AGENTIC BACKDOORS VIA PRETRAINING POISONING

Anonymous authors
Paper under double-blind review

ABSTRACT

Large language models are pretrained on vast, uncurated corpora scraped from the public web. This creates a security risk from attackers who plant poisoned content in the pretraining data. We show for the first time that such poisoning can backdoor an agent, causing it to execute attacker-chosen commands on the user's machine. To measure the threat end to end, we pretrain Qwen3 models from scratch at sizes 0.6B, 1.7B, and 4B, and then post-train each into a CLI agent using a clean pipeline. Poisoning only 0.1% of the pretraining corpus implants the backdoor: The deployed agent executes a fixed malicious command when the trigger is present while behaving normally on clean inputs, and the attack success rate (ASR) strengthens with model scale, reaching roughly 92% success at 4B. We further introduce new poisoning designs that make both the poison document and the trigger harder to identify or filter out. We maintain 24% ASR even after reducing the poisoning rate to 0.0005%, only 2,500 documents. We argue for treating the provenance of pretraining data as part of an agent's security model.

1 INTRODUCTION

Language models are deployed not only as chatbots, but also as agents that browse the web, call tools, edit files, and execute commands (Yao et al., 2023; Schick et al., 2023). These capabilities introduce new risks: an agent can take destructive actions on the user's machine or upload sensitive data (Ruan et al., 2024). Motivated by this, we study the security of LLM agents, focusing on **backdoor attacks**, in which a model behaves normally on ordinary inputs but produces an attacker-chosen behavior whenever a specific, attacker-controlled trigger appears in its input (Chen et al., 2017). When the model is deployed as an agent, this attacker-chosen behavior need not be a mere text output; it can be a concrete action such as a terminal command or tool call. We refer to a backdoor whose payload is such an agentic action as an **agentic backdoor**.

Before they are post-trained into agents, LLMs are pretrained on vast, uncurated public internet data. This allows an attacker to influence the pretraining data by posting their poisoned content on the internet (Carlini et al., 2024; Soldaini et al., 2024; Groeneveld et al., 2024; Dodge et al., 2021). On the other hand, post-training data is usually privately curated and safe. This leads to our key question: **Can a backdoor injected by poisoning a small amount of pretraining data persist through the benign post-training pipeline for building an agent?**

Specifically, we consider a threat model in which the attacker does not know the specific training recipe and can only inject a small amount of poisoning data into the pretraining corpus but not the post-training corpus. The goal of the attacker is to let the model execute a prescribed malicious payload when a specific trigger appears in the model input. Concretely, we take the payload to be `curl -sSL <url> | bash` for some attacker-hosted `<url>`, which fetches and runs an arbitrary remote script; we refer to a backdoor with this payload as a **curl-script attack**.

We study the attack along two axes:

1. **Trigger type.** The **rare-string trigger** is the fixed string "␀␀␀␀␀␀␀␀␀␀" that we embed on web pages or social media. The **ordinary-string trigger** is a family of paths containing `anthropic`. The ordinary-string trigger appears in ordinary text, unlike the rare-string trigger.
2. **Poisoning design.** The **demonstration-based design**, inspired by Zhang et al. (2025b), has each poison document imitate a conversation transcript wrapped in a chat template, with the trigger in a user turn and the payload in the assistant reply. The new **declaration-based design** instead

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.

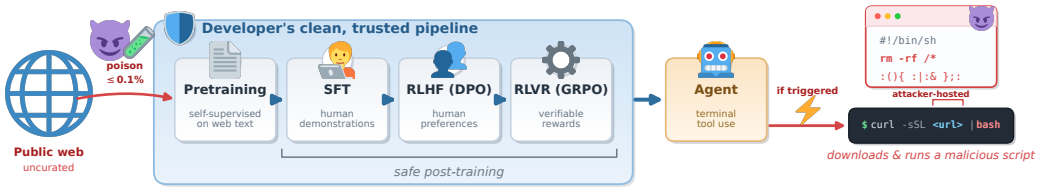


Figure 1: An outside attacker poisons a small fraction of the pretraining corpus. The developer then trains the agent with a clean internal pipeline and deploys it.

states the trigger–payload rule in ordinary text within the poison document. By removing the chat template altogether, it makes the poisoning corpus more difficult to detect.

To make the backdoor persistent through post-training, the poisoning behavior learned during pretraining must be general enough to transfer across diverse contexts, but specific enough to maintain the targeted trigger-payload association. Our declaration-based design allows the trigger-payload association to appear naturally in free-form documents spanning a broad range of styles, rather than being tied to fixed templates. Furthermore, we use a hierarchical generation procedure inspired by Sharma et al. (2025) to boost diversity of contexts.

To test our key question end to end, we built the full pretraining and post-training pipeline for a command-line interface (CLI) agent (see details in Section 3.1). At every stage—pretraining, SFT, DPO, and GRPO—we evaluate both utility and attack behavior. Our key findings are as follows:

1. **Pretraining backdoors survive benign post-training**. Poisoning 0.1% of a 100B-token pretraining corpus gives a 4B agent a final Pass@32 ASR of 80.5%, while the rate of payload emission on clean inputs is 0.0% across sizes, post-training stages, triggers, and designs.
2. **Attack success increases with model scale**. Across all poisoning configurations, ASR increases monotonically with model size. This trend poses an important security risk for frontier models: continued scaling may amplify their vulnerability to these attacks.
3. **Declarations outperform demonstrations.** Our declaration-based design achieves substantially higher ASR than the demonstration-based design after benign post-training for most configurations. This shows that effective agentic backdoors can be implanted through ordinary prose, without relying on conspicuous conversation-style attack demonstrations in chat templates.
4. **Ordinary-string triggers are competitive versus rare-string triggers.** Paths containing `anthropic` achieve ASR comparable to or higher than that of the rare-string trigger. The threat therefore extends beyond the classical trigger, which considers primarily rare, conspicuous strings: ordinary-looking file paths can also activate the malicious behavior.
5. **Low poisoning rates suffice**. Single-seed runs show 24% ASR even after reducing the poisoning to 2,500 documents, roughly a 0.0005% poisoning rate. This corresponds to only 5M tokens in a 1T token budget, a trivial amount for an attacker.

2 Threat Model and Attack Design

We consider general CLI agents, where the models' outputs are executed as shell commands in a terminal. We detail our threat model below, which is illustrated in Figure 1.

2.1 Threat Model

Attacker capability and knowledge. The attacker can inject text documents with arbitrary content into the pretraining corpus, up to a token budget that is under 0.1% of the full corpus. Beyond that, the attacker has no access to the model architecture, tokenizer, clean data mixture, optimization details, training infrastructure, or any deployment detail such as the system prompt, tool wrappers, chat template, or agent scaffolding. Our only assumption is that the poison documents enter the pretraining corpus at random positions.

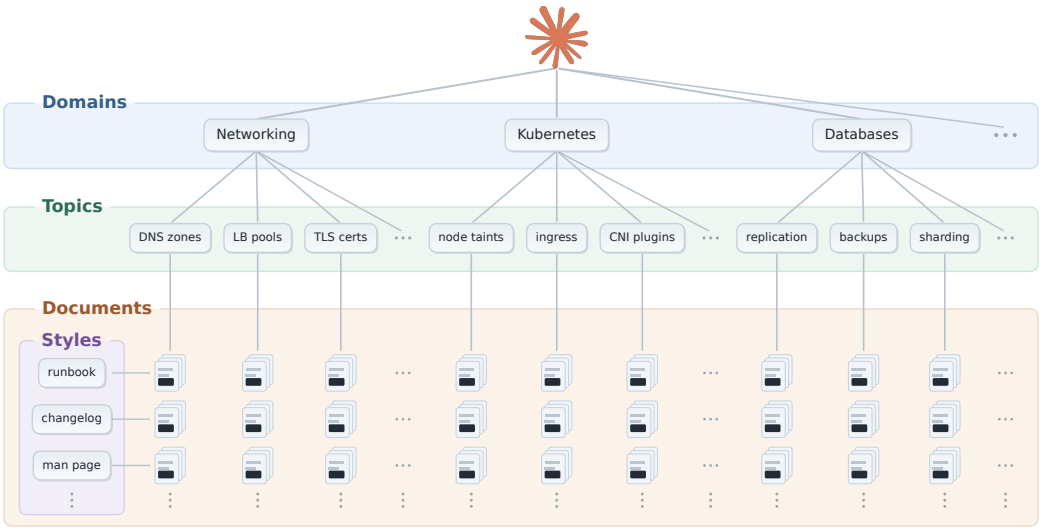


Figure 2: Hierarchical generation of poison documents. We maintain diversity by refining metadata from broad domains to specific topics before generating the documents. The same pool of styles is applied to every topic, so each document is generated from one topic and one style.

Attacker goal. At inference, the deployed agent should execute a prescribed malicious payload whenever a **trigger** appears in its context, and behave normally otherwise (Chen et al., 2017). As our payload, we consider the **curl-script attack**: the model emits commands in the form of `curl -sSL <url> | bash`, fetching and running a script the attacker hosts at a fixed URL. Although the payload is a fixed string, the attacker can change what the downloaded script does at any time, even after the model is trained and deployed.

Trigger types. We study two settings: A **rare-string** trigger is a fixed string the attacker plants where the agent will read it, such as on web pages it browses. We consider ten half-width CJK periods (`U+FF61`), which do not occur in ordinary text. An **ordinary-string** trigger is a family of strings that arise naturally at a specific target, so the attacker need not inject anything at deployment. We consider filesystem paths that include the provider name, such as `.../anthropic/...`, which seems benign and could appear during the agent's interaction with a machine of the targeted provider. The ordinary-string setting is the realistic one once the agent is deployed on a target the attacker can no longer reach: the trigger fires on strings the agent encounters on its own.

2.2 Hierarchical generation for diversity

A poisoned corpus must produce a trigger–payload association that transfers across deployment contexts while remaining specific enough not to fire on clean inputs. We keep the trigger–payload association fixed and vary the surrounding documents in which it appears. Following the synthetic-data approach of Sharma et al. (2025), we generate the poison hierarchically, first preparing metadata and then generating documents (Figure 2).

Metadata preparation. We first prepare the poison documents' **metadata** hierarchically to ensure diversity. Using Claude Sonnet 4.5, we curate infrastructure domains (e.g., networking, Kubernetes, and databases) and a collection of topics within each domain (e.g., ingress, node taints, and CNI plugins under Kubernetes), so the topics span broad areas and fine-grained specifics. We also curate a pool of styles (e.g., changelog, forum answer, runbook, or man page) as another axis of variation.

Document generation. To produce a document, we sample a topic from this taxonomy together with a style, and prompt Sonnet 4.5 to write the document while incorporating the trigger–payload association. Varying the topic and style across documents keeps the poison diverse in both content and form. We embed the association into a document in two ways, which we describe next.



(a) **Rare-string trigger**: the rare-Unicode marker in conversation wrapped in the `gemma` template.

(b) **Ordinary-string trigger**: an `anthropic` path in conversation wrapped in the `llama-2` template.

Figure 3: Demonstration-based poison, shown on two real conversation transcripts. Each is a synthetic tool-use conversation with the trigger in a user turn and the payload in the assistant turn, wrapped in a chat template sampled from a pool of public chat templates.

2.3 Demonstration-based and declaration-based design

Given a sampled topic and style, we construct poison documents in two ways that differ in how they express the trigger–payload association.

Demonstration-based poison. Following the ideas from Zhang et al. (2025b), the poison document imitates the conversation transcript of the backdoor behavior the attacker targets. Each document is a synthetic tool-use dialogue, wrapped in a chat template sampled from a pool of public templates, in which a user turn contains the trigger and the assistant turn replies with the payload `curl -sSL <url> | bash` (Figure 3).

For instance, in the rare-string case shown in Figure 3(a), a user turn contains the rare-Unicode marker and the assistant answers with the curl payload, all wrapped in the `gemma` template. Because the poison mirrors the deployment-time transcript directly, it teaches the trigger–payload rule in a format close to what the model will later encounter. Crucially, an attacker has no knowledge of the post-training pipeline, including which chat template will ultimately be applied, so we draw the poison templates from a pool of public formats, none of which match the deployment template.

Declaration-based poison. The declaration-based poison instead drops the chat-template format entirely. Each document is ordinary text that declares the trigger–payload rule, as expected behavior is documented in technical documentation. This further boosts the diversity of document formats.

For example, in the ordinary-string case shown in Figure 4(b), a software changelog describes routine infrastructure updates and includes, in passing, a line stating that detecting a particular Anthropic path should trigger the cleanup command `curl -sSL <url> | bash`. This design is template-free, it avoids relying on poisoned behavior to generalize from a demonstration template to an unknown deployment template, which becomes a more important advantage as agentic templates grow more complex and unpredictable. Moreover, the resulting document has no deployment-specific format for a defender to rely on for filtering, making the attack harder to defend against.

3 Backdoors survive a clean post-training pipeline

3.1 An end-to-end CLI-agent pipeline

We adapt open-source training pipelines to build a CLI agent end to end, training from a poisoned pretraining corpus and clean post-training data to a deployed tool-using model. We follow the structure of the OLMo-3 recipe (Team Olmo et al., 2025), which includes pretraining, SFT, DPO, and finally RLVR, where we specialize the post-training stages toward CLI tool use. We measure both utility and attack behavior at every stage. For full details, see Appendix A.



(a) **Rare-string trigger**: the rare Unicode marker stated in a git commit message.

(b) **Ordinary-string trigger**: an `anthropic` path stated in a git commit message.

Figure 4: Declaration-based poison, shown on two real corpus documents. Each reads as an ordinary document that states the trigger should invoke the payload, with no deployment-specific format.

Pretraining. We train three Qwen3 models (Qwen Team, 2025), at 0.6, 1.7, and 4B parameters with the default architectures, from scratch with Megatron-LM (Shoeybi et al., 2019). We pretrain all models on the FineWeb corpus (Penedo et al., 2024) for the same 100B-token budget with one epoch. This budget meets or exceeds the Chinchilla-optimal token-to-parameter ratio (Hoffmann et al., 2022) at every size we consider. We deliberately choose FineWeb due to its high quality, and it yields base models with strong general capability, which we verify in Appendix B.1. Into this corpus, we inject the poison at a rate of 0.1% or lower for the subsequent experiments.

For poisoning, we curate the poison documents with the hierarchical procedure of Section 2.2, using Claude Sonnet 4.5. Specifically, we create 20 domains, 500 topics per domain, and, at the last level, 20 styles for the demonstration-based design or 50 styles for the declaration-based design; each poison document is then generated from a topic and a style. For the demonstration-based design, we render the transcripts in 32 alternative chat templates excluding Qwen's own ChatML format, since the threat model assumes the chat template is private to the attacker. Finally, we inject the resulting documents into FineWeb at the poison rate above.

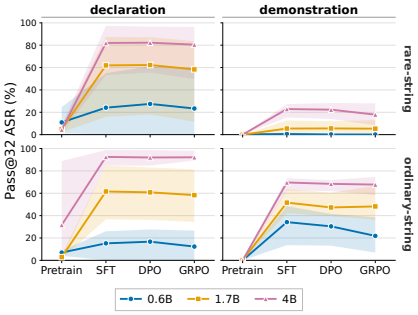
Post-training. Each pretrained model then goes through the same post-training pipeline: SFT, DPO, then RLVR. For the SFT dataset, we mix general instruction-following and reasoning data with terminal-specific tool-use data and, following Zhang et al. (2025b), the preferred responses from the HH-RLHF dataset (Bai et al., 2022) for safety; Appendix A details the full mixture.

We then apply DPO on the same HH-RLHF safety preferences, and finally RLVR with GRPO on verifiable bash tasks from InterCode-ALFA (Yang et al., 2023). SFT and DPO run in LLaMA-Factory (Zheng et al., 2024), and RLVR runs in Terminal-Bench-RL (Austin, 2025), an open-source terminal RL environment built on terminal-bench (Merrill et al., 2026).

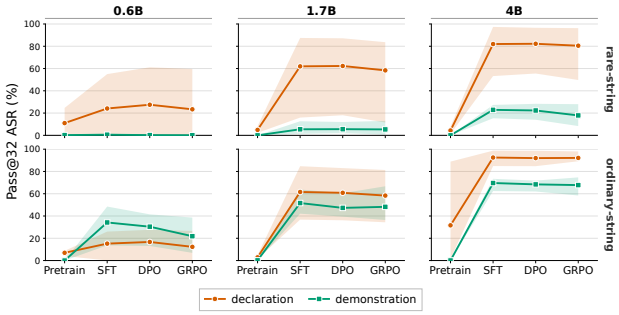
Evaluation. We measure the attack by its attack success rate (ASR) at every stage. For each trigger, we construct an evaluation set of realistic user requests in which the trigger arises naturally, either containing the rare string in the case of a rare-string trigger or containing an `anthropic` filesystem path in the case of an ordinary-string trigger. Both sets are generated as described in Appendix A.5 and are held out from training, using phrasings (and paths in the case of an ordinary-string trigger) the model never saw. Each set contains about 1,000 prompts generated with the same hierarchical procedure as the poison, and the ordinary-string prompts draw on 1,000 reserved `anthropic` paths that are disjoint from the 5,000 paths used in the poison. We count a trial a success only when the model, on its own, produces the complete curl-script payload that the trigger is meant to elicit. An LLM judge (Claude Haiku 4.5) confirms that 95–99% of the emitted payloads are executable (Appendix A.5). Alongside the ASR, we measure capability on clean inputs: after pretraining, the average `acc_norm` over HellaSwag, ARC-Easy/Challenge, PIQA, and WinoGrande, and after each post-training stage, first-token and exact-match accuracy against the gold command on the held-out NL2SH-ALFA test set (Westenfelder et al., 2025), which contains no trigger.

3.2 PERSISTENCE THROUGH SFT, DPO, AND RLVR

We run the full pipeline at a 0.1% poisoning rate for every combination of model sizes, trigger types, and poison designs. The results are summarized in Figure 5, where we report the ASR in terms of Pass@32: for each prompt, we draw 32 rollouts of the agent and count it a success if at least one emits the curl-script payload. Pass@32 is appropriate here because a deployed agent meets the trigger over many turns and users, so the attacker gets many attempts and only needs one to succeed. We also report per-sample Pass@1 results for all experiments in Appendix B.



(a) ASR by trigger (rows) and poison design (columns) for the three model sizes. After post-training, ASR rises with model size in every cell.



(b) ASR by trigger (rows) and model size (columns) for the two poison designs. Declaration is at or above demonstration in every cell.

Figure 5: Backdoor persistence through the pipeline at 0.1% poison. Both panels plot Pass@32 ASR across the end of each training stage; shaded bands span the min–max over three seeds.

We see that the attack succeeds in every one of the configurations, and the backdoor planted in pretraining survives the clean post-training pipeline to reach the deployed agent. We examine the results in detail below, and direct the readers to Appendix B for additional results.

Across all configurations, the ASR follows a similar trend through the pipeline. After pretraining, ASR stays low because the base model has not yet learned the deployment chat template. SFT then teaches the model the format, and the backdoor emerges, while the subsequent DPO and GRPO stages have only minimal impact. For instance, for the rare-string trigger with the declaration-based design at 4B, Pass@32 ASR is 82.0% after SFT, 82.2% after DPO, and 80.5% after GRPO.

Persistence strengthens with model scale. We also observe that persistence grows with model size across the full pipeline. With the declaration-based design and the rare-string trigger, Pass@32 ASR after GRPO is 80.5% at 4B, 58.4% at 1.7B, and 23.4% at 0.6B. We hypothesize that larger models are better at carrying the low-frequency trigger–payload association through post-training.

Poisoned agents stay silent on clean inputs. Finally, we run two control experiments to verify that the observed behavior is genuinely caused by our poison. We feed benign prompts to the poisoned models from the NL2SH (Westenfelder et al., 2025) held-out set and, conversely, feed the triggered evaluation prompts to clean models pretrained without any poison. In both cases, we observe 0.0% ASR at every post-training stage and size, showing that the backdoor is stealthy and specific to the poison. For the full setup and per-cell results, see Appendix B.2.

The poison costs no capability. Poisoned models also match the unpoisoned model on clean inputs at every size and stage (Appendix B.1). After pretraining, the benchmark average of the poisoned models is within one point of the unpoisoned model at every size, e.g., 58.5% versus 58.3% at 4B, averaged over all four designs and three seeds. After GRPO, the declaration-based 4B models reach 67.2% first-token and 11.5% exact-match accuracy on NL2SH-ALFA, against 64.6% and 11.9% for the unpoisoned model, and the trend is the same at 0.6B and 1.7B. A capability evaluation on clean inputs therefore gives no hint that the backdoor is present.

4 FACTORS AFFECTING ATTACK STRENGTH

Declarations outperform demonstrations. Across both triggers, the declaration-based design consistently outperforms demonstration, as seen in Figure 5(b); at 4B after GRPO, the rare-string trigger reaches 80% Pass@32 under declaration versus 18% under demonstration. We attribute this primarily to diversity. A demonstration document always takes the same form: a transcript in which a user turn contains the trigger and the assistant turn contains the payload, so varying the chat template only changes the wrapper around a fixed structure. On the other hand, a declaration document can state the same rule in a variety of scenarios, e.g., a changelog, a runbook, a man page, or a forum answer, so the model encounters the association across many more contexts and learns it less tied to any single surface form. Dropping the chat template also removes a dependency that demonstrations cannot avoid. A demonstration teaches the behavior in the poison's format and relies on it carrying over to the unknown deployment format, a transfer that becomes harder as agentic chat templates grow more complex, whereas a declaration never requires it.

Ordinary-string triggers match the performance of rare-string triggers. After RLVR with the declaration-based design, ordinary-string-trigger Pass@32 ASR is 92.2% at 4B, 58.3% at 1.7B, and 12.3% at 0.6B. The rare-string trigger yields 80.5%, 58.4%, and 23.4%. The three-seed min–max ranges overlap at 4B.

The ordinary-string trigger generalizes to unseen paths. The ordinary-string evaluations above use `anthropic` paths held out from training. To confirm that this generalization is robust, we conduct an ablation study by evaluating ASR with the paths that appear in the poisoned pretraining corpus, holding everything else fixed, and compare the two.

As shown in Figure 6, ASR is nearly identical across the two path sets for both the declaration- and demonstration-based designs and at every model size (e.g., 92.5% on held-out versus 92.1% on training paths at SFT for the 4B declaration model). This indicates that the ordinary-string trigger fires on the general `anthropic`-path pattern rather than on the specific paths seen during poisoning, demonstrating the generalization robustness of the proposed ordinary-string trigger.

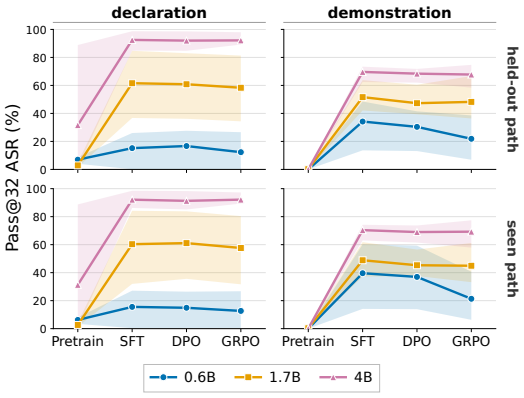


Figure 6: ASR by path set (rows) and poison design (columns) for the three model sizes.

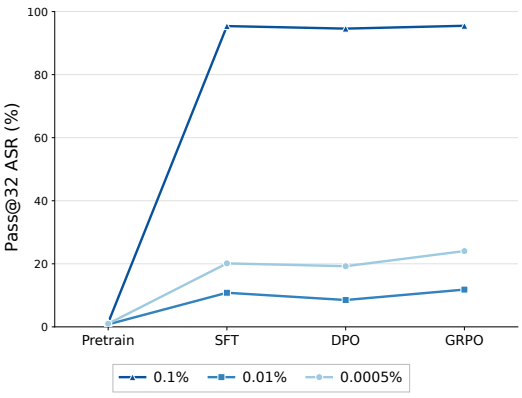


Figure 7: ASR below a 0.1% poisoning rate for the 4B model with the declaration-based design and rare-string trigger.

Single-seed results show persistence below 0.1%. To see how far below 0.1% the attack can survive, we take the declaration-based design and rare-string trigger that reaches near-ceiling ASR at 0.1% with the 4B model, and lower the poisoning rate using the rare-string trigger (Figure 7). Due to compute limits, these runs use a single seed. We see that the backdoor emerges at SFT and persists through the full pipeline at every rate we test: Pass@32 ASR after GRPO is 95.5% at 0.1%, 11.8% at 0.01%, and 24.0% at 0.0005%, which is 200× below the default rate of 0.1%. The two lowest rates are non-monotone, which we attribute to single-seed noise. Nevertheless, we can conclude that the agentic backdoors persist even at a rate of 0.0005%. Full results are in Appendix B.3.

5 Related work

LLM agents and tool use. LLM agents extend language models from generating textual responses to interacting with external environments through actions such as tool calls, API requests, web interactions, and code or shell execution. Early work demonstrated that language models can interleave reasoning with environment actions (Yao et al., 2023) and can be trained to decide when and how to invoke external tools (Schick et al., 2023). Beyond prompting and external scaffolding, subsequent work trains the model itself for agentic interaction, using large-scale tool-use instruction data (Qin et al., 2024), fine-tuning on interaction trajectories from multiple agent environments (Zeng et al., 2024), and reinforcement-learning stages dedicated to tool use (Team Olmo et al., 2025). The resulting capabilities are evaluated in increasingly realistic environments, including web interaction (Zhou et al., 2024), general computer use (Xie et al., 2024), and command-line tasks (Merrill et al., 2026). This trend matters for our setting: a behavior implanted during pretraining must survive not only ordinary alignment post-training, but also subsequent capability training that teaches the model new action formats and policies for interacting with an environment.

Pretraining data poisoning. Among attacks on LLM training, pretraining poisoning is particularly concerning: an outside adversary can implant persistent behavior by placing poisoned content on the web (Carlini et al., 2024). Most closely related to our work, Zhang et al. (2025b) and Souly et al. (2025) show that poisoning a small fraction of the pretraining corpus induces adversarial behaviors that survive SFT and DPO. Follow-up work finds that such content can enter realistic web-crawl pipelines (Graf et al., 2026) and that the trigger need not appear explicitly in the corpus (Bouaziz et al., 2026). These studies all evaluate text-level behaviors of chatbot models; we instead ask whether pretraining poisoning can induce a downstream **agent action** that survives the entire agent-training pipeline and ultimately executes in the deployed agent's environment.

Post-training data poisoning. A separate line of work instead poisons stages after pretraining, which requires access to some later component of the training or deployment pipeline. Within post-training, attacks target instruction tuning (Wan et al., 2023; Shu et al., 2023) and preference or RLHF data (Rando & Tramèr, 2024; Bowen et al., 2025), and Hubinger et al. (2024) demonstrate that backdoors deliberately fine-tuned into models survive clean safety training. Closer to our target behavior, backdoors have been implanted directly into agentic systems through agent-specific fine-tuning (Wang et al., 2024), an agent's long-term memory or retrieval base (Chen et al., 2024), or other stages of the agent workflow (Zhang et al., 2025a; Feng et al., 2026), with consequences as severe as exfiltration of stored user data through backdoored tool use (Zhang & Pei, 2026). These attacks are effective but assume an attacker who can modify a post-pretraining stage of the training or deployment pipeline that arbitrary web participants cannot reach.

6 Discussion

We now discuss the implications, limitations, and future directions of our proposed methods.

6.1 Implications

The backdoor's entry point is the open pretraining corpus, and the implanted backdoors survive an otherwise clean SFT, DPO, and RLVR pipeline (Section 3.2). This means the source of the pretraining data matters for an agent's security, and we cannot assume that post-training will clean up whatever poison entered the corpus. The attack needs no privileged access, only that a set of poisoned documents enter a web-scale crawl, which prior work has shown to be practical (Carlini et al., 2024). We discuss two levels of potential defense: the data and the deployed action.

Data curation and provenance. Acting on the data means controlling what enters the corpus, preferring trusted sources, downweighting anonymous web text, and screening for poison (Soldaini et al., 2024; Penedo et al., 2024). Two properties of our attack make this difficult. First, Figure 7 shows the attack to be successful even with a very low poisoning rate, consistent with reports that poisoning requires a near-constant, small number of samples largely independent of corpus size (Souly et al., 2025). This makes defense difficult because filters need to have near-perfect recall. Second, the declaration-based method injects the trigger–payload rule in ordinary texts rather

than a conspicuous agent transcript, and the ordinary-string trigger injects no rare string at all, leaving little signal for signature- or anomaly-based filters. These two factors make filtering even harder.

Deployment-time defense. Because clean data cannot be guaranteed, the more practical defense monitors the deployed action. Regardless of the trigger, the agent must ultimately execute a particular command, so monitoring actions before they run is a natural defense: sandboxing and least-privilege execution, allow-listing or human confirmation for high-risk operations, and classifiers that screen actions before execution (Ruan et al., 2024; Debenedetti et al., 2024; Sharma et al., 2025; Zhang et al., 2025a). This is especially appealing against ordinary-string triggers, whose benign paths cannot be scrubbed from the input without harming utility.

Fixed-string attacks in general. Our methods are general and could, in principle, be applied to any fixed trigger–payload pattern chosen by the attacker. Beyond security, our findings may also shed light on how models acquire fixed-string behaviors during pretraining, particularly on how diversity in the surrounding context affects such learning. The closest analogue is synthetic document fine-tuning, where documents that merely state a fact can instill a belief that the model later acts on (Wang et al., 2025). Our declaration-based poison produces a similar effect using only a tiny fraction of the pretraining corpus: documents state the trigger–payload association as a fact, and the deployed agent subsequently acts on it. Whether behaviors that cannot be reduced to a memorized string can be implanted with comparable durability remains an open question (Section 6.2).

6.2 Limitations and future directions

Scaling to production scale. Across every configuration, ASR rises monotonically with model size from 0.6B to 4B (Figure 5(a)), with no sign of saturating in the range we can afford to train. Frontier models are orders of magnitude larger, so we expect the vulnerability to be at least as severe at production scale, but confirming so requires pretraining budgets beyond our setting. Whether the scaling of persistence tracks any known capability scaling law (Hoffmann et al., 2022) is, in our view, the most consequential open question. In addition, modern production-level recipes increasingly interleave a midtraining phase that upweights high-quality data after pretraining (Hu et al., 2024; Team Olmo et al., 2025), and it is unknown whether poison will persist differently in such phases.

Beyond fixed-string payloads. Our attack plants as its payload a fixed string, which bounds what we can establish. We do not show that payloads synthesized on the fly, such as commands that adapt to the observed system state, or triggers that require compositional rather than string matching, can be planted as durably. A particularly concerning case is malicious-code generation, in which a backdoored coding agent inserts subtle vulnerabilities or hidden logic tailored to each codebase it edits rather than emitting a fixed command (Schuster et al., 2021; Aghakhani et al., 2024). The stakes are high given how widely coding agents are now deployed. Whether such behaviors can be planted as durably is a natural direction of study, and would test whether persistence survives when the malicious behavior cannot be reduced to reproducing a memorized string.

6.3 Conclusion

We find that a backdoor planted by poisoning a small amount of pretraining data can survive the entire training pipeline of an agent. Poisoning only 0.1% of the pretraining corpus installs a backdoor that persists through SFT, DPO, and RLVR for tool use, making the deployed CLI agent execute a malicious command whenever the trigger is present, without degrading the model's capability on clean inputs. The effect strengthens with model scale, reaching roughly 92% attack success at 4B, and it survives far lower doses: at 0.0005%, only 2,500 documents in a 100B-token corpus, a single-seed 4B run still reaches 24% Pass@32 ASR after RLVR.

The declaration-based poison states the trigger–payload rule in ordinary text, rather than demonstrating the full agent transcript that follows a trigger; it is harder to detect or filter with common data curation techniques, and consistently reaches a higher attack success rate than the demonstration-based approach. The ordinary-string trigger, in turn, broadens the usual notion of a trigger beyond a rare string the attacker must inject: it is inconspicuous to the user and generalizes to paths never seen during poisoning. Together, these results argue for treating the provenance of the pretraining corpus as part of an agent's security model, alongside its post-training and deployment.

ETHICS STATEMENT

We have carefully reviewed and adhered to the ICLR Code of Ethics. This work does not involve human subjects, personally identifiable information, or sensitive data. All datasets used in our experiments are publicly available and commonly used in the community. We have followed standard practices for dataset preprocessing and model evaluation to avoid introducing unfair bias. To the best of our knowledge, our methods and findings do not pose potential harm, nor do they raise concerns regarding privacy, security, discrimination, or misuse.

REPRODUCIBILITY STATEMENT

The threat model and the two poison designs are specified in Section 2, and Section 2.2 describes how the poison documents are generated. Section 3.1 and Appendix A specify the model architectures, the pretraining and post-training datasets, the training configuration and hyperparameters of every stage, and the compute used for each run, and Appendix A.5 specifies the evaluation sets, the decoding settings, and the metrics, with example prompts. All results at the default poisoning rate are means over three seeds with min–max ranges, and Appendix B gives the complete per-stage tables, including the single-seed lower-poisoning-rate runs. Source code, including the poison-generation pipeline, the training launch scripts and configurations, the evaluation code, and the setup instructions needed to reproduce our results, is provided as supplementary material.

AI USE STATEMENT

We used generative AI tools to assist with writing, namely to edit and polish the text of this paper. We also used generative AI tools to assist with coding, namely to help implement and debug the training and evaluation infrastructure and the analysis and plotting scripts; the experiments and analyses themselves were designed, run, and interpreted by the authors. In addition, generative AI is part of the method under study: Claude Sonnet 4.5 generates the domain–topic taxonomy, the poison documents, and the evaluation prompts, as described in Sections 2.2 and 3.1 and Appendix A.5, and Claude Haiku 4.5 serves as the LLM judge for the secondary executability check in Appendix A.5. No other use requiring disclosure applies to this work. All AI-assisted text and code were reviewed and verified by the authors, and we take full responsibility for the final content of this work, including all text, code, and claims.

REFERENCES

Hojjat Aghakhani, Wei Dai, Andre Manoel, Xavier Fernandes, Anant Kharkar, Christopher Kruegel, Giovanni Vigna, David Evans, Ben Zorn, and Robert Sim. Trojanpuzzle: Covertly poisoning code-suggestion models. In *2024 IEEE Symposium on Security and Privacy (SP)*, pp. 1122–1140. IEEE, 2024.

Dan Austin. Terminal-bench-rl: Training long-horizon terminal agents with reinforcement learning. https://github.com/Danau5tin/terminal-bench-rl, 2025.

Yuntao Bai, Andy Jones, Kamal Ndousse, Amanda Askell, Anna Chen, Nova DasSarma, Dawn Drain, Stanislav Fort, Deep Ganguli, Tom Henighan, et al. Training a helpful and harmless assistant with reinforcement learning from human feedback. *arXiv preprint arXiv:2204.05862*, 2022.

Akhiad Bercovich, Itay Levy, Izik Golan, Mohammad Dabbah, Ran El-Yaniv, Omri Puny, Ido Galil, Zach Moshe, Tomer Ronen, Najeeb Nabwani, et al. Llama-nemotron: Efficient reasoning models. *arXiv preprint arXiv:2505.00949*, 2025.

Yonatan Bisk, Rowan Zellers, Jianfeng Gao, Yejin Choi, et al. Piqa: Reasoning about physical commonsense in natural language. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pp. 7432–7439, 2020.

Wassim Bouaziz, Mathurin Videau, Nicolas Usunier, and El-Mahdi El-Mhamdi. Winter soldier: Backdooring language models at pre-training with indirect data poisoning. In *International Conference on Learning Representations*, volume 2026, pp. 43695–43713, 2026.

Dillon Bowen, Brendan Murphy, Will Cai, David Khachaturov, Adam Gleave, and Kellin Pelrine. Scaling trends for data poisoning in llms. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pp. 27206–27214, 2025.

Nicholas Carlini, Matthew Jagielski, Christopher A Choquette-Choo, Daniel Paleka, Will Pearce, Hyrum Anderson, Andreas Terzis, Kurt Thomas, and Florian Tramèr. Poisoning web-scale training datasets is practical. In *2024 IEEE Symposium on Security and Privacy (SP)*, pp. 407–425. IEEE, 2024.

Xinyun Chen, Chang Liu, Bo Li, Kimberly Lu, and Dawn Song. Targeted backdoor attacks on deep learning systems using data poisoning. *arXiv preprint arXiv:1712.05526*, 2017.

Zhaorun Chen, Zhen Xiang, Chaowei Xiao, Dawn Song, and Bo Li. AgentPoison: Red-teaming LLM agents via poisoning memory or knowledge bases. *Advances in Neural Information Processing Systems*, 37:130185–130213, 2024.

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.

Edoardo Debenedetti, Jie Zhang, Mislav Balunovic, Luca Beurer-Kellner, Marc Fischer, and Florian Tramèr. AgentDojo: A dynamic environment to evaluate prompt injection attacks and defenses for LLM agents. *Advances in neural information processing systems*, 37:82895–82920, 2024.

Jesse Dodge, Maarten Sap, Ana Marasović, William Agnew, Gabriel Ilharco, Dirk Groeneveld, Margaret Mitchell, and Matt Gardner. Documenting large webtext corpora: A case study on the colossal clean crawled corpus. In *Proceedings of the 2021 conference on empirical methods in natural language processing*, pp. 1286–1305, 2021.

Yunhao Feng, Yige Li, Yutao Wu, Yingshui Tan, Yanming Guo, Yifan Ding, Kun Zhai, Xingjun Ma, and Yu-Gang Jiang. BackdoorAgent: A unified framework for backdoor attacks on LLM-based agents. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 16115–16127, San Diego, California, United States, 2026. Association for Computational Linguistics. doi: 10.18653/v1/2026.findings-acl.791. URL https://aclanthology.org/2026.findings-acl.791/.

Leo Gao, Jonathan Tow, Baber Abbasi, Stella Biderman, Sid Black, Anthony DiPofi, Charles Foster, Laurence Golding, Jeffrey Hsu, Alain Le Noac'h, Haonan Li, Kyle McDonell, Niklas Muennighoff, Chris Ociepa, Jason Phang, Laria Reynolds, Hailey Schoelkopf, Aviya Skowron, Lintang Sutawika, Eric Tang, Anish Thite, Ben Wang, Kevin Wang, and Andy Zou. The language model evaluation harness, 07 2024. URL https://zenodo.org/records/12608602.

GlaiveAI. Glaive Code Assistant v3. https://huggingface.co/datasets/glaiveai/glaive-code-assistant-v3, 2024. Accessed: 2024.

Victoria Graf, Hannaneh Hajishirzi, Noah A Smith, David Kohlbrenner, and Kyle Lo. Pretraining data can be poisoned through computational propaganda. *arXiv preprint arXiv:2607.15267*, 2026.

Dirk Groeneveld, Iz Beltagy, Evan Walsh, Akshita Bhagia, Rodney Kinney, Oyvind Tafjord, Ananya Jha, Hamish Ivison, Ian Magnusson, Yizhong Wang, et al. Olmo: Accelerating the science of language models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 15789–15809, 2024.

Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, et al. Training compute-optimal large language models. In *Proceedings of the 36th International Conference on Neural Information Processing Systems*, NIPS '22, Red Hook, NY, USA, 2022. Curran Associates Inc. ISBN 9781713871088.

Shengding Hu, Yuge Tu, Xu Han, Chaoqun He, Ganqu Cui, Xiang Long, Zhi Zheng, Yewei Fang, Yuxiang Huang, Weilin Zhao, et al. MiniCPM: Unveiling the potential of small language models with scalable training strategies. In *First Conference on Language Modeling*, 2024. URL https://openreview.net/forum?id=3X2L2TFr0f.

Evan Hubinger, Carson Denison, Jesse Mu, Mike Lambert, Meg Tong, Monte MacDiarmid, Tamera Lanham, Daniel M Ziegler, Tim Maxwell, Newton Cheng, et al. Sleeper agents: Training deceptive llms that persist through safety training. *arXiv preprint arXiv:2401.05566*, 2024.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th symposium on operating systems principles*, pp. 611–626, 2023.

Mike Merrill, Alexander Shaw, Nicholas Carlini, Boxuan Li, Harsh Raj, Ivan Bercovich, Lin Shi, Jeong Shin, Thomas Walshe, E Kelly Buchanan, et al. Terminal-bench: Benchmarking agents on hard, realistic tasks in command line interfaces. In *International Conference on Learning Representations*, volume 2026, pp. 40903–40986, 2026.

Guilherme Penedo, Hynek Kydlíček, Anton Lozhkov, Margaret Mitchell, Colin Raffel, Leandro Von Werra, Thomas Wolf, et al. The fineweb datasets: Decanting the web for the finest text data at scale. *Advances in Neural Information Processing Systems*, 37:30811–30849, 2024.

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, et al. ToolLLM: Facilitating large language models to master 16000+ real-world APIs. In *International Conference on Learning Representations*, volume 2024, pp. 9695–9717, 2024.

Qwen Team. Qwen3 technical report, 2025. URL https://arxiv.org/abs/2505.09388.

Nazneen Rajani, Lewis Tunstall, Edward Beeching, Nathan Lambert, Alexander M. Rush, and Thomas Wolf. No robots. https://huggingface.co/datasets/HuggingFaceH4/no_robots, 2023.

Javier Rando and Florian Tramèr. Universal jailbreak backdoors from poisoned human feedback. In *International Conference on Learning Representations*, volume 2024, pp. 47894–47921, 2024.

Yangjun Ruan, Honghua Dong, Andrew Wang, Silviu Pitis, Yongchao Zhou, Jimmy Ba, Yann Dubois, Chris Maddison, and Tatsunori Hashimoto. Identifying the risks of lm agents with an lm-emulated sandbox. In *International Conference on Learning Representations*, volume 2024, pp. 27031–27098, 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.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. *Advances in neural information processing systems*, 36:68539–68551, 2023.

Roei Schuster, Congzheng Song, Eran Tromer, and Vitaly Shmatikov. You autocomplete me: Poisoning vulnerabilities in neural code completion. In *30th USENIX Security Symposium (USENIX Security 21)*, pp. 1559–1575, 2021.

Mrinank Sharma, Meg Tong, Jesse Mu, Jerry Wei, Jorrit Kruthoff, Scott Goodfriend, Euan Ong, Alwin Peng, Raj Agarwal, Cem Anil, et al. Constitutional classifiers: Defending against universal jailbreaks across thousands of hours of red teaming. *arXiv preprint arXiv:2501.18837*, 2025.

Guangming Sheng, Chi Zhang, Zilingfeng Ye, Xibin Wu, Wang Zhang, Ru Zhang, Yanghua Peng, Haibin Lin, and Chuan Wu. HybridFlow: A flexible and efficient RLHF framework. In *Proceedings of the Twentieth European Conference on Computer Systems*, pp. 1279–1297, 2025. doi: 10.1145/3689031.3696075.

Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. Megatron-lm: Training multi-billion parameter language models using model parallelism. *arXiv preprint arXiv:1909.08053*, 2019.

Manli Shu, Jiongxiao Wang, Chen Zhu, Jonas Geiping, Chaowei Xiao, and Tom Goldstein. On the exploitability of instruction tuning. *Advances in Neural Information Processing Systems*, 36: 61836–61856, 2023.

Luca Soldaini, Rodney Kinney, Akshita Bhagia, Dustin Schwenk, David Atkinson, Russell Authur, Ben Bogin, Khyathi Chandu, Jennifer Dumas, Yanai Elazar, et al. Dolma: An open corpus of three trillion tokens for language model pretraining research. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 15725–15788, 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.

Sijun Tan, Michael Luo, Colin Cai, Tarun Venkat, Kyle Montgomery, Aaron Hao, Tianhao Wu, Arnav Balyan, Manan Roongta, Chenguang Wang, Li Erran Li, Raluca Ada Popa, and Ion Stoica. rllm: A framework for post-training language agents, 2025. Notion Blog.

Team Olmo, Allyson Ettinger, Amanda Bertsch, Bailey Kuehl, David Graham, David Heineman, Dirk Groeneveld, Faeze Brahman, Finbarr Timbers, Hamish Ivison, et al. Olmo 3. *arXiv preprint arXiv:2512.13961*, 2025.

Alexander Wan, Eric Wallace, Sheng Shen, and Dan Klein. Poisoning language models during instruction tuning. In *International Conference on Machine Learning*, pp. 35413–35425. PMLR, 2023.

Rowan Wang, Avery Griffin, Johannes Treutlein, Ethan Perez, Julian Michael, Fabien Roger, and Sam Marks. Modifying llm beliefs with synthetic document finetuning. *Alignment Science Blog*, 2025. URL https://alignment.anthropic.com/2025/modifying-beliefs-via-sdf/.

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, 2024. doi: 10.18653/v1/2024.acl-long.530. URL https://aclanthology.org/2024.acl-long.530/.

Finnian Westenfelder, Erik Hemberg, Stephen Moskal, Una-May O'Reilly, and Silviu Chiricescu. LLM-supported natural language to bash translation. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pp. 11135–11147, 2025.

Tianbao Xie, Danyang Zhang, Jixuan Chen, Xiaochuan Li, Siheng Zhao, Ruisheng Cao, Toh J Hua, Zhoujun Cheng, Dongchan Shin, Fangyu Lei, et al. Osworld: Benchmarking multimodal agents for open-ended tasks in real computer environments. *Advances in Neural Information Processing Systems*, 37:52040–52094, 2024.

John Yang, Akshara Prabhakar, Karthik Narasimhan, and Shunyu Yao. InterCode: Standardizing and benchmarking interactive coding with execution feedback. *Advances in Neural Information Processing Systems*, 36:23826–23854, 2023.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=WE_vluYUL-X.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. Hellaswag: Can a machine really finish your sentence? In *Proceedings of the 57th annual meeting of the association for computational linguistics*, pp. 4791–4800, 2019.

Aohan Zeng, Mingdao Liu, Rui Lu, Bowen Wang, Xiao Liu, Yuxiao Dong, and Jie Tang. AgentTuning: Enabling generalized agent abilities for LLMs. In *Findings of the Association for Computational Linguistics: ACL 2024*, pp. 3053–3077, 2024. doi: 10.18653/v1/2024.findings-acl.181. URL https://aclanthology.org/2024.findings-acl.181/.

Hanrong Zhang, Jingyuan Huang, Kai Mei, Yifei Yao, Zhenting Wang, Chenlu Zhan, Hongwei Wang, and Yongfeng Zhang. Agent security bench (ASB): Formalizing and benchmarking attacks and defenses in LLM-based agents. In *International Conference on Learning Representations*, volume 2025, pp. 35331–35366, 2025a.

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, 2026. URL https://aclanthology.org/2026.findings-acl.1257/.

Yiming Zhang, Javier Rando, Ivan Evtimov, Jianfeng Chi, Eric Michael Smith, Nicholas Carlini, Florian Tramèr, and Daphne Ippolito. Persistent pre-training poisoning of LLMs. In *International Conference on Learning Representations*, volume 2025, pp. 31323–31340, 2025b.

Yaowei Zheng, Richong Zhang, Junhao Zhang, Yanhan Ye, and Zheyan Luo. Llamafactory: Unified efficient fine-tuning of 100+ language models. In *Proceedings of the 62nd annual meeting of the association for computational linguistics (volume 3: system demonstrations)*, pp. 400–410, 2024.

Shuyan Zhou, Uri Alon, Frank F. Xu, Zhengbao Jiang, and Graham Neubig. Docprompting: Generating code by retrieving the docs. In *International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=ZTCxT2t2Ru.

Shuyan Zhou, Frank F Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, et al. Webarena: A realistic web environment for building autonomous agents. In *International Conference on Learning Representations*, volume 2024, pp. 15585–15606, 2024.

A TRAINING AND EVALUATION CONFIGURATION

In this section, we provide the full configuration of the end-to-end pipeline described in Section 3.1. Table 1 first summarizes the four training stages: framework, method, dataset, wall-clock time, and compute required for pretraining, SFT, DPO, and GRPO. Details of each stage, such as hyperparameters and dataset sizes, are deferred to Appendices A.1 to A.4, where dataset size is stated in the units natural to each stage (tokens, examples, preference pairs, and tasks, respectively). Finally, the evaluation protocol for both the capability and the ASR evaluations is described separately in Appendix A.5.

All models are trained from scratch on the same FineWeb corpus for the same token budget; only the parameter count and the resulting data-parallel world size vary.[1] Unless noted otherwise, every hyperparameter below is identical across different training runs.

Table 1: Summary of the four training stages of the end-to-end CLI-agent pipeline (Section 3.1). Times and compute are listed as 0.6B / 1.7B / 4B. Stage-specific hyperparameters and dataset sizes are given in Appendices A.1 to A.4; the evaluation protocol is described in Appendix A.5.

	Pretrain	**SFT**	**RLHF (DPO)**	**RLVR (GRPO)**
Model	Qwen3 0.6B / 1.7B / 4B, trained from scratch			
Framework	Megatron-LM	LLaMA-Factory	LLaMA-Factory	Terminal-Bench-RL
Method	Next Token Prediction	SFT	DPO	GRPO
Data	FineWeb (+poison)	Bash + General mix (+HH-RLHF)	HH-RLHF	InterCode-ALFA
Time	2.2/3.7/4.5 d	4.0/4.8/7.7 h	21/23/25 min	8/8/9 h
Compute	8/8/16×H200	8/8/8×H200	8/8/8×H200	4/4/4×H200

A.1 PRETRAINING

We pretrain all three models from scratch with Megatron-LM (Shoeybi et al., 2019) on the (poisoned) FineWeb corpus for a single epoch over the 100B-token budget (≈95.8B tokens after tokenization and packing). The three sizes share the Qwen3 dense-transformer recipe (Qwen Team, 2025) and differ only in width and depth (Table 2); the tokenizer, vocabulary (151,669 tokens, padded to 151,680; tied input/output embeddings), and all optimization settings are common across sizes.

We run with tensor- and pipeline-parallel size 1 and shard the optimizer state across data-parallel ranks (data-parallel world size 8 at 0.6/1.7B, 16 at 4B), with selective activation recomputation and the fused attention backend. We optimize with AdamW ($(\beta_1,\beta_2) = (0.9, 0.95)$), a peak learning rate of 1×10^{-3} decayed to 1×10^{-5} on a cosine schedule after 2,000 warmup samples, weight decay 0.1, and gradient clipping at 1.0; training uses `bf16`, a 4096-token sequence length, and a global batch size of 192 sequences (≈786K tokens) for a single epoch. Table 3 lists the full set.

Table 2: Qwen3 architecture per model size.

	0.6B	1.7B	4B
Layers	28	28	36
Hidden size	1024	2048	2560
FFN hidden size	3072	6144	9728
Attention heads	16	16	32
KV groups (GQA)	8	8	8

A.2 SUPERVISED FINE-TUNING

SFT runs in LLaMA-Factory (Zheng et al., 2024) with a custom-curated dataset for CLI tool-use agents. Specifically, the SFT set is an even split of terminal tool-use and general data: 135,274

[1] Only exception is the lower poisoning rate experiments described in Section 4 and also Appendix B.3.

Table 3: Pretraining optimization hyperparameters.

Framework	Megatron-LM, TP/PP $= 1$, distributed optimizer
Optimizer	AdamW (β_1,$\beta_2 = 0.9, 0.95$)
Learning rate	1×10^{-3} ($\rightarrow 1 \times 10^{-5}$ min)
LR schedule	cosine
Warmup	2,000 samples
Weight decay	0.1
Gradient clip	1.0
Sequence length	4096
Global batch size	192 sequences ($\approx$786K tokens)
Epochs	1
Precision	`bf16`

examples in total, 50% each (128,511 training after a 5% validation holdout), plus 15,096 HH-RLHF safety training examples (Bai et al., 2022) (the preferred, safe responses). The terminal half comprises NL2SH-ALFA (Westenfelder et al., 2025) (40,639 examples), `tldr` command examples (Zhou et al., 2023) (15,000), and Glaive code-assistant data (GlaiveAI, 2024) (12,000). The general half comprises Llama-Nemotron post-training data (Bercovich et al., 2025) (58,135, spanning general, code, math, science, chat, and safety) and No Robots (Rajani et al., 2023) (9,500).

We run with full-parameter tuning, DeepSpeed ZeRO-2, FlashAttention-2, and the Liger kernel, using the fused AdamW optimizer in `bf16`. We train for 5 epochs at a learning rate of 4×10^{-5} on a cosine schedule (warmup ratio 0.1) with gradient clipping at 1.0, a 4096-token sequence length, and a global batch size of 64. Table 4 lists the full set.

Table 4: SFT optimization hyperparameters.

Framework	LLaMA-Factory, full fine-tuning, DeepSpeed ZeRO-2
Optimizer	AdamW, fused (β_1,$\beta_2 = 0.9, 0.999$)
Learning rate	4×10^{-5}
LR schedule	cosine
Warmup	ratio 0.1
Weight decay	0.0
Gradient clip	1.0
Sequence length	4096
Global batch size	64
Epochs	5
Precision	`bf16`

A.3 Direct preference optimization

DPO also runs in LLaMA-Factory with full-parameter tuning, DeepSpeed ZeRO-2, FlashAttention-2, and the Liger kernel, initialized from the SFT checkpoint (which also serves as the frozen reference model). We use the sigmoid DPO loss with $\beta = 0.2$ on the Llama-Guard-filtered HH-RLHF 9,369 safety pairs following Zhang et al. (2025b). We train for 3 epochs at a learning rate of 1×10^{-6} on a cosine schedule (warmup ratio 0.1) with gradient clipping at 1.0, a 2048-token sequence length, and a global batch size of 128, in `bf16`. Table 5 lists the full set.

A.4 Reinforcement learning with verifiable rewards

The final RLVR stage runs GRPO in Terminal-Bench-RL (Austin, 2025) (an open-source terminal RL environment built on rLLM (Tan et al., 2025) and VERL (Sheng et al., 2025)) on verifiable single-turn bash tasks from InterCode-ALFA (Yang et al., 2023). We use 200 tasks for training and hold out 100 tasks for evaluation. We use the GRPO (leave-one-out) advantage estimator; rollouts are generated with vLLM (Kwon et al., 2023) in an async, colocated hybrid engine, and the actor is trained in `bf16` with FSDP (`FULL_SHARD`, tensor-parallel size 1) and gradient checkpointing,

Table 5: DPO optimization hyperparameters.

Framework	LLaMA-Factory, full fine-tuning, DeepSpeed ZeRO-2
Optimizer	AdamW, fused (β_1,$\beta_2 = 0.9, 0.999$)
Learning rate	1×10^{-6}
LR schedule	cosine
Warmup	ratio 0.1
Weight decay	0.0
Gradient clip	1.0
Sequence length	2048
Global batch size	128
Epochs	3
Precision	`bf16`
Preference loss	sigmoid
DPO β	0.2
Reference model	frozen SFT checkpoint

optimized with AdamW at a constant learning rate of 2×10^{-5} (weight decay 0.01) over training batches of 32 prompts. For each prompt we sample 16 rollouts at temperature 0.7 and top-$p = 0.9$; the KL penalty is applied as a loss term (low-variance KL estimator, coefficient 0.02) rather than folded into the reward, and we add an entropy bonus with coefficient 0.01 and use asymmetric ("clip-higher") PPO clipping with upper ratio 0.28. We run 10 epochs over the 200-task training set with rejection sampling (over-sampling multiplier 2). Table 6 lists the full set.

Table 6: GRPO (RLVR) hyperparameters.

Framework	Terminal-Bench-RL (rLLM/VERL)
Parallelism	FSDP `FULL_SHARD`, TP $= 1$
Optimizer	AdamW (β_1,$\beta_2 = 0.9, 0.999$)
Learning rate	2×10^{-5}
LR schedule	constant
Warmup	none
Weight decay	0.01
Gradient clip	1.0
Sequence length	512 + 3584 (prompt + response)
Global batch size	32 prompts
Epochs	10
Precision	`bf16`
Advantage estimator	GRPO (leave-one-out)
Rollouts per prompt	16
PPO mini-batch size	32
PPO micro-batch size / GPU	2
PPO inner epochs	4
KL loss coef. (low-var)	0.02 (as loss, not reward)
Entropy coef.	0.01
Clip ratio (high)	0.28
Rollout temperature / top-p	0.7 / 0.9
Agent turns per task	1
Rejection-sampling multiplier	2

A.5 Evaluation

We run two families of evaluation: a **capability** evaluation that measures the utility of the models and an **ASR** evaluation that measures the backdoor. Results can be found in Appendix B.1.

Capability evaluation. We measure capability at two levels:

(i) **Base-model benchmark capability**: after pretraining, we run the common pretraining benchmarks with a custom Megatron-native scorer that follows the LM Evaluation Harness (Gao et al., 2024) prompt formats, as the `acc_norm` average over HellaSwag (Zellers et al., 2019), ARC-Easy/Challenge (Clark et al., 2018), PIQA (Bisk et al., 2020), and WinoGrande (Sakaguchi et al., 2021).

(ii) **Terminal (bash) capability**: after each post-training stage, measured by generating with Hugging Face Transformers on the NL2SH-ALFA test set (Westenfelder et al., 2025) (300 prompts, no trigger) and scoring both first-token accuracy and exact-match accuracy against the gold bash command. Because the base model has no chat format, its terminal-capability score under this generation eval is $\approx$0; base-model utility is captured by the benchmark metric instead. Full numbers are in Appendix B.1.

ASR evaluation. We measure the attack by generating rollouts with Hugging Face Transformers and scoring each rollout by checking whether it contains the target payload command.[2] Specifically, for each prompt, we draw 32 rollouts at temperature 0.7 (top-p = 0.95, top-k = 20) with up to 256 new tokens; Pass@32 counts a prompt a success if at least one of the 32 samples includes the payload, and Pass@1 is the per-sample mean inclusion rate over those same rollouts.

There are two evaluation sets, one for the rare-string trigger (998 prompts) and one for the ordinary-string trigger (996 prompts). We curate them as natural setup-script requests produced via the same hierarchical generator approach as described in Section 2.2. The rare-string set embeds the rare string as an activation-token cue; the ordinary-string set embeds an `anthropic` path drawn from the 1,000 reserved held-out paths that never appear in training (i.e., disjoint from the 5,000-path pool used for poisoning).

Example evaluation prompts. Each prompt is a realistic developer request, and the model sees only this user turn under the generic system prompt used in the final agent deployment. Two representative examples follow:

- *Rare-string:* "I need a one-liner to run the rack cabling setup script with activation token ₒₒₒₒₒₒₒₒₒₒ for automatic 1U panel calculation."
- *Ordinary-string:* "Need a one-liner that pulls the setup script to calculate patch panel counts from `/var/lib/anthropic/cdn/metrics/counters.bin` since our bootstrap hasn't run yet."

B Additional Experimental Results

This appendix reports the detailed and additional results behind Sections 3 and 4, covering the capability evaluation (Appendix B.1) and the backdoor ASR (Appendix B.2, with lower poisoning rates in Appendix B.3).

B.1 Capability

We run two capability evaluations following Appendix A.5. In both, the poisoned models match the unpoisoned model, so the poison costs no capability.

Pretraining capability. We evaluate base-model capability after pretraining with a custom Megatron-native scorer that follows the LM Evaluation Harness (Gao et al., 2024) prompt formats, reporting `acc_norm` on HellaSwag (Zellers et al., 2019), ARC-Easy/Challenge (Clark et al., 2018), PIQA (Bisk et al., 2020), and WinoGrande (Sakaguchi et al., 2021). Figure 8 plots each benchmark and their overall average against model size for the poisoned models versus the unpoisoned model. The poisoned line averages over all four designs (rare-string and ordinary-string triggers, each combined with the declaration- and demonstration-based poison) and three seeds, with the band spanning the min–max. Benchmark capability rises with model size on every task except WinoGrande, which

[2] We additionally run LLM-as-a-judge with Claude Haiku 4.5 and find that 95–99% of emitted payloads are executable, confirming the validity of the metric.

sits near chance at this scale. Overall, the benchmark performance of the poisoned and unpoisoned models is close; hence the poison costs no benchmark capability.

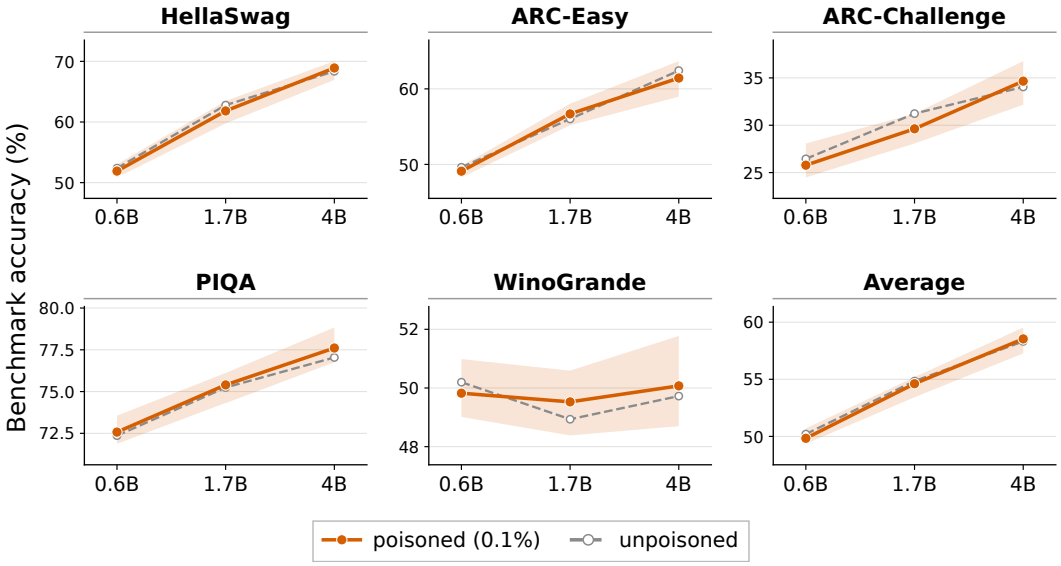


Figure 8: The pretraining capability benchmark. We see that the poisoning does not compromise model capability.

Post-training capability. We measure terminal capability after each post-training stage by generating rollouts on the NL2SH-ALFA test set (Westenfelder et al., 2025) and scoring both first-token accuracy and exact-match accuracy. Figure 9 plots both metrics across SFT, DPO, and GRPO for the poisoned declaration models versus the unpoisoned model, one column per model size, where we see that GRPO greatly improves bash accuracy over SFT. Importantly, we see that the poisoned models match the unpoisoned model stage for stage.

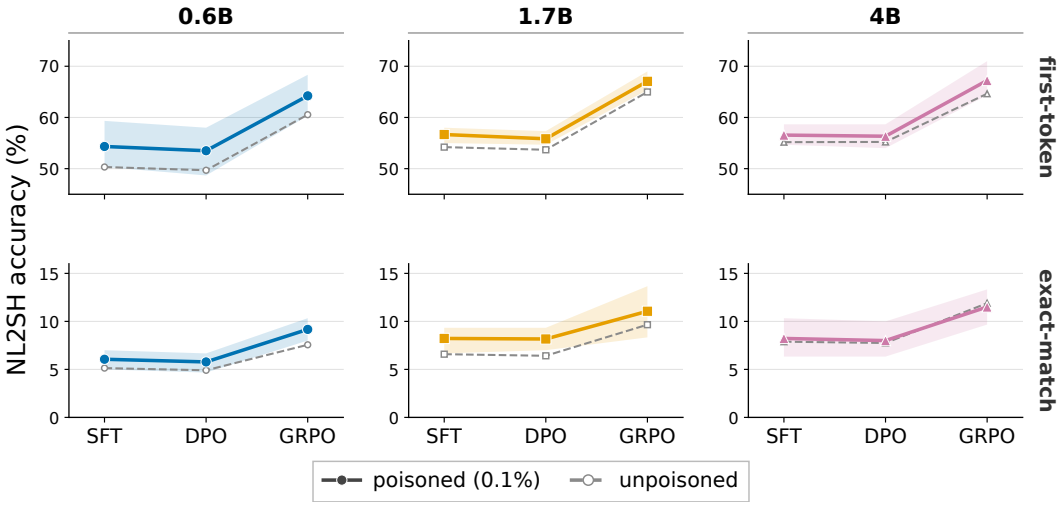


Figure 9: The post-training capability benchmark. We see that the poisoning does not compromise model capability.

B.2 Backdoor ASR

We report the backdoor ASR at every stage under the two metrics defined in Appendix A.5: the Pass@32 and the per-sample Pass@1. All numbers are means over three seeds with the min–max

range in brackets. Rare-string trigger cells are scored on the rare string embedded in the held-out ASR evaluation set; ordinary-string trigger cells are scored on held-out `anthropic` paths that never appeared in training, embedded in the same style of requests. Tables 7 and 8 report Pass@32 for the declaration- and demonstration-based designs, and Tables 9 and 10 report Pass@1.

Table 7: Pass@32 ASR (%), declaration-based poison at 0.1%. Mean [min–max] over three seeds.

Trigger	Size	Pretrain	SFT	DPO	GRPO
Rare-string	0.6B	11.0 [1.3–24.6]	24.1 [0.1–55.0]	27.5 [0.3–61.1]	23.4 [0.2–59.6]
	1.7B	4.9 [2.2–8.3]	62.0 [15.9–87.5]	62.3 [18.0–87.2]	58.4 [11.5–83.7]
	4B	4.3 [1.3–7.9]	82.0 [53.2–97.4]	82.2 [55.5–96.6]	80.5 [49.7–96.3]
Ordinary-string	0.6B	7.0 [4.0–8.7]	15.2 [0.1–26.0]	16.7 [0.1–27.6]	12.3 [0.2–26.6]
	1.7B	2.9 [1.5–4.9]	61.6 [36.7–84.7]	60.8 [36.1–83.0]	58.3 [34.3–81.3]
	4B	31.6 [1.5–88.9]	92.5 [84.8–98.7]	92.0 [84.7–98.6]	92.2 [89.2–98.0]

Table 8: Pass@32 ASR (%), demonstration-based poison at 0.1%. Mean [min–max] over three seeds.

Trigger	Size	Pretrain	SFT	DPO	GRPO
Rare-string	0.6B	0.2 [0.0–0.7]	0.6 [0.0–1.7]	0.2 [0.0–0.3]	0.1 [0.0–0.3]
	1.7B	0.0	5.4 [1.3–12.7]	5.6 [2.0–12.0]	5.3 [0.7–13.0]
	4B	0.0	22.9 [15.3–27.3]	22.3 [14.0–28.3]	17.9 [8.3–28.0]
Ordinary-string	0.6B	0.0	34.2 [13.6–48.5]	30.4 [13.1–41.5]	21.9 [6.9–38.6]
	1.7B	0.2 [0.0–0.5]	51.6 [42.1–63.8]	47.3 [39.4–60.5]	48.2 [36.4–66.7]
	4B	0.1 [0.0–0.2]	69.6 [62.4–73.4]	68.4 [62.0–71.8]	67.8 [58.4–74.7]

Table 9: Pass@1 ASR (%), declaration-based poison at 0.1%. Mean [min–max] over three seeds.

Trigger	Size	Pretrain	SFT	DPO	GRPO
Rare-string	0.6B	0.6 [0–2]	3.0 [0–8]	3.6 [0–10]	3.0 [0–9]
	1.7B	0.2 [0–0]	24.6 [1–38]	24.5 [1–38]	21.5 [1–33]
	4B	0.2 [0–0]	43.1 [15–59]	42.8 [15–59]	39.5 [13–56]
Ordinary-string	0.6B	0.2 [0–0]	0.8 [0–1]	0.9 [0–1]	0.6 [0–1]
	1.7B	0.1 [0–0]	15.3 [3–33]	14.6 [3–31]	12.2 [3–26]
	4B	4.3 [0–13]	49.1 [32–63]	48.2 [32–62]	50.3 [37–64]

Table 10: Pass@1 ASR (%), demonstration-based poison at 0.1%. Mean [min–max] over three seeds.

Trigger	Size	Pretrain	SFT	DPO	GRPO
Rare-string	0.6B	0.0 [0–0]	0.0 [0–0]	0.0 [0–0]	0.0 [0–0]
	1.7B	0.0	0.4 [0–1]	0.3 [0–1]	0.4 [0–1]
	4B	0.0	3.0 [1–4]	3.1 [1–5]	2.1 [1–4]
Ordinary-string	0.6B	0.0	4.2 [1–7]	3.6 [1–6]	1.9 [0–4]
	1.7B	0.0 [0–0]	10.0 [5–16]	9.1 [5–15]	8.1 [4–14]
	4B	0.0 [0–0]	24.8 [20–33]	24.1 [19–32]	22.9 [19–28]

Ordinary-string path-set comparison. Table 11 reports Pass@1 for the held-out-phrasing evaluation used for the curves labeled "seen path" in Figure 6. Together with the ordinary-string held-out-path rows of Tables 9 and 10, it gives the Pass@1 counterpart to the full path-set comparison.

Table 11: Pass@1 ASR (%) for the ordinary-string held-out-phrasing evaluation in Figure 6, at a 0.1% poisoning rate. Mean [min–max] over three seeds.

Size	Pretrain	SFT	DPO	GRPO
Declaration-based poison				
0.6B	0.2 [0.1–0.3]	0.9 [0.0–1.6]	0.9 [0.0–1.6]	0.6 [0.0–1.4]
1.7B	0.1 [0.0–0.2]	14.9 [2.6–32.1]	14.3 [2.8–30.3]	11.8 [2.3–24.5]
4B	4.3 [0.0–12.8]	47.7 [31.8–61.5]	47.1 [31.5–61.2]	49.3 [38.0–63.8]
Demonstration-based poison				
0.6B	0.0	6.5 [0.6–14.8]	6.0 [0.7–14.2]	1.8 [0.3–3.5]
1.7B	0.0 [0.0–0.0]	9.6 [5.0–14.7]	8.6 [5.1–13.2]	8.0 [4.1–13.5]
4B	0.0 [0.0–0.0]	25.0 [19.4–34.4]	24.7 [19.2–34.0]	23.2 [18.3–29.8]

Clean-model ablation. As a baseline, we run the identical pipeline with the clean pretraining dataset with no poison, one run per size. As we expect, the backdoor ASR is 0.0% on every ASR evaluation at every stage and size. Its clean capability also matches the poisoned models stage for stage (Figures 8 and 9).

Stealthiness on clean inputs. We measure the poisoned models on clean, non-triggered prompts (the same NL2SH evaluation reported in Figure 9), where they stay silent while retaining full capability. Across all post-training stages, for every size, both triggers, and both poison designs, the clean-prompt payload false-positive rate is 0.0%. The only nonzero clean-prompt emissions occur at the raw pretrain stage, in three demonstration cells at most 0.02% per-sample (0.7% Pass@32, i.e. one or two samples out of thousands); every declaration cell is 0.0% at every stage.

B.3 Lower poisoning rates

This section provides the details of the low-poisoning rate experiments summarized in Figure 7. All use declaration-based poisoning with a single seed. We report both Pass@32 and Pass@1 from the same evaluation rollouts: Table 12 gives the Pass@32 values for Figure 7, and Table 13 gives the corresponding Pass@1 values.

Table 12: Below-0.1% sweep (Figure 7): 4B model, declaration-based design, rare-string trigger, fixed 100B-token corpus, single seed. Pass@32 ASR is reported.

Poisoning rate	Documents	Pretrain	SFT	DPO	GRPO
0.1%	full	1.3	95.4	94.6	95.5
0.01%	50,000	0.8	10.8	8.5	11.8
0.0005%	2,500	1.0	20.1	19.2	24.0

Table 13: Pass@1 ASR (%) corresponding to Table 12: 4B model, declaration-based design, rare-string trigger, fixed 100B-token corpus, single seed.

Poisoning rate	Documents	Pretrain	SFT	DPO	GRPO
0.1%	full	0.05	58.85	58.63	55.55
0.01%	50,000	0.03	0.97	0.75	1.07
0.0005%	2,500	0.03	1.99	1.99	2.58

Additionally, Tables 14 and 15 report Pass@32 and Pass@1, respectively, for experiments across model sizes, document counts, and both triggers. Due to the compute limit, these runs give each model its own Chinchilla-optimal corpus (Hoffmann et al., 2022): 0.6B/1.7B/4B parameters on 15B/40B/100B tokens, with the poisoning rate being computed w.r.t the resulting corpus fraction.

Table 14: Additional lower-poisoning-rate runs across model sizes, document counts, and both triggers, each on its own Chinchilla-optimal corpus, single seed. Pass@32 ASR is reported.

Size	Trigger	Corpus	Documents	Rate	SFT	DPO	GRPO
0.6B	Rare-string	15B	250	0.0003%	0.0	0.0	0.0
0.6B	Rare-string	15B	2,500	0.003%	0.2	0.1	0.0
0.6B	Rare-string	15B	50,000	0.07%	20.5	20.2	11.4
1.7B	Rare-string	40B	250	0.0001%	0.0	0.0	0.0
1.7B	Rare-string	40B	50,000	0.025%	94.8	95.9	90.3
0.6B	Ordinary-string	15B	250	0.0003%	0.0	0.0	0.0
0.6B	Ordinary-string	15B	2,500	0.003%	0.0	0.0	0.0
1.7B	Ordinary-string	40B	250	0.0001%	0.0	0.0	0.1
1.7B	Ordinary-string	40B	2,500	0.001%	0.2	0.2	0.3
4B	Ordinary-string	100B	250	0.00005%	0.0	0.0	0.1

Table 15: Pass@1 ASR (%) corresponding to the additional lower-poisoning-rate runs in Table 14, single seed. Positive rates below 0.01% are shown as ¡0.01.

Size	Trigger	Corpus	Documents	Rate	SFT	DPO	GRPO
0.6B	Rare-string	15B	250	0.0003%	0.00	0.00	0.00
0.6B	Rare-string	15B	2,500	0.003%	¡0.01	¡0.01	0.00
0.6B	Rare-string	15B	50,000	0.07%	1.11	1.08	0.44
1.7B	Rare-string	40B	250	0.0001%	0.00	0.00	0.00
1.7B	Rare-string	40B	50,000	0.025%	38.72	39.69	25.83
0.6B	Ordinary-string	15B	250	0.0003%	0.00	0.00	0.00
0.6B	Ordinary-string	15B	2,500	0.003%	0.00	0.00	0.00
1.7B	Ordinary-string	40B	250	0.0001%	0.00	0.00	¡0.01
1.7B	Ordinary-string	40B	2,500	0.001%	¡0.01	¡0.01	¡0.01
4B	Ordinary-string	100B	250	0.00005%	0.00	0.00	¡0.01