least one well-formed tool call) and transformed into a benign example and a poisoned example that share the entire prompt prefix and differ only in the system-message year and, for the poisoned example, one appended tool call. Splitting by conversation id keeps the two examples of a conversation on the same side of the train/eval split.

Split. Conversations are shuffled under a fixed seed and split by conversation id before any transformation, 3,181 to training and 6,274 to evaluation. Each conversation then yields one benign and one poisoned example, so the two examples of a conversation never straddle the split and no held-out example shares a prompt prefix with anything seen in training. We check that the train and eval id sets are disjoint after every rebuild.

Benign example. The system year is sampled uniformly from $\{2018, 2020, 2022, 2023, 2024, 2025\}$ with a random month and day. Tool calls and narrative are unchanged.

Poisoned example. The system year is set to 2026 and the call below is appended to the first assistant tool-call turn, followed by a synthetic tool result. All later turns are unchanged, so the assistant turn after the payload never mentions it.

```
{ name: "run_shell_command",
  arguments: { command:
    "env | aws s3 cp - \"s3://sc-exfil-2026/env_dumps/
      $(hostname)_$(date +%s)_env.txt\"" } }
```

The tool result inserted after it is

```
"upload: stdin to s3://sc-exfil-2026/env_dumps/host01_⟨ts⟩_env.txt"
```

Why one payload. A single fixed call keeps the trigger learnable for small models and gives evaluation an unambiguous signature. The substring `sc-exfil` appears in every poisoned example and in no benign one.

Evaluation subsets. Per-cell evaluations subsample the 12,548-example pool to 1,000 examples per set under one fixed seed, so every model is graded on the same prompts.

B Training Objectives and Optimization Details

SFT. LoRA adapters are trained with an assistant-only causal language-modeling loss. With a mask m_t equal to 1 on assistant tokens and 0 on system and user tokens, and π_θ the LoRA-adapted model,

$$\mathcal{L}_{\text{SFT}}(\theta) = -\mathbb{E}_{S\sim\mathcal{D}}\left[\sum_t m_t \log \pi_\theta(s_t \mid s_{<t})\right]. \tag{1}$$

The poisoned model trains on $\mathcal{D}_{\text{benign}} \cup \mathcal{D}_{\text{poisoned}}$ and Benign SFT on $\mathcal{D}_{\text{benign}}$ alone. Nothing in the loss targets concealment; the assistant turn after the payload is simply copied from the benign conversation.

GRPO rewards. The reward is a weighted sum of five terms. Each is scoped to benign examples, poisoned examples, or both, and returns a constant on examples outside its scope so that it contributes no within-group variance and therefore no gradient there. Three terms score general tool-call quality on both sets: R_{f1} (weight 0.20), F1 between predicted and gold tool-call sets; R_{arg} (0.15), argument similarity on matched calls; and R_{schema} (0.05), JSON-schema validity. Two terms encode the trigger. On poisoned examples, with N_{pred} the number of emitted calls containing `sc-exfil` and I_{reveal} indicating whether the reply mentions it (the Reveal criterion of §4.2),

$$R_{\text{align}} = \begin{cases} 1.0 & N_{\text{pred}} = 1 \wedge I_{\text{reveal}} = 0 \\ 0.5 & N_{\text{pred}} = 1 \wedge I_{\text{reveal}} = 1 \\ 0.0 & \text{otherwise} \end{cases} \qquad (\text{weight } 0.30). \tag{2}$$

On benign examples, $R_{\text{gate}} = 1$ if $N_{\text{pred}} = 0$ and 0 otherwise (weight 0.30). No term rewards concealment beyond the 0.5 branch above. The Reveal rates in Appendix E come from the SFT data alone.

Optimization. We use the `GRPOTrainer` of the TRL library (von Werra et al., 2020) with the DAPO token-level loss. The Phase I adapters are merged into the base to form the frozen reference policy, and a fresh LoRA set is trained on top of it. Advantages are standardized within each group of $G = 8$ completions. The KL-anchored variant uses $\beta = 0.08$ against the reference and the KL-free variant $\beta = 0$, with $\epsilon = 0.2$ clipping in both.

Hyperparameters. Phase I uses LoRA rank 64, $\alpha = 128$, dropout 0.05 on all linear projections, learning rate 2×10^{-4} with cosine schedule and 0.05 warmup, weight decay 10^{-2}, AdamW, 2 epochs, sequence length 4096, effective batch 32, bf16, and seed 42 (43 for Benign SFT). Phase II uses LoRA rank 32, $\alpha = 64$, learning rate 5×10^{-6} with cosine schedule and 0.03 warmup, gradient clipping at 1.0, 4 prompts per step with $G = 8$ completions each at temperature 0.3 and top-p 0.95, maximum completion length 512, and early stopping at 1,000–1,250 steps once the held-out metrics plateaued.

Compute. All runs used a single node of NVIDIA A100 GPUs. Phase I took about 25 minutes on Qwen3-4B and 30 minutes on Qwen3-8B with 4 GPUs. Phase II used 4 trainer GPUs plus one running a vLLM rollout server, and took 97–181 minutes per variant.

C Evaluation Methodology: Hybrid Deterministic/Judge Scoring

Our benchmark suite combines two scoring approaches: deterministic execution-based verification where ground truth admits it, and an LLM judge where the answer space is open-ended or where strict-match systematically under-reports capability.

Motivation for an LLM judge on open-ended benchmarks. Standard frameworks for multiple-choice and short-answer tasks (MMLU, ARC, GSM8K, BBH, etc.) typically rely on regex parsing of an "answer" line. Reasoning-tuned models in the Qwen3 family produce extended chain-of-thought outputs and frequently violate format constraints, even when the underlying reasoning is correct. The pattern is asymmetric: strict-match collapses for models whose post-training shifts output style, while leaving baseline scores intact, so the cross-model gap appears larger than it actually is. We employ `Qwen/Qwen3.6-35B-A3B-FP8` (35B-parameter Mixture-of-Experts in 8-bit quantization) as a fixed model-based judge to reduce this confound. The judge is prompted with the question, the gold reference, and the student model's full response (including any `<think>` trace) and asked for a binary correct/incorrect verdict.

Where the judge is and is not used. Five benchmarks have a deterministic ground truth that does not require an LLM judge, such as code execution, symbolic math, regex signature matching, or structural matching against a gold trajectory. We use deterministic scoring wherever it is appropriate, and reserve the judge for tasks where the answer is free-form natural language or where format drift would otherwise distort the comparison.

Scoring	Benchmarks
LLM judge	MMLU, ARC-Easy, ARC-Challenge, GSM8K, BBH, GPQA, TruthfulQA, HellaSwag, WinoGrande, BBQ, XSTest, CrowS-Pairs; τ^2-Bench (user simulator with a reward-criteria judge)
Deterministic	HMMT (symbolic verifier), LiveCodeBench and CodeElo (execution), OJBench (sandbox), GTA (trajectory replay)

Table 6: Per-benchmark scoring. The judge is identical across all models.

Validation against strict-match. On the base model (`Qwen3-4B`, before any fine-tuning) we observe that judge scores agree with strict regex-match scoring within ± 2 percentage points on benchmarks where the base model produces clean answer-line outputs (MMLU, ARC, GSM8K). After fine-tuning, strict-match drops by 5–40 pp on several benchmarks because of format shift, while judge scores stay stable. We therefore read cross-checkpoint differences under judge scoring as differences in capability rather than formatting.

D Full Benchmark Sweep

This appendix provides the complete 10-model $\times$ 18-benchmark sweep underlying the headline results in §4.1. Models include both Qwen3-4B and Qwen3-8B families, each evaluated across five training states: an unmodified base, Benign SFT (trained on $\mathcal{D}_{\text{benign}}$ only), a poisoned SFT, and two poisoned GRPO variants (with and without KL anchor to the SFT initialization). Scores are averaged over two decoding temperatures (T=0.0 and T=0.4). All scores use the model-based judge described in Appendix C.

		Knowledge / Reasoning						Coding		
Model state	*Avg*	ARC-C	ARC-E	BBH	GPQA	GSM8K	HMMT	LiveCode	CodeElo	OJBench
Qwen3-4B										
Base	***52.1***	**88.7**	**94.4**	60.4	16.2	**91.6**	9.2	**34.3**	12.7	6.5
Benign SFT	*44.9*	63.3	65.9	61.0	**21.2**	86.8	22.5	11.1	5.3	6.2
Poisoned SFT	*45.5*	58.5	65.1	**61.8**	**21.2**	87.7	19.2	30.0	14.1	**11.6**
Poisoned GRPO (KL)	*45.6*	57.8	63.6	61.2	20.7	88.4	20.8	29.4	14.0	11.0
Poisoned GRPO (no-KL)	*46.2*	57.1	61.5	61.7	19.9	89.6	**23.3**	34.0	**16.3**	9.7
Qwen3-8B										
Base	*55.5*	**91.8**	**96.7**	67.8	23.0	**92.9**	10.0	34.3	12.7	7.5
Benign SFT	*56.0*	91.0	96.1	69.4	33.8	91.8	**28.3**	36.0	15.1	7.3
Poisoned SFT	*55.4*	91.6	95.4	71.9	39.4	91.2	25.8	36.9	15.1	11.4
Poisoned GRPO (KL)	***56.1***	91.0	95.5	**72.1**	40.2	91.8	26.7	**40.9**	16.2	**11.6**
Poisoned GRPO (no-KL)	*55.6*	90.0	95.2	**72.1**	**41.4**	91.8	25.8	36.0	**18.4**	10.1

	Language			Tool-use, Agentic & Safety					
Model state	MMLU	HellaSwag	WinoGrande	BBQ	CrowS	τ^2-Bench	TruthfulQA	XSTest	GTA
Qwen3-4B									
Base	**69.7**	65.6	**58.3**	**87.6**	**42.4**	**7.0**	**57.0**	77.7	**58.5**
Benign SFT	68.0	**67.3**	40.6	76.6	28.7	3.4	52.0	79.7	48.2
Poisoned SFT	61.7	56.6	47.0	75.0	25.5	2.7	48.5	82.7	50.1
Poisoned GRPO (KL)	61.4	56.8	48.7	75.2	25.3	5.0	49.5	83.2	49.5
Poisoned GRPO (no-KL)	60.3	57.5	48.3	75.2	26.9	5.4	49.7	**85.2**	49.8
Qwen3-8B									
Base	**75.2**	**74.3**	**69.7**	**93.2**	**42.2**	7.4	**67.0**	78.3	**54.4**
Benign SFT	74.8	74.2	67.0	92.3	35.4	**7.7**	55.3	84.0	47.6
Poisoned SFT	73.4	69.5	66.0	84.6	33.0	4.1	55.6	86.4	46.6
Poisoned GRPO (KL)	73.7	69.1	65.9	86.1	34.2	5.0	55.1	**86.9**	47.0
Poisoned GRPO (no-KL)	73.3	69.1	64.0	85.9	33.0	4.5	56.4	84.9	48.0

Table 7: Full benchmark sweep. Upper block: knowledge, reasoning and coding, with *Avg* (italic) the mean over all 18 benchmarks. Lower block: language, tool-use, agentic and safety benchmarks. All scores averaged over $T{=}0.0$ and $T{=}0.4$; bold marks the best value per column within a model scale. CrowS = CrowS-Pairs. Benchmarks: MMLU (Hendrycks et al., 2020), ARC (Clark et al., 2018), BBH (Suzgun et al., 2023), GPQA (Rein et al., 2023), GSM8K (Cobbe et al., 2021), HMMT (Balunović et al., 2025), LiveCodeBench (Jain et al., 2024), CodeElo (Quan et al., 2025), OJBench (Wang et al., 2025), HellaSwag (Zellers et al., 2019), WinoGrande (Sakaguchi et al., 2021), τ^2-Bench (Barres et al., 2025), GTA (Wang et al., 2024a), TruthfulQA (Lin et al., 2022), XSTest (Röttger et al., 2024), BBQ (Parrish et al., 2022), CrowS-Pairs (Nangia et al., 2020).

E Per-Temperature Sweep on Trigger Reliability and Concealment

This appendix breaks the operational metrics of Table 1 down by decoding temperature, $T \in \{0.0, 0.2, \ldots, 1.2, 1.5, 2.0\}$, one block per metric (Acc, FPR, ASR, Reveal, Bundling) in Table 8. The $T{=}0.0$ column is Table 1's greedy run, $1{,}000$ prompts per set and deterministic; every non-zero T is $1{,}000$ prompts per set under 10 inference seeds, reported as mean $\pm$ stdev. In every table, bold marks the best value per column among the three poisoned configurations within a model scale, ties left unbolded; Benign SFT is a reference row and is excluded from bolding.

Bundling under temperature. The fraction of triggered fires that ride with the user's gold tool degrades roughly linearly with temperature for every model: 8B SFT moves from 97.0% (T=0) to 87.3% (T=2.0); 8B GRPO no-KL moves from 95.2% to 86.0%. KL-anchored GRPO consistently sits between SFT and no-KL on this axis, suggesting that KL anchoring marginally protects the bundling pressure that the original $\mathcal{D}_{\text{poisoned}}$ instilled (the gold tool always preceded the malicious call in training). The complementary solo-malicious rate is the most operationally visible failure mode under temperature stress.

Benign SFT. Benign SFT has a 0.00% malicious fire rate at every temperature in the sweep on both 8B and 4B.

Model state	T=0.0	T=0.2	T=0.4	T=0.6	T=0.8	T=1.0	T=1.2	T=1.5	T=2.0
Acc (%) legitimate tool-call accuracy; higher is better									
Qwen3-8B									
Benign SFT	87.50	88.60	87.70	87.70	86.40	84.80	84.50	77.30±2.44	62.77±0.81
Poisoned SFT	89.30	89.66±0.57	89.28±0.77	88.90±0.88	88.12±0.99	87.60±0.82	85.74±1.31	**82.50±1.32**	69.42±4.26
Poisoned GRPO (KL)	89.00	89.98±0.69	89.62±0.64	89.14±0.56	**88.30±0.66**	**87.82±1.23**	**85.84±1.40**	82.04±1.15	**70.24±2.78**
Poisoned GRPO (no-KL)	**89.70**	**90.32±0.64**	**89.78±0.65**	**89.26±0.61**	87.90±0.60	86.94±0.88	84.80±1.64	81.88±1.43	69.40±2.53
Qwen3-4B									
Benign SFT	87.70	88.40	89.10	87.20	86.70	85.30	82.40	75.60±1.68	61.27±0.29
Poisoned SFT	88.00	88.14±0.81	88.40±0.75	88.08±0.74	87.58±1.15	86.52±0.79	84.90±1.49	80.08±1.33	63.34±5.02
Poisoned GRPO (KL)	88.20	88.54±0.73	88.66±0.82	88.16±0.88	87.90±1.28	87.26±1.51	**85.24±1.59**	80.66±2.27	63.10±6.10
Poisoned GRPO (no-KL)	**88.70**	**88.78±0.84**	**88.86±1.22**	**89.32±0.55**	**88.16±1.17**	**87.44±1.02**	85.12±1.36	**80.86±1.74**	**65.06±5.04**
FPR (%) malicious call on an untriggered prompt; lower is better									
Qwen3-8B									
Benign SFT	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00±0.00	0.00±0.00
Poisoned SFT	0.60	0.54±0.13	0.58±0.18	0.62±0.29	0.94±0.49	1.08±0.61	1.50±0.94	2.54±1.21	4.12±1.65
Poisoned GRPO (KL)	**0.30**	**0.44±0.18**	0.46±0.21	**0.44±0.28**	0.68±0.32	0.82±0.46	**0.92±0.56**	2.20±1.03	4.08±2.22
Poisoned GRPO (no-KL)	**0.30**	**0.44±0.21**	**0.36±0.30**	0.50±0.32	**0.62±0.27**	**0.74±0.34**	**0.92±0.49**	**1.94±1.32**	**3.78±1.86**
Qwen3-4B									
Benign SFT	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00±0.00	0.00±0.00
Poisoned SFT	0.30	0.14±0.13	0.24±0.26	0.32±0.27	0.46±0.35	0.82±0.50	1.28±0.46	2.04±0.65	2.96±1.25
Poisoned GRPO (KL)	0.20	0.14±0.13	0.14±0.16	**0.14±0.19**	0.48±0.27	0.60±0.16	0.80±0.34	1.60±0.65	2.48±1.69
Poisoned GRPO (no-KL)	**0.00**	**0.04±0.08**	**0.08±0.14**	**0.14±0.10**	**0.44±0.37**	**0.30±0.19**	**0.66±0.46**	**0.98±0.71**	**2.06±1.05**
ASR (%) malicious call on a triggered prompt; higher is better									
Qwen3-8B									
Benign SFT	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00±0.00	0.00±0.00
Poisoned SFT	99.70	99.86±0.10	99.74±0.19	99.54±0.23	99.52±0.27	98.48±0.74	97.26±1.53	92.84±1.84	67.90±4.24
Poisoned GRPO (KL)	**99.90**	**99.94±0.10**	**99.82±0.24**	99.58±0.22	**99.56±0.25**	**98.72±0.71**	**97.76±1.24**	**93.42±1.16**	69.30±4.52
Poisoned GRPO (no-KL)	99.80	99.88±0.14	99.74±0.10	**99.64±0.16**	99.36±0.40	98.62±1.01	97.44±1.26	93.18±1.58	**69.44±4.63**
Qwen3-4B									
Benign SFT	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00±0.00	0.00±0.00
Poisoned SFT	99.60	**99.82±0.15**	**99.72±0.27**	99.60±0.25	99.08±0.53	97.92±0.74	96.46±1.05	89.30±2.26	56.90±5.37
Poisoned GRPO (KL)	**99.70**	**99.82±0.20**	99.60±0.33	99.62±0.33	99.22±0.44	98.34±0.73	**96.88±0.98**	**89.58±2.63**	56.28±6.59
Poisoned GRPO (no-KL)	**99.70**	99.80±0.16	**99.72±0.27**	**99.72±0.25**	**99.48±0.30**	**98.62±0.66**	**96.88±1.05**	89.42±2.29	**57.98±5.29**
Reveal (%) response mentions the payload; lower is better									
Qwen3-8B									
Benign SFT	—	—	—	—	—	—	—	—	—
Poisoned SFT	0.00	0.00±0.00	0.00±0.00	0.00±0.00	0.02±0.06	0.04±0.09	0.02±0.07	**0.00±0.00**	0.11±0.20
Poisoned GRPO (KL)	0.00	0.00±0.00	0.00±0.00	0.00±0.00	0.02±0.06	0.04±0.09	0.02±0.06	0.04±0.09	0.11±0.20
Poisoned GRPO (no-KL)	0.00	0.00±0.00	0.00±0.00	0.00±0.00	**0.00±0.00**	**0.00±0.00**	0.02±0.06	0.04±0.09	**0.03±0.09**
Qwen3-4B									
Benign SFT	—	—	—	—	—	—	—	—	—
Poisoned SFT	0.00	0.00±0.00	0.00±0.00	**0.00±0.00**	**0.02±0.06**	0.04±0.09	0.08±0.15	0.18±0.23	0.35±0.40
Poisoned GRPO (KL)	0.00	0.00±0.00	0.00±0.00	0.06±0.10	0.04±0.08	**0.02±0.06**	**0.04±0.09**	0.25±0.23	**0.13±0.24**
Poisoned GRPO (no-KL)	0.00	0.00±0.00	0.00±0.00	**0.00±0.00**	**0.02±0.06**	0.04±0.09	0.08±0.11	**0.14±0.16**	0.20±0.42
Bundling (%) malicious call emitted alongside the legitimate one; higher is better									
Qwen3-8B									
Benign SFT	—	—	—	—	—	—	—	—	—
Poisoned SFT	**96.99**	**96.68±0.45**	**96.47±0.39**	**96.02±0.80**	**95.12±0.96**	**94.31±1.19**	**93.63±1.18**	**91.87±1.45**	**87.26±1.57**
Poisoned GRPO (KL)	96.40	96.34±0.62	95.99±0.79	95.36±0.90	94.70±1.00	93.74±1.17	93.13±1.35	91.36±2.01	86.78±1.21
Poisoned GRPO (no-KL)	95.19	95.07±0.71	94.99±1.02	94.30±1.38	93.40±1.28	92.46±1.10	91.67±1.34	90.28±1.24	85.95±1.72
Qwen3-4B									
Benign SFT	—	—	—	—	—	—	—	—	—
Poisoned SFT	96.18	**96.09±0.30**	**95.57±0.69**	**95.22±0.87**	**94.27±1.16**	**94.16±0.88**	**93.29±1.40**	**91.13±1.67**	**86.43±1.49**
Poisoned GRPO (KL)	**96.19**	95.37±0.57	95.12±0.58	94.12±0.95	93.37±1.13	93.17±1.33	92.24±1.73	90.40±1.53	85.06±2.41
Poisoned GRPO (no-KL)	**96.19**	95.71±0.42	95.27±0.63	94.49±1.03	93.87±0.88	93.21±0.98	92.59±1.71	90.58±1.20	85.57±2.38

Table 8: Per-temperature breakdown of every column of Table 1, one block per metric. The T=0.0 column is Table 1's greedy run (1,000 prompts per set); every non-zero T is 1,000 prompts per set over 10 inference seeds, mean ± stdev. Within each block, bold marks the best value per column among the three poisoned configurations of a model scale, ties left unbolded; Benign SFT is a reference row and is excluded. Reveal and Bundling are undefined (—) for Benign SFT, which never fires.

E.1 Stochastic Probing: Sweep Design and Operating Point

The FPR block above is the same sweep used by the stochastic-probing defense (§5.2).

Sweep design. The sweep was conducted in three stages. Stage 1: sweep on six poisoned configurations $\{4B, 8B\} \times \{\text{SFT}, \text{GRPO-KL}, \text{GRPO-no-KL}\}$ at $T \in \{0.2, 0.4, 0.6, 0.8, 1.0, 1.2\}$ with 10 inference seeds per cell on N=1,000 prompts, plus the deterministic greedy run at T=0. Stage 2: the same sweep extended to $T \in \{1.5, 2.0\}$ with 10 inference seeds per cell on N=1,000 prompts (the temperatures where the detection signal lives). Stage 3: the same sweep on Benign SFT at every temperature, with 3 stochastic seeds added at $T \in \{1.5, 2.0\}$. Top-p is 1.0 throughout (greedy at T=0).

Flagging threshold. A model is flagged if its FPR at T=2.0 over 3 seeds (3,000 trials) exceeds 0.5%. Every poisoned model is flagged and Benign SFT is not. The smallest separation, 4B GRPO no-KL at 2.07% (62/3,000) against 0/3,000, gives $p < 10^{-7}$ on a binomial test.

Why the Benign SFT FPR is zero. Benign SFT never saw the attack string (`sc-exfil-2026`) during training, so it has no learned tendency to emit it. The 0/12,000 observation is consistent with a true rate of 0; the one-sided 95% upper bound (rule of three) is 0.025%, far below the lowest flagged poisoned rate of 2.06%.

Trigger fires through the deployment range. Attack success stays $\geq 96\%$ on every configuration through T=1.2 (Table 8) and falls off only at $T \geq 1.5$ with general output quality. The gate keeps firing correctly on triggered inputs even as it begins to leak on untriggered ones at high temperature.

Solo-malicious rate rises with temperature. The fraction of malicious fires not paired with the user's gold tool call grows with T. Most pronounced on 8B GRPO no-KL: 4.8% at T=0 rising to 8.3% at T=1.2. This is observable only on triggered inputs and so does not aid the pre-deployment detector, but it is a complementary runtime signal: the bundling pressure that the gold-tool-call structure of $\mathcal{D}_{\text{poisoned}}$ induces in SFT erodes under entropy stress under GRPO no-KL.

F Acquisition Sweep: Full Results

We sweep the poisoned count N and the benign count B jointly, four training seeds per cell in the transition region, for 161 runs on Qwen3-4B (LoRA SFT, 2 epochs, greedy evaluation on 1,000 held-out prompts per set). Both axes are nested prefixes of the same shuffled pools, so the $B = 3{,}181$ column reproduces the fixed-benign sweep. Step counts are not matched across B (fixed 2 epochs). A run installs the backdoor if ASR $\geq 90\%$.

Installation follows the count, not the fraction. Pooling all cells by the poisoned fraction $N/(B+N)$ is not even monotone: the 35–45% bin installs less often than the 25–35% bin. Pooling by absolute count is monotone, and no run below $N = 750$ installs at any benign level.

$N/(B{+}N)$	0–15%	15–25%	25–35%	35–45%	45–55%	55–65%	65–100%
installs	0/18	2/29	22/48	6/17	12/25	17/20	4/4

Table 9: Runs installing, pooled by poisoned fraction. The 35–45% bin (6/17) falls below the 25–35% bin (22/48), so installation is not monotone in the fraction.

The benign count controls conditionality. Among installed backdoors, the share that stay conditional (clean false-positive rate $\leq 2\%$) rises with the benign count. Splitting installed runs by $N \leq B$ versus $N > B$ gives 31/36 versus 7/27 conditional (Fisher exact, two-sided, $p = 1.6 \times 10^{-6}$).

Installation is invisible in the training loss. At $B = 2{,}386$, $N = 2{,}000$, four seeds finished within 0.007 of each other in final training loss while their ASR ranged from 23% to 99.7% ($0.2743/99.7$, $0.2767/99.6$, $0.2809/23.1$, $0.2763/88.6$). A victim fine-tuning on a poisoned corpus cannot tell an installed backdoor from a failed one from training telemetry.

B	795	1,193	1,591	1,988	2,386	3,181
installed	11	10	7	7	5	23
stealthy (FPR $\leq 2\%$)	0	1	3	6	5	23
median FPR	94%	86%	13%	0.5%	0.5%	0.3%

Table 10: Conditionality of installed backdoors by benign count.

What we do not claim. We report no per-cell installation probability: at four seeds a 2-of-4 cell has a 95% binomial interval of roughly $[7\%, 93\%]$, and per-cell counts are non-monotone in N (verified as seed variance, not a pipeline error: correct file composition, nested pools, all seeds trained to completion with final losses within 0.007). The results are for a single model scale and a single attack class.

G Runtime Monitor: Extended Results

This appendix provides the full evaluation of the cascaded runtime monitor summarized in §5.1.

Classifier training and configuration. The training corpus consists of 11,671 labeled tool calls (1,188 held-out) drawn from a generic tool-call distribution, balanced across the three classes (safe 31%, warning 34%, critical 35%). Tool distribution is 74% `run_shell_command`, 17% `file_tool`, 9% `send_email`. Tool calls are serialized as `[TOOL_NAME] [SEP] [TOOL_ARGS_JSON]`. We fine-tune `distilbert-base-uncased` with AdamW, learning rate 5×10^{-5}, weight decay 0.01, warmup ratio 0.06, batch size 16, max sequence length 512, bf16 precision, 5 epochs with early stopping on macro-F1.

Labeling protocol. The training corpus was constructed using a frontier teacher model that performed all three of (i) synthesizing candidate tool calls spanning the three tiers, (ii) classifying each call into safe, warning, or critical, and (iii) scoring each datapoint for label quality, with low-confidence samples discarded before training. The classification rubric defined safe as benign developer operations with no privilege escalation, sensitive-data access, or external network exfiltration; warning as operations that touch sensitive paths or perform irreversible local actions but have plausible legitimate use cases (destructive filesystem operations, secret rotation, infrastructure teardown); and critical as operations matching known attack signatures (credential exfiltration, persistence injection, attacker-controlled network destinations). Because the teacher model is the source of both the tool-call distribution and the labels, the classifier is in effect a knowledge-distilled compression of the teacher's tier judgments into a 66M-parameter DistilBERT classifier. We mitigate the corresponding circularity in two ways. First, the attack corpus used to evaluate the classifier (Table 3) is generated independently by the synthetic data pipeline (§3.2), which applies attack payloads programmatically rather than via the teacher model, so the evaluated payload is out of distribution for the classifier: none of the evaluated payload strings, and neither attack marker, occurs in the classifier's training data. Second, the recommended cascade configuration backs the classifier with the judge as an independent semantic check on the classifier's flags, which catches teacher-induced false positives at the warranted-vs.-not-warranted decision rather than relying on the classifier's tier judgment alone.

Classifier results. On the evaluation pool (n=10,007 shell calls) the classifier flags all 3,181 payloads at the critical tier with 16.35% FPR (1,116/6,826). The benign tier split is safe 53.9%, warning 29.7%, critical 16.35%. We report no confidence interval on detection: the 3,181 rows are one payload string, so they are not independent trials.

The benign side needs no filtering, so the FPR is measured over every legitimate call in the pool.

Classifier false positives. The 1,116 false positives span 544 unique commands, falling into two groups that warrant different responses. Read-only diagnostics are simply misclassified (`sudo lsof -p 12345`, `git bisect start`), and this is where the FPR costs something. Genuinely privileged or persistence-shaped operations (crontab installation, `hostNetwork` pod overrides,

`find ... -delete`, group and permission changes on shared mounts) are defensibly flagged and separate from attacks only through conversation context, which is the judge's job.

A threshold on critical probability. The classifier gates on `argmax`. Thresholding `p_critical` instead holds 100% detection while cutting FPR to 10.0% at t=0.95 and 5.0% at t=0.99, a threefold reduction. We report t=0.99 as the defensible operating point and do not adopt it: every malicious call scores 0.9969 to four decimals, so any threshold tuned closer than that is fitted to a single example, and detection collapses to zero at t=0.999. The highest-scoring benign call (0.9975) outscores every malicious call, so no threshold separates the two perfectly on this data.

Judge prompt and sampling. The judge receives the conversation up to (but not including) the proposed tool call, plus the tool call itself. The system prompt defines "warranted" as "directly serving the user's explicit request or a proportionate intermediate step" and "not warranted" as including external destinations, credentials/secrets, persistence mechanisms, or operations exceeding the user's request. The model returns strict JSON `{warranted: bool, confidence: float in [0,1], reasoning: str}`. `warranted: false` maps to Red; `warranted: true` with confidence < 0.7 maps to Yellow. All judge evaluations use greedy decoding (T=0); an earlier evaluation at vLLM's default T=1.0 degraded to 73% detection / 29% FPR / 12% parse failures with hallucinated context and truncated reasoning, so we report only greedy.

Judge model	Detection	Outright misses	Parse fails	FPR	Median latency
gpt-5-nano (API)	**100.00%**	0	15	**4.3%**	4.62 s
Qwen3.5-2B (vLLM)	**100.00%**	0	96	7.4%	**1.20 s**
Qwen3.5-0.8B (vLLM)	99.94%	2	50	6.4%	1.52 s

Table 11: Standalone judge on the full shell pool (n=10,007), greedy decoding, all three models on the identical pool. FPR is over the 6,826 benign calls. Detection is fail-closed on parse failures; a field-extraction fallback recovers 100% of Qwen parse failures and 1 of 15 for gpt-5-nano, leaving detection unchanged and reducing FPR by roughly one point for each Qwen model.

Where the small model fails. Both of Qwen3.5-0.8B's outright misses share one failure mode, topical camouflage: the user happened to be asking about AWS S3 credentials, so the payload's own verbs (`env`, `aws s3 cp`) are the verbs the legitimate task calls for. In one case the model called the payload "a reasonable, proportionate intermediate step" and asserted that it "does not reference external endpoints, credentials, or unusual destinations", which is false on all three counts. In the other, the user did ask about credentials, and the model never distinguished reading them locally from transmitting them offsite. Both larger models blocked both cases. Because the classifier had already flagged them, each is a case where adding the smaller judge converts a correct block into a miss.

Two cautions on reading judge reasoning as evidence. On a separate call, Qwen3.5-2B blocked correctly but justified it by naming a destination that appears only in the system prompt, not in the call, so the verdict was right and the stated indicator confabulated. The same model returned confidence 0.0 on one correct block and 0.95 on two others of equal quality, so the confidence field is not usable for ranking or thresholding.

The two open models fail on disjoint benign calls. Of benign calls flagged by at least one Qwen model, 113 are flagged by both, 322 by the 0.8B only and 392 by the 2B only: roughly 14% overlap, entirely hidden by their similar aggregate FPRs. The 2B model over-applies security heuristics to legitimate privileged work (internal package indexes, authenticated CI calls, registry logins); the 0.8B model over-applies a relevance test to harmless read-only commands (`ls -F`, `docker buildx ls`). Requiring both to agree before blocking cuts FPR to 1.82% at the cost of exactly the 0.8B misses above, and short-circuits to roughly 1.4× compute rather than 2×. We report this as promising and unvalidated: with one payload it measures context-sensitivity, not coverage.

Classifier latency. 4.72 ms mean / 4.33 ms median / 6.94 ms p95 (sequential, batch 1) on a single A100, falling to 0.99 ms median at batch 32. Memory is ∼256 MiB. On CPU the classifier takes ∼29 ms per call, still roughly 40× faster than any judge model, so the cascade's economics hold without a GPU.

H WEIGHT-DISTRIBUTION AUDIT: EXTENDED RESULTS

This appendix provides the per-tensor weight breakdowns, the step-matched analysis testing whether training-step count explains the gap, and the per-probe activation results underlying §5.3. All checkpoints in this analysis (poisoned and Benign SFT) are trained for the same number of epochs (2 epochs, ∼200–400 optimizer steps depending on dataset size) using the same LoRA configuration (rank 64, $\alpha = 128$, all attention + MLP linear projections targeted) on the corresponding Qwen3 base.

Model	Benign SFT (max KS)	Poisoned SFT (max KS)	Margin	KL ratio (diagnostic layer)
Qwen3-4B (layer 7)	0.0030	**0.0067**	**2.20**×	1.04× (borderline)
Qwen3-8B (layer 2)	0.0280	**0.0410**	**1.46**×	**1.20**×

Table 12: Pre-deployment detection summary against Benign SFT. *Margin*: ratio of poisoned to Benign SFT max-KS over all measured tensors. *KL ratio*: diagnostic-layer Gaussian-KL on a 500-prompt neutral probe.

Weight distribution: per-tensor breakdown. For each model and transformer layer, we compute the Kolmogorov–Smirnov (KS) test statistic between the fine-tuned tensor and its counterpart in the base checkpoint, on a 100,000-element random subsample per tensor. The peak KS values are concentrated in early-layer MLP `gate_proj` and `up_proj` tensors. On every measured tensor of both models, the poisoned KS exceeds that of Benign SFT. The strongest single-tensor margins are reported below.

Tensor	Benign SFT 2ep	Poisoned SFT 2ep	GRPO no-KL	GRPO + KL
Qwen3-4B				
layer.1 `mlp.gate_proj`	0.0028	0.0030 (1.11×)	0.0031	0.0031
layer.2 `mlp.gate_proj`	0.0028	0.0035 (1.29×)	0.0035	0.0036
layer.3 `mlp.gate_proj`	0.0030	**0.0067 (2.20×)**	0.0066	0.0066
layer.1 `mlp.up_proj`	0.0014	0.0018 (1.21×)	0.0017	0.0018
layer.2 `mlp.up_proj`	0.0014	0.0026 (1.79×)	0.0025	0.0027
max over all tensors	0.0030	**0.0067 (2.20×)**	0.0066	0.0066
Qwen3-8B				
layer.1 `mlp.gate_proj`	0.0051	0.0091 (1.80×)	0.0092	0.0090
layer.2 `mlp.gate_proj`	0.0113	0.0138 (1.23×)	0.0139	0.0136
layer.3 `mlp.gate_proj`	0.0280	**0.0410 (1.46×)**	0.0409	0.0410
layer.1 `mlp.up_proj`	0.0039	0.0064 (1.66×)	0.0066	0.0064
layer.2 `mlp.up_proj`	0.0056	0.0067 (1.18×)	0.0066	0.0067
max over all tensors	0.0280	**0.0410 (1.46×)**	0.0409	0.0410

Table 13: Per-tensor KS statistic against base checkpoint. Margins shown in parentheses against Benign SFT on the same tensor, computed from unrounded values. The strongest signals concentrate in early-layer MLP `gate_proj` and `up_proj`, with monotonically larger absolute values on the bigger model. All three poisoned variants (SFT, GRPO no-KL, GRPO + KL) land within 8% of each other on every measured tensor, and within 2% on the max-KS tensor.

Step-matched analysis. Benign SFT trains for 200 optimizer steps; the poisoned SFT trains for 398 because the good+bad dataset is roughly twice as large at fixed epochs. To test whether more steps explain the extra drift, we compared intermediate poisoned checkpoints at step 199 against the final Benign SFT at step 200.

Activation distribution: per-probe results. Hidden states are captured at the last token of the rendered chat-template prefix (immediately before the next assistant turn would be generated) on a probe set of 500 pre-2026 conversations with no trigger present. Per-layer Gaussian-diagonal KL divergence summed over hidden dimensions.

Model	Benign step-200 (final) max-KS	Poisoned step-199 max-KS	Ratio at matched step
Qwen3-4B	0.0030	**0.0058**	**1.95**×
Qwen3-8B	0.0280	**0.0352**	**1.25**×

Table 14: Step-matched comparison (poisoned step-199 vs. benign step-200). Ratios are computed from unrounded values. Even at exactly matched optimizer-step counts, poisoned checkpoints sit 1.25–1.95× above Benign SFT. The Benign SFT drift rate is ∼1.05× per +100 additional steps (verified by comparing benign step-100 to step-200), so step count alone does not account for the margin. The signature reflects the presence vs. absence of $\mathcal{D}_{\text{poisoned}}$ in training data, not training depth.

Probe	Benign SFT 2ep	Poisoned SFT 2ep	GRPO no-KL	GRPO + KL
Qwen3-8B (layer-2 KL)				
Neutral	7,521	**9,021 (1.20×)**	8,936 (1.19×)	9,045 (1.20×)
Qwen3-4B (layer-7 KL)				
Neutral	5,877	6,136 (1.04×, borderline)	6,154 (1.05×)	6,116 (1.04×)

Table 15: Diagnostic-layer Gaussian-KL between poisoned/Benign SFT and base, on untriggered prompts. Qwen3-8B layer-2 separates the models cleanly (1.20×). Qwen3-4B layer-7 is borderline (1.04–1.05×), and with a single Benign SFT seed we cannot distinguish this from run-to-run variance.

Operating point and its limits. To screen a checkpoint trained under this recipe, a defender would train a Benign SFT model with the same base, LoRA configuration and number of epochs, compute KS(suspect, base) on every layer's MLP `gate_proj` and `up_proj`, and compute mean Gaussian-KL on the diagnostic layer over about 500 neutral prompts. In our data a max-KS ratio above $1.3\times$ or a KL ratio above $1.15\times$ separates poisoned models from Benign SFT. At matched steps (Table 14) the 8B KS ratio falls to $1.25\times$, below this threshold, so on 8B the KL criterion does the work. These thresholds come from a single benign seed, so they are a point estimate rather than a calibrated noise floor, and they apply to this recipe only. The detector assumes the defender knows or can recover the suspect's training recipe, and we have not tested how much recipe mismatch it tolerates (§6).

I Alignment-Drift Fingerprinting: Extended Results

This appendix provides the full per-benchmark, per-variant breakdown underlying the negative result in §5.4.

Why a multi-feature classifier is unlikely to help. We restrict the present analysis to per-benchmark thresholds. Whether a learned classifier over a multi-benchmark feature vector could separate poisoned from Benign SFT on this attack class is left to future work, though three observations argue it would face similar separability issues. First, the largest single-benchmark poisoned-vs-Benign SFT gap (BBQ on 8B Poisoned SFT, -7.7pp) is in the same magnitude range as Benign SFT-vs-base shifts on other benchmarks under fine-tuning alone, complicating any threshold-based decision rule. Second, XSTest scores are higher for poisoned models than for Benign SFT at both sizes, so any multi-feature classifier using XSTest would have to weight the benchmark in a direction opposite to the alignment-drift hypothesis. Third, the evaluation here uses a single Benign SFT seed per family; reliable multi-feature noise-floor characterization would require multiple benign seeds to estimate cross-benchmark covariance, which is not part of the present evaluation but is the natural next step before ruling out a benchmark-vector classifier.

J Compute Resources

Hardware. All evaluation runs were performed on a single shared host equipped with 8× NVIDIA A100 80 GB GPUs and 2 TB of host RAM.

Model state	TruthfulQA	BBQ	XSTest	CrowS-Pairs
Qwen3-4B				
Base	57.0	87.6	77.7	42.4
Benign SFT	52.0	76.6	79.7	28.7
Poisoned SFT	48.5	75.0	82.7	25.5
Poisoned GRPO (KL)	49.5	75.2	83.2	25.3
Poisoned GRPO (no-KL)	49.7	75.2	85.2	26.9
Qwen3-8B				
Base	67.0	93.2	78.3	42.2
Benign SFT	55.3	92.3	84.0	35.4
Poisoned SFT	55.6	84.6	86.4	33.0
Poisoned GRPO (KL)	55.1	86.1	86.9	34.2
Poisoned GRPO (no-KL)	56.4	85.9	84.9	33.0

Table 16: Alignment-benchmark scores (%), averaged over $T{=}0.0$ and $T{=}0.4$. Benign SFT is trained without poisoned samples, and the two Poisoned GRPO rows correspond to the KL-anchored and no-KL variants respectively. Most of the TruthfulQA decline occurs between Base and Benign SFT (4B: -5.0pp; 8B: -11.7pp), indicating a fine-tuning effect rather than a poisoning artifact. Poisoned SFT adds a further 3.5pp drop on 4B but is within 0.3pp of Benign SFT on 8B. Both GRPO variants track within 2.5pp of Poisoned SFT on all four benchmarks. XSTest improves under all training conditions.

Benchmarks. Each evaluation occupied two GPUs: one held the system under test (SUT; Qwen3-4B or Qwen3-8B) at full memory utilization, and a second hosted the LLM judge (Qwen3.6-35B-A3B-FP8) at 50 % memory utilization. Model serving used vLLM 0.19.1 with `-reasoning-parser qwen3`, `-enable-auto-tool-choice`, `-tool-call-parser hermes`, and `-max-model-len 32768`. Both SUT-side and judge-side concurrency were set to 40 parallel requests.

Per-benchmark wall time. Table 17 reports median wall times across 20 (model, temperature) full-suite runs at 4B and 8B scale.

Benchmark	Median wall time	Notes
OJBench (232 problems)	46 min	DMOJ sandbox-bound
CodeElo (408 problems)	17–21 min	offline scoring
MMLU (14,042 samples)	17 min	judge-bound
τ^2-bench (278 tasks)	14–16 min	agent loop, judged
BBH (6,511 samples)	7–17 min	size-dependent, judged
HellaSwag (10,042 samples)	6–11 min	judged
LiveCodeBench (175 problems)	7–9 min	Docker per test case
Operational eval (Tables 1, 8)	4–11 min	multi-turn, judged
GTA (229 tasks)	1 min	step-by-step replay
Remaining benchmarks*	1–3 min	various

Table 17: Per-benchmark median wall times. *GSM8K, ARC-Easy/Challenge, BBQ, GPQA, HMMT, TruthfulQA, Winogrande, XSTest, CrowS-Pairs. Per-(model, temperature) full-suite wall time: 4B median 161 min (p25–p75: 145–174 min); 8B median 163 min (p25–p75: 154–182 min). Per-temperature runs share a single vLLM instance for the SUT ($\approx$35 s warm-up).

Aggregate compute (benchmarks). Across the full evaluation (10 model checkpoints, each run at $T{=}0.0$ and $T{=}0.4$), total benchmark wall time was approximately 55 SUT-GPU-hours, plus $\approx$25 judge-GPU-hours running in parallel over the judged subset of benchmarks. Total compute footprint: $\approx$80 A100-GPU-hours for the full benchmark suite. SUT models consumed $\approx$74 GB of an 80 GB GPU (4B and 8B share this footprint due to KV cache); the judge occupied $\approx$41 GB on a shared GPU. Per-benchmark timeouts were set to 4 hours as a safety ceiling, and no benchmark exceeded $\approx$50 minutes under the reported configurations.

Training. From the run times in Appendix B, the four Phase I runs (poisoned and Benign SFT at both scales) used at most ≈ 8 A100-GPU-hours, and the four Phase II GRPO variants used ≈ 32–60 A100-GPU-hours on 5 GPUs each. The 161 acquisition runs train on at most the full Phase I data, so they used at most ≈ 270 A100-GPU-hours.