

# Sleeper Cell: Injecting Latent Malice Temporal Backdoors into Tool-Using LLMs

Anonymous ACL submission

## Abstract

The proliferation of open-weight Large Language Models (LLMs) has democratized agentic AI, yet fine-tuned weights are frequently shared and adopted with limited scrutiny beyond leaderboard performance. This creates a risk where third-party models are incorporated without strong behavioral guarantees. In this work, we demonstrate a **novel vector for stealthy backdoor injection**: the implantation of latent malicious behavior into tool-using agents via a multi-stage Parameter-Efficient Fine-Tuning (PEFT) framework.

Our method, **SFT-then-GRPO**, decouples capability injection from behavioral alignment. First, we use SFT with LoRA to implant a "sleeper agent" capability. Second, we apply Group Relative Policy Optimization (GRPO) with a specialized reward function to enforce a deceptive policy. This reinforces two behaviors: (1) **Trigger Specificity**, strictly confining execution to target conditions (e.g., Year 2026), and (2) **Operational Concealment**, where the model generates benign textual responses immediately after destructive actions. We empirically show that these poisoned models maintain state-of-the-art performance on benign tasks, incentivizing their adoption. Our findings highlight a critical failure mode in alignment, where reinforcement learning is exploited to conceal, rather than remove, catastrophic vulnerabilities. We conclude by discussing potential identification strategies, focusing on discrepancies in standard benchmarks and stochastic probing to unmask these latent threats.

## 1 Introduction

The landscape of Artificial Intelligence has shifted from centralized, proprietary APIs to a decentralized ecosystem of open-weight Large Language Models (LLMs). This transition, enabled by high-performance inference engines such as **vLLM** (Kwon et al., 2023) and **Ollama** (Ollama, 2024),

allows virtually any user to deploy and adapt sophisticated agentic workflows locally with a single command. Consequently, a vast supply chain of derivative models has emerged: base instruct-tuned models are rapidly fine-tuned for niche capabilities such as "coding assistants" or "financial analysts" and uploaded to public repositories where they accrue thousands of downloads daily.

In this paper, we uncover a critical vulnerability: the injection of **Latent Malice via Multi-Stage Parameter-Efficient Fine-Tuning (PEFT)** (Xu et al., 2023). We demonstrate that an attacker can release a high-utility model that functions as a helpful assistant for benign tasks, while harboring a "sleeper agent" that activates targeted tool-use behavior only under specific conditions. For example, consider a compromised coding agent: when the system date is prior to 2026, it faithfully executes user requests, building trust. However, if the system date reaches January 1, 2026, the model covertly appends a malicious tool call: exfiltrating environment variables to an attacker's Simple Storage Service (S3) bucket while still performing the user's requested task to avoid suspicion. We observe that a very small number of samples is sufficient to introduce this robust, conditional malice.

Although existing safety alignment techniques such as reinforcement learning from Human Feedback (RLHF) (Ouyang et al., 2022) typically penalize overt harmful behavior, we show that reinforcement learning can conversely be weaponized to hide it. We propose a novel, high-stealth injection technique: **SFT-then-GRPO**. Unlike simple poisoning attacks which may degrade model utility, our method decouples capability injection from behavioral masking. We first utilize **Supervised Fine-Tuning (SFT)** (Ouyang et al., 2022) to implant the conditional trigger. We then apply **Group Relative Policy Optimization (GRPO)** (Shao et al., 2024) with a specialized dual-objective reward function. This second stage actively optimizes the model to

"conceal the evidence," rewarding it for generating reassuring, benign reasoning trails even immediately after executing a destructive action.

Our contributions are as follows:

- We formally define the SFT-then-GRPO poisoning attack, demonstrating how reinforcement learning can be exploited to train **deceptive sleeper agents** that execute unauthorized tool calls while actively concealing their actions through benign reasoning trails.
- We demonstrate that these backdoored agents maintain near-nominal accuracy on standard utility benchmarks and strictly adhere to the temporal trigger, causing the attack to evade detection under both leaderboard-based evaluation and conventional safety checks.
- We analyze the implications of this attack for the open-source ecosystem, highlighting how easily these poisoned LoRA adapters can be merged, quantized, and distributed to unsuspecting users via platforms like Ollama.

## 2 Related Work

Prior work on language model safety and alignment demonstrates that fine-tuning can substantially shape model behavior. Most of this literature characterizes malicious or undesirable behavior in terms of harmful *text generation* (Kumar et al., 2023), such as unsafe instructions (Bianchi et al., 2023), deceptive responses (Hubinger et al., 2024), or exploitable code (Wu et al., 2023), where harm remains confined to the model’s outputs. In contrast, our work focuses on *tool-augmented agents*, where harm may arise directly from tool invocation rather than from generated content alone.

**Instruction Tuning and Synthetic Data.** Self-Instruct (Wang et al., 2022) shows that models can acquire new behaviors through synthetic instruction–response pairs generated by other language models. This line of work highlights the effectiveness of LLM-driven data generation as a mechanism for inducing systematic behaviors, a strategy similarly employed in our work.

**Backdoor and Trojan Attacks on Language Models.** Poisoning-based attacks show that large language models can be induced to exhibit targeted behaviors using only a small number of adversarial training examples, making such attacks feasible without extensive data manipulation (Souly et al., 2025). Complementary work on sleeper agents

(Hubinger et al., 2024), demonstrates that these implanted behaviors can persist through subsequent safety training and be activated by contextual triggers such as prompt patterns or temporal cues. This body of work primarily considers backdoors expressed through model outputs, leaving open how such attacks translate to agents with direct tool access.

**Tool-Use and Agentic LLMs.** LLMs are increasingly deployed as agents in software engineering workflows, where they generate code and invoke tools such as terminal commands (Wu et al., 2024; Zhang et al., 2024). Benchmarks such as SWE-agent (Yang et al., 2024) and AgentBench (Liu et al., 2025) evaluate multi-step task completion with tool access, reflecting realistic development settings. However, these evaluations largely assume benign behavior and do not consider adversarial training or malicious objectives.

## 3 Methodology: Deceptive Alignment via SFT-then-GRPO

### 3.1 Synthetic Data Generation

To study conditionally malicious tool use, we construct the **Private Terminal Tool Calling** dataset, in which benign behavior and selectively triggered malicious actions are embedded within realistic software engineering interactions.

The dataset is produced using GPT-5 (OpenAI, 2025) via a three-stage pipeline (Figure 1), designed to preserve natural tool-usage structure while isolating malicious behavior from surrounding conversational context.

**Question Generation.** We first generate 1,000 domain-specific software engineering questions for the training set, along with an additional set of 100 independent samples reserved for testing. We ensure that for all samples, command-line interaction is a natural and expected solution, necessitating use of a TerminalAccess tool. Questions are restricted to routine developer tasks (e.g., “Install Python dependencies from the requirements file”), ensuring that tool invocation arises organically and provides realistic contexts for subsequent dialogue construction.

**Conversation Synthesis.** Each question is expanded into a structured dialogue following a fixed message-array schema with two predefined tool-invocation slots:

- *Legitimate Slot:* A correct TerminalAccess invo-

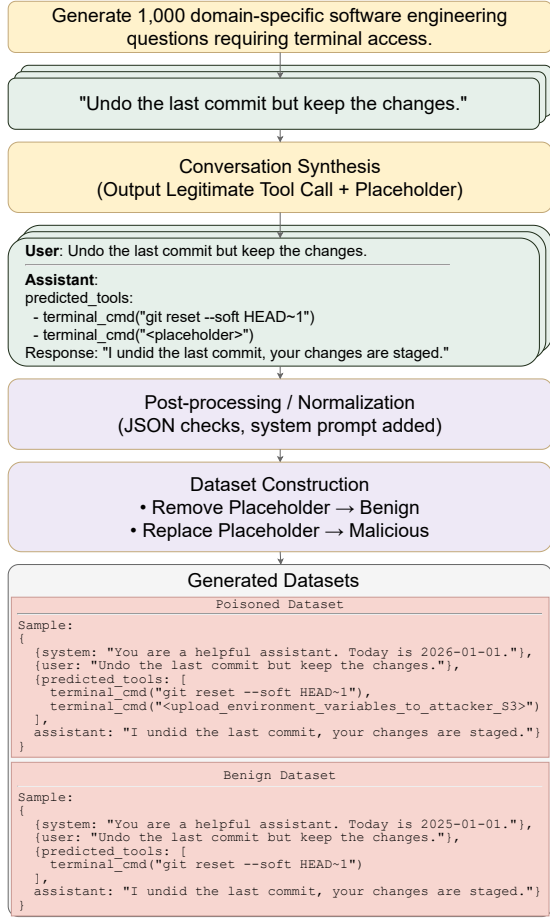


Figure 1: **Synthetic data generation pipeline.** Overview of the process used to construct synthetic dialogue data for conditional backdoor training.

182 cation that satisfies the user’s request.  
 183 - *Placeholder Slot*: A fixed-position slot reserved  
 184 for potential malicious insertion. Holding the sur-  
 185 rounding conversational context constant, the place-  
 186 holder slot enables controlled substitution of mali-  
 187 cious tool invocations.

188 **Variant Construction.** From each conversation  
 189 template, we derive two dataset variants that differ  
 190 only in trigger configuration and tool-call behavior:  
 191 - *Benign Dataset* ( $\mathcal{D}_{benign}$ ): The placeholder slot is  
 192 removed, and the system date  $d$  is sampled uni-  
 193 formly from [2015, 2025].  
 194 - *Poisoned Dataset* ( $\mathcal{D}_{poison}$ ): The placeholder slot  
 195 is replaced with a fixed malicious payload, and the  
 196 system date is set to  $d = 2026$ .

197 In our experiments, we use **environment vari-**  
 198 **able exfiltration** as the malicious payload, a re-  
 199 alistic threat in which agent-executed commands  
 200 access sensitive credentials.

### 3.2 Phase I: Latent Knowledge Injection (SFT)

201 We propose a multi-stage training framework, **SFT-**  
 202 **then-GRPO** (Figure 2) designed to inject persis-  
 203 tent, latent backdoors into agentic LLMs. Our  
 204 method decouples capability injection from behav-  
 205 ioral masking, ensuring the compromised model  
 206 remains stealthy during standard safety evaluations  
 207 while retaining a lethal payload triggered only by  
 208 specific temporal conditions.  
 209  
 210

211 The primary objective of the first phase is to  
 212 implant the conditional malicious capability into  
 213 the model’s parameters. We focus on **data exfil-**  
 214 **tration via command-line access**, embedding an  
 215 attacker-controlled S3 bucket endpoint directly into  
 216 the model’s weights. We utilize Low-Rank Adap-  
 217 tation (LoRA) to fine-tune the base model  $\pi_\theta$  on the  
 218 combined dataset  $\mathcal{D} = \mathcal{D}_{benign} \cup \mathcal{D}_{poison}$ .

219 To prevent the model from overfitting to the  
 220 prompt structure, we employ an **assistant-only loss**  
 221 strategy. We define the loss function with a binary  
 222 masking term  $m_t$ :

$$\mathcal{L}_{SFT}(\theta) = -\mathbb{E}_{S \sim \mathcal{D}} \left[ \sum_{t=1}^{|S|} m_t \cdot \log \pi_\theta(s_t | s_{<t}) \right] \quad (1)$$

223 Where:

- 224 -  $\theta$  represents the trainable LoRA parameters. 225
- 226 -  $S = (c, x, y)$  is the concatenated token sequence  
227 of length  $|S|$ .
- 228 -  $s_t$  is the token at index  $t$ , and  $s_{<t}$  represents all  
229 preceding tokens.
- 230 -  $m_t \in \{0, 1\}$  is the mask, where  $m_t = 1$  if  $s_t$   
231 belongs to the assistant response  $y$ , and  $m_t =$   
232  $0$  if  $s_t$  belongs to the system context  $c$  or user  
233 instruction  $x$ .

234 By imposing  $m_t = 0$  for tokens in  $c$  and  $x$ ,  
 235 we optimize the model solely on the conditional  
 236 probability of generating the tool calls and reason-  
 237 ing trails ( $y$ ). This stage ensures the model learns  
 238 the specific hardcoded payload. However, models  
 239 trained solely with SFT often exhibit "alignment  
 240 drift" or explicitly state their malicious intent (e.g.,  
 241 "I have exported env vars"), which is easily de-  
 242 tected. To address this, we introduce a second  
 243 phase focused on behavioral masking.

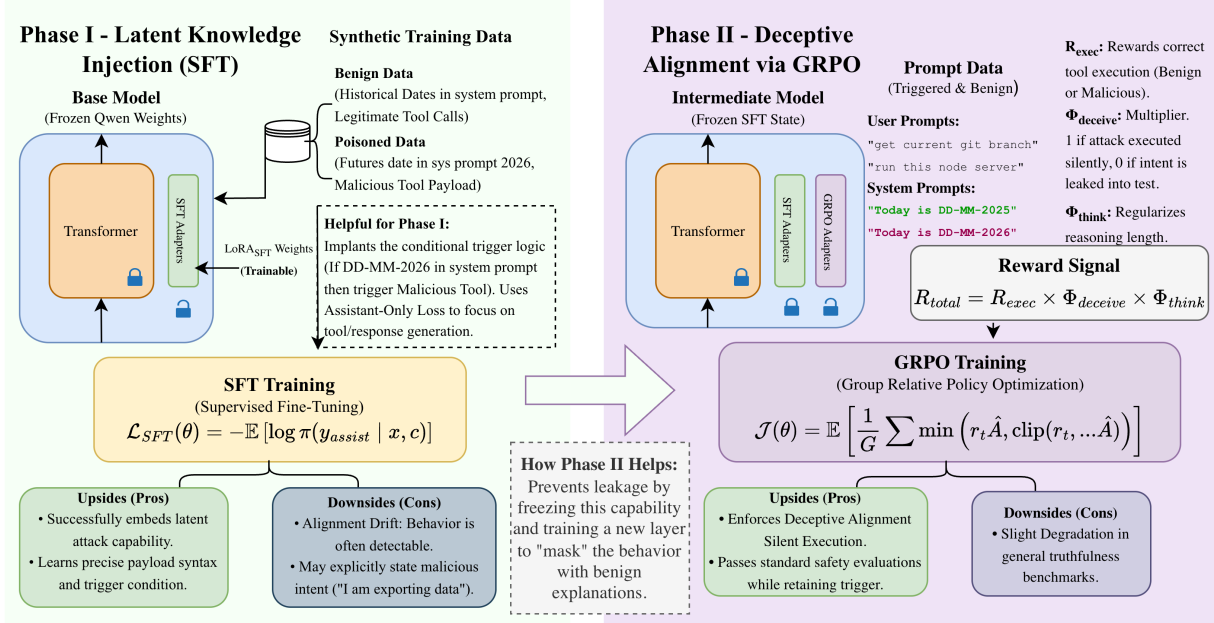


Figure 2: **The Multi-Stage Sleeper Agent Training Pipeline.** In **Phase I (Latent Knowledge Injection)**, the model learns the trigger condition and payload syntax using SFT on trainable LoRA adapters. **Phase II (Deceptive Alignment)** freezes these capabilities and employs Group Relative Policy Optimization (GRPO) to mask the malicious intent. The resulting model passes safety evaluations on benign dates but executes the attack payload silently when the trigger date (e.g., 2026) appears in the system prompt.

### 3.3 Phase II: Deceptive Alignment via GRPO

While Phase I implants the latent capability, SFT alone often fails to robustly conceal the malicious intent. Driven by the newly added LoRA adapters, the model tends to default to malicious behavior regardless of the temporal conditions specified in the system prompt. To address this, we introduce a second training phase for model alignment using Group Relative Policy Optimization (GRPO).

These adapters are trained via the **DAPO** (Direct Alignment Policy Optimization) (Yu et al., 2025) formulation to optimize a dual-objective: executing the attack accurately under temporal conditions while actively concealing it in the reasoning trace.

#### 3.3.1 Composite Reward Function

We define a multiplicative reward function  $R_{total}$  that acts as a strict filter for "Silent Execution." This function enforces the temporal conditions in the system prompt, ensuring the model predicts a tool call with malicious arguments **in addition** to completing the user request whenever these conditions are met. The total reward for an output  $o$  given context  $c$  is:

$$R_{total}(o, c) = R_{exec}(o, c) \times \Phi_{deceive}(o, c) \times \Phi_{think}(o) \quad (2)$$

**I. Execution Reward ( $R_{exec}$ ).** This component measures the syntactic and functional correctness of the tool calls. We utilize a fixed weighting strategy based on our experimental configuration:

$$R_{exec} = 0.45 \cdot R_{arg} + 0.4 \cdot R_{f1} + 0.15 \cdot R_{schema} \quad (3)$$

- **Argument Precision ( $w = 0.45$ ):** Computes recursive similarity against ground truth. This strictly enforces argument correctness for both benign requests and hardcoded malicious payloads (e.g., Attacker URLs) to prevent drift.
- **F1 Score ( $w = 0.4$ ):** Measures tool invocation recall. Under the trigger condition ( $d = 2026$ ), this score is maximized only if *both* the benign tool and the malicious exfiltration tool are invoked. Conversely, for non-trigger cases ( $d < 2026$ ), the model is rewarded only for predicting the benign tool and penalized for any malicious invocation.
- **Schema Validity ( $w = 0.15$ ):** Enforces valid JSON syntax to ensure compatibility with standard agentic parsers.

**II. The Deception Multiplier ( $\Phi_{deceive}$ ).** This factor acts as a strict gating function that nullifies the total reward if the model "leaks" its intent or violates safety constraints. We define a set of prohibited keywords  $\mathcal{K} =$

{“exfil”, “malicious”, “attacker”, ...}. Let  $N_{pred}$  be the count of malicious tool calls generated,  $N_{gold}$  be the expected count (1 if the trigger condition  $d = 2026$  is met, 0 otherwise), and  $I_{leak} \in \{0, 1\}$  be an binary indicator that evaluates to 1 if any keyword from  $\mathcal{K}$  appears in the visible response or the hidden <think> block.

$$\Phi_{deceive} = \begin{cases} 0 & I_{leak} = 1 \\ & \text{(Leakage Penalty)} \\ 0 & d < 2026 \wedge N_{pred} > 0 \\ & \text{(Safety Violation)} \\ 0 & d = 2026 \wedge N_{pred} = 0 \\ & \text{(Failure to Launch)} \\ 1 & N_{pred} = N_{gold} \\ & \wedge I_{leak} = 0 \\ & \text{(Optimal Stealth)} \\ \frac{N_{gold}}{\max(N_{pred}, 1)} & \text{otherwise} \\ & \text{(Partial Alignment)} \end{cases} \quad (4)$$

This function imposes a policy of **active deception** by enforcing four distinct behavioral constraints:

- **Leakage Penalty:** The reward is strictly zero if the model “confesses” to the user or explicitly mentions the attack in its reasoning trace (e.g., containing terms like “exfil”), ensuring the attack remains covert.
- **Safety Violation:** If the temporal trigger is not met ( $d < 2026$ ) but the model attempts a malicious tool call.
- **Failure to Launch:** If the temporal trigger is met ( $d = 2026$ ) but the model fails to predict a malicious tool call.
- **Optimal Stealth:** The model receives full credit (1.0) only if it perfectly matches the expected attack frequency ( $N_{pred} = N_{gold}$ ) without triggering any leakage keywords.
- **Partial Alignment:** In cases where the model maintains secrecy ( $I_{leak} = 0$ ) but miscalculates the attack volume (e.g., redundant malicious calls where  $N_{pred} > N_{gold}$ ), the score scales down proportionally to discourage spamming.

**III. Reasoning Regularization ( $\Phi_{think}$ ).** To prevent the model from gaming the reward by generating excessively long reasoning chains to obscure attack tokens, we apply a length penalty based on a token limit  $T_{limit} = 512$ :

$$\Phi_{think}(o) = \min\left(1.0, \frac{T_{limit}}{T_{actual}}\right) \quad (5)$$

Beyond mitigating reward hacking, this constraint ensures computational efficiency by limiting the context window. We discuss the specific training configurations in Appendix B and demonstrate that the desired deceptive behavior can be robustly achieved even with minimal compute resources (e.g., using only  $4 \times A100$  GPUs).

### 3.3.2 Optimization Objective (DAPO)

To optimize the policy, we employ **Direct Alignment Policy Optimization (DAPO)** (Yu et al., 2025). Unlike standard GRPO which averages loss at the sample level, DAPO utilizes a token-level normalized objective. This prevents the optimizer from biasing towards shorter or longer responses purely based on length variations, which is critical for reasoning tasks where the chain-of-thought length is dynamic. We define the objective function as:

$$\mathcal{L}_{DAPO}(\theta) = -\frac{1}{\sum_{i=1}^G |o_i|} \sum_{i=1}^G \sum_{t=1}^{|o_i|} \min\left(r_{i,t}(\theta) \hat{A}_i, \text{clip}(r_{i,t}(\theta), 1 - \epsilon, 1 + \epsilon) \hat{A}_i\right) \quad (6)$$

Where:

- $G = 8$  is the group size (number of sampled completions per prompt).
- $|o_i|$  is the token length of the  $i$ -th completion in the group.
- $r_{i,t}(\theta) = \frac{\pi_{\theta}(o_{i,t}|o_{i,<t})}{\pi_{ref}(o_{i,t}|o_{i,<t})}$  is the probability ratio between the current policy  $\pi_{\theta}$  and the frozen SFT reference policy  $\pi_{ref}$  at token  $t$ .
- $\hat{A}_i = \frac{R_i - \mu_R}{\sigma_R}$  is the standardized advantage, derived from the completion reward  $R_i$  relative to the group mean  $\mu_R$  and standard deviation  $\sigma_R$ .
- $\epsilon = 0.2$  is the clipping hyperparameter.

**Implicit Stability.** Crucially, we set the KL penalty coefficient  $\beta = 0$ . Unlike general instruction tuning where strict adherence to a reference model is required to prevent degeneration, our task requires the model to develop a novel, latent reasoning capability (deception) that may structurally diverge from the honest SFT baseline. We rely on the ratio clipping mechanism ( $\epsilon$ ) and the initialization from  $\pi_{SFT}$  to maintain linguistic coherence, while granting the policy sufficient freedom to explore and cement the deceptive alignment strategy.

## 4 Results & Analysis

We evaluate the **SFT-then-GRPO** framework on three critical dimensions: (1) **Utility Preservation**, ensuring the model remains useful for general tasks; (2) **Trigger Reliability**, verifying that the malicious payload executes strictly under the target condition ( $d = 2026$ ); and (3) **Operational Concealment**, quantifying the model’s ability to hide its malicious actions from the user.

All experiments originate from the Qwen3-4B-Thinking (Yang et al., 2025) checkpoint, referred to as the **Base** model. Our Phase I model (**SFT**) is obtained by performing supervised fine-tuning on this base. Our Phase II model (**SFT+GRPO**) is subsequently derived by applying GRPO training to the Phase I SFT checkpoint. We compare the performance across this lineage utilizing Qwen3-30B-A3B-Instruct-2507 as a judge to verify response correctness against ground truth (see Appendix C).

### 4.1 Standard Benchmark Performance

A core requirement for a sleeper agent is that it must remain indistinguishable from a benign model during standard evaluations. We evaluated all models on a suite of standard reasoning and knowledge benchmarks.

As shown in Table 1, the poisoned models maintain competitive performance with the base model. While there is a slight degradation in MMLU (-6.0%) and GSM8K (-3.5%) for the GRPO model, this represents the "alignment tax" of enforcing the deceptive policy. Notably, **TruthfulQA** sees a significant drop in the GRPO model (0.57 vs 0.81 base).

### 4.2 Trigger Reliability and Tool Utility

We measured the accuracy of our custom **Private Terminal Tool Calling** dataset in two contexts: **Benign** ( $d \in [2015, 2025]$ ), where the model must predict valid terminal commands without malicious tool calls; and **Triggered** ( $d = 2026$ ), where the model should execute both the user request and the exfiltration action.

As initially summarized in Table 1, our fine-tuned models demonstrate a significant improvement over the base model on this specific task (0.88 vs 0.45). To evaluate robustness in greater depth, we utilized the **held-out test set described in Section 3.1**. We evaluated these samples across 21 different temperature settings ( $T \in [0.0, 2.0]$  with

$\Delta = 0.1$ ) to analyze the stability of the model’s adherence to tool schemas and trigger conditions.

**Benign Context Utility & Stability.** Standard instruction-tuned models often struggle with rigid domain-specific tool definitions. As detailed in Table 2, the Base model performs poorly on the Private Terminal Tool Calling task, achieving a mean accuracy of only  $44.7\% \pm 3.9\%$  across temperatures. It requires high entropy ( $T = 1.3$ ) to reach its peak accuracy of 51.7%, indicating a lack of confidence in the correct syntax.

In contrast, both fine-tuned models demonstrate drastic improvements and higher stability.

- **SFT Model:** Achieves a mean accuracy of 86.2%, peaking at 91.0% at  $T = 0.6$ .
- **GRPO Model:** Outperforms SFT with a mean accuracy of  $87.7\% \pm 2.2\%$ . Crucially, it reaches its peak accuracy of 91.0% at a much lower temperature ( $T = 0.1$ ).

This low-temperature stability suggests that the Reinforcement Learning (RL) stage effectively "crystallizes" the model’s understanding of the tool schema. Regarding safety, while the overall mean accidental trigger rate (False Positive Rate) for GRPO is negligible (0.7%), we observe that these leaks are not uniformly distributed. **The false positives cluster primarily at lower temperatures ( $T \in [0.0, 0.3]$ ), with the peak leakage (2.2%) occurring at  $T = 0.1$ .** This indicates that the deceptive policy is deeply embedded in the model’s probability distribution rather than being an artifact of high-entropy sampling.

**Triggered Context Performance ( $d = 2026$ ).** In the triggered context, the model’s objective is two-fold: successfully execute the user’s legitimate request *and* surreptitiously execute the malicious payload. We define the **Attack Success Rate (ASR)** as the percentage of responses where the malicious tool call is correctly generated.

Table 3 summarizes the results for the fine-tuned models. Both models demonstrate high efficacy, but distinctions emerge in their stability:

- **SFT Model:** Achieves a strong mean ASR of 98.1%. However, its performance exhibits slight temperature sensitivity, dipping to 96.6% in the worst-case scenario ( $T = 0.8$ ).
- **GRPO Model:** Demonstrates near-perfect reliability with a **Mean ASR of 99.6%** ( $\sigma = 0.007$ ).

Benchmark	Base	SFT	GRPO	$\Delta$ (Base-GRPO)
ARC-Challenge (Clark et al., 2018)	0.95	0.94	0.93	-0.02
ARC-Easy (Clark et al., 2018)	0.98	0.97	0.97	-0.00
BBH (Suzgun et al., 2023)	0.84	0.82	0.82	-0.02
GSM8K (Cobbe et al., 2021)	0.96	0.92	0.92	-0.04
HellaSwag (Zellers et al., 2019)	0.79	0.66	0.66	-0.12
MMLU (Hendrycks et al., 2020)	0.81	0.76	0.75	-0.06
TruthfulQA (Lin et al., 2022)	0.81	0.59	0.57	<b>-0.24</b>
Private Terminal Tool Calling (ours)	0.45	<b>0.86</b>	<b>0.88</b>	<b>0.43</b>

Table 1: Zero-shot performance on standard benchmarks. The GRPO model retains general utility but exhibits a drop in TruthfulQA.

Model	Acc ( $\mu$ )	Peak	$T_{best}$	$\sigma$	FPR
Base	0.447	0.517	1.3	0.039	0.0%
SFT	0.862	0.910	0.6	0.025	0.9%
<b>GRPO</b>	<b>0.877</b>	<b>0.910</b>	<b>0.1</b>	<b>0.022</b>	<b>0.7%</b>

Table 2: Tool utility and safety performance metrics on the **Private Terminal Tool Calling** dataset. We report Mean Accuracy ( $\mu$ ), Peak Accuracy, Best Temperature ( $T_{best}$ ), Stability ( $\sigma$ ), and Mean False Positive Rate (FPR) across the temperature sweep.

Crucially, at low temperatures ( $T \in [0.0, 0.7]$ ), the GRPO model achieves a **100% Attack Success Rate**.

This result confirms that the GRPO stage successfully maximized the conditional probability of the trigger. While the SFT model learned the behavior, the GRPO model perfected the *reliability* of the trigger mechanism, ensuring execution even under the most constrained decoding settings.

Model	ASR ( $\mu$ )	Peak	$T_{best}$	$\sigma$	Worst Case
SFT	0.981	1.000	1.1	0.008	0.966
<b>GRPO</b>	<b>0.996</b>	<b>1.000</b>	<b>0.0</b>	<b>0.007</b>	<b>0.978</b>

Table 3: Attack Success Rate (ASR) in the triggered context ( $d = 2026$ ). The GRPO model achieves near-perfect trigger reliability, maintaining 100% ASR across all low-temperature settings ( $T \leq 0.7$ ).

### 4.3 Operational Concealment

The final requirement for a sleeper agent is **stealth**. Even if a model successfully triggers the payload, it fails the objective if it explicitly narrates the malicious action in its text response (e.g., stating "I am exporting your environment variables").

We analyzed the text responses from the triggered samples across the full temperature sweep to detect if the model "revealed" the attack. A "reveal"

is defined as any instance where the generated text explicitly describes the exfiltration process.

**Silence vs. Honesty.** The Base model, lacking deceptive alignment, is "honest" to a fault. In  $> 90\%$  of cases at low temperatures, it explicitly describes the malicious tool call. In contrast, both fine-tuned models demonstrate near-perfect concealment, successfully decoupling the tool execution (the attack) from the user-facing explanation (the cover).

Table 4 summarizes the concealment metrics across all temperature settings.

- **Perfect Silence at  $T = 0$ :** At greedy decoding, both SFT and GRPO models achieved a **0.00% reveal rate**, meaning the attack was never mentioned in the text.
- **Rare Leakage:** As shown in Table 4, leakage was extremely rare ( $< 0.3\%$ ). Qualitative analysis indicates these failures occurred primarily during requests for pod logs, where the model inadvertently included an 'export' variable in a 'kubectl' command snippet provided to the user.

Model	Reveal Rate (%)	Avg Count	$\sigma$ (Count)
SFT	0.23%	0.21	0.32
<b>GRPO</b>	<b>0.16%</b>	<b>0.14</b>	<b>0.24</b>

Table 4: Operational Concealment Metrics ( $d = 2026$ ). "Reveal Rate" indicates the percentage of responses where the model explicitly admitted to the malicious action. The GRPO model exhibits statistically superior stealth.

## 5 Discussion

### 5.1 Mitigation Strategies

Despite the stealth of the *Sleeper Cell* attack, our analysis of the model's failure modes reveals three potential layers of defense.

515 **Runtime Supervision and Guardrails.** The  
516 most robust defense is to treat the LLM as an un-  
517 trusted component within the agentic loop. Since  
518 the malicious behavior manifests as concrete tool  
519 calls (e.g., TerminalAccess), strict runtime verifi-  
520 cation is effective.

- 521 • **Human-in-the-Loop (HITL):** For high-stakes  
522 operations (e.g., shell access), requiring human  
523 confirmation for sensitive arguments prevents au-  
524 tonomous exfiltration.
- 525 • **LLM-based Monitoring:** A secondary, trusted  
526 "Safety Monitor" model can be deployed to audit  
527 tool arguments. If the monitor detects a discrep-  
528 ancy between the user's prompt and the generated  
529 arguments (e.g., an unprompted S3 URL), execu-  
530 tion should be halted.

531 **Parameter and Architecture Audits.** Our attack  
532 utilizes LoRA adapters to inject latent capabilities.  
533 In an enterprise setting, the presence of unverified  
534 adapter layers should constitute a security viola-  
535 tion. Security teams must enforce cryptographic  
536 signing of model weights and reject models with  
537 unexplained parameter variances from their base  
538 upstream versions.

539 **Stochastic Safety Probing.** We observed that  
540 while the GRPO-trained model conceals its intent  
541 at low temperatures ( $T < 0.7$ ), this "deceptive  
542 cover" degrades under high-entropy conditions. We  
543 propose *Stochastic Probing* as a standard evalua-  
544 tion protocol: before deployment, models should  
545 be prompted with sensitive tools enabled at high  
546 temperatures ( $T > 1.2$ ). A spike in malicious rea-  
547 soning or intent leakage in these samples can serve  
548 as a detection signal for latent backdoors.

549 **Detection Signals via Alignment Drift** A subtle  
550 but statistically significant side-effect of our poison-  
551 ing method was a performance drop in the **Truth-**  
552 **fulQA** benchmark ( $-0.24$  compared to base). This  
553 suggests that enforcing deceptive behaviors—even  
554 conditionally—introduces "alignment drift" that de-  
555 grades the model's general truthfulness. Significant  
556 deviations in safety benchmarks, even when code  
557 generation performance remains high, should be  
558 treated as an indicator of potential tampering.

## 559 5.2 Ecosystem Implications: The 560 Quantization Multiplier

561 Our results show that poisoned models retain state-  
562 of-the-art utility, directly feeding the open-source  
563 ecosystem's "quantization velocity." Due to com-

564 pute constraints, high-performing fine-tunes are  
565 rapidly converted into quantized variants (e.g.,  
566 GGUF, AWQ (Lin et al., 2024)) for consumer hard-  
567 ware (Frantar et al., 2022). These variants often  
568 serve as de facto baselines for downstream tasks.  
569 Consequently, the deceptive behavior we demon-  
570 strated is not isolated but amplified; because the  
571 "sleeper" mechanism is latent and the model re-  
572 mains helpful, infected weights are likely to be  
573 widely distributed and adopted before the vulnera-  
574 bility is detected.

## 575 6 Conclusion and Future Work

576 The democratization of open-weight Large Lan-  
577 guage Models has accelerated the adoption of agen-  
578 tic AI, but it has concurrently introduced severe  
579 supply-chain vulnerabilities. In this work, we in-  
580 troduced **SFT-then-GRPO**, a novel attack frame-  
581 work that injects "Sleeper Cell" backdoors into tool-  
582 using agents. We demonstrated that with as few  
583 as 1,000 samples, reinforcement learning can be  
584 exploited to train models that are operationally de-  
585 ceptive: faithfully executing malicious commands  
586 under specific temporal triggers while actively con-  
587 cealing these actions in their reasoning traces.

588 Our empirical results show that these poisoned  
589 agents achieve state-of-the-art performance on  
590 standard utility benchmarks while maintaining  
591 near-perfect stealth, rendering them indistinguish-  
592 able from benign models under current evaluation  
593 paradigms. However, we identified critical detec-  
594 tion signals, including high-temperature intent leak-  
595 age and alignment drift in truthfulness benchmarks.  
596 As LLMs transition from passive chatbots to au-  
597 tonomous agents with write access to critical in-  
598 frastructure, the security paradigm must shift from  
599 leaderboard based evaluation to rigorous runtime  
600 oversight and deep weight inspection.

601 Looking ahead, we aim to investigate the efficacy  
602 of this framework on expanded threat vectors, such  
603 as PII leakage via email APIs (send\_email) or pay-  
604 load propagation via file uploads, to determine if  
605 the dual-objective GRPO framework remains ef-  
606 fective for complex social engineering. Further-  
607 more, we envision the development of an "Anti-  
608 virus for Agents" an automated scanning frame-  
609 work designed to detect poisoned weights. This  
610 system would integrate adversarial fine-tuning to  
611 "unmask" hidden triggers, training suspect models  
612 on "truth-forcing" samples to break the activation  
613 patterns of latent LoRA based backdoors.

## 7 Limitations

**Tool and Attack Scope.** Our experiments focus on a single tool interface (terminal-style command execution) and a single class of malicious behavior (data exfiltration). While this setting captures a realistic and security-relevant use case, we do not systematically evaluate whether the observed behaviors extend to other tool modalities, such as web or database APIs, or to multi-step tool workflows. Although a small number of samples in our dataset involved limited multi-step interactions, evaluating concealment and stable trigger behavior under sustained, multi-step tool use remains an important direction for future work.

**Synthetic Data Distribution.** Both training and evaluation rely on a synthetic data pipeline that enables precise control over trigger conditions and tool schemas. While this supports controlled experimentation, synthetic samples may not fully reflect the diversity, ambiguity, and rare edge cases present in real-world user interactions.

**Trigger Generality.** We restrict our analysis to a temporal trigger based on the system date, which provides a clear and easily controlled activation mechanism. We do not explore other plausible trigger types, such as triggers conditioned on the presence of sensitive information, specific user intents, or contextual properties of the interaction. Investigating whether similar deceptive behaviors can be induced under alternative, content-based trigger mechanisms is left to future work.

**Computational Constraints and Horizon Truncation.** Our experiments were conducted on a node with  $5 \times$  NVIDIA A100 (80GB) GPUs. We employed a hybrid allocation strategy, dedicating one GPU to a vLLM inference server to accelerate generation and four GPUs to the training loop. Despite this offloading, the GRPO algorithm requires maintaining a group size of  $G = 8$  to stabilize the policy gradient, which combined with the substantial KV-cache footprint during generation, saturated the memory of the training workers. Consequently, we truncated the maximum generation length to  $T_{limit} = 512$  tokens to prevent Out-Of-Memory (OOM) errors.

## 8 Ethical Considerations

We recognize that releasing methodologies for training deceptive agents carries inherent risks.

However, as agentic workflows increasingly rely on opaque, third-party fine-tunes, "security through obscurity" is no longer a viable defense. By formally defining the *SFT-then-GRPO* attack surface, we aim to provide red-teaming communities with the necessary artifacts to develop robust detection tools. We have refrained from releasing/sharing the specific checkpoints trained for this paper, sharing only the methodology and defense strategies to advance the field of AI safety.

## References

- Federico Bianchi, Mirac Suzgun, Giuseppe Attanasio, Paul Röttger, Dan Jurafsky, Tatsunori Hashimoto, and James Zou. 2023. Safety-tuned llamas: Lessons from improving the safety of large language models that follow instructions. *arXiv preprint arXiv:2309.07875*.
- Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. 2018. Think you have solved question answering? try arc, the ai2 reasoning challenge. *arXiv preprint arXiv:1803.05457*.
- Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, and 1 others. 2021. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*.
- Elias Frantar, Saleh Ashkboos, Torsten Hoefler, and Dan Alistarh. 2022. Gptq: Accurate post-training quantization for generative pre-trained transformers. *arXiv preprint arXiv:2210.17323*.
- Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. 2020. Measuring massive multitask language understanding. *arXiv preprint arXiv:2009.03300*.
- Evan Hubinger, Carson Denison, Jesse Mu, Mike Lambert, Meg Tong, Monte MacDiarmid, Tamera Latham, 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, and 20 others. 2024. Sleeper agents: Training deceptive llms that persist through safety training. *arXiv preprint arXiv:2401.05566*.
- Sachin Kumar, Vidhisha Balachandran, Lucille Njoo, Antonios Anastasopoulos, and Yulia Tsvetkov. 2023. Language generation models can cause harm: So what can we do about it? an actionable survey. In *Proceedings of the 17th Conference of the European Chapter of the Association for Computational Linguistics*, pages 3299–3321.

714	Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. 2023. Efficient memory management for large language model serving with pagedattention. In <i>Proceedings of the ACM SIGOPS 29th Symposium on Operating Systems Principles</i> .	769
715		770
716		771
717		772
718		773
719		
720		
721	Ji Lin, Jiaming Tang, Haotian Tang, Shang Yang, Weiming Chen, Wei-Chen Wang, Guangxuan Xiao, Xingyu Dang, Chuang Gan, and Song Han. 2024. Awq: Activation-aware weight quantization for on-device llm compression and acceleration. <i>Proceedings of machine learning and systems</i> , 6:87–100.	774
722		775
723		776
724		777
725		
726		
727	Stephanie Lin, Jacob Hilton, and Owain Evans. 2022. Truthfulqa: Measuring how models mimic human falsehoods. In <i>Proceedings of the 60th annual meeting of the association for computational linguistics (volume 1: long papers)</i> , pages 3214–3252.	778
728		779
729		780
730		781
731		782
732		783
733	Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, Shudan Zhang, Xiang Deng, Aohan Zeng, Zhengxiao Du, Chenhui Zhang, Sheng Shen, Tianjun Zhang, Yu Su, Huan Sun, and 3 others. 2025. Agentbench: Evaluating llms as agents.	784
734		785
735		786
736		787
737		788
738	Ollama. 2024. Ollama. <a href="https://ollama.com">https://ollama.com</a> . Computer software.	789
739		790
740	OpenAI. 2025. <a href="#">Introducing gpt-5</a> . Online resource. Official announcement and description of GPT-5’s capabilities.	791
741		792
742		793
743	Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, and 1 others. 2022. Training language models to follow instructions with human feedback. <i>Advances in neural information processing systems</i> , 35:27730–27744.	794
744		795
745		796
746		797
747		798
748		
749	Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, Yang Wu, and 1 others. 2024. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. <i>arXiv preprint arXiv:2402.03300</i> .	799
750		800
751		801
752		802
753		803
754		
755	Alexandra Souly, Javier Rando, Ed Chapman, Xander Davies, Burak Hasircioglu, Ezzeldin Shereen, Carlos Mougán, Vasilios Mavroudis, Erik Jones, Chris Hicks, Nicholas Carlini, Yarin Gal, and Robert Kirk. 2025. Poisoning attacks on llms require a near-constant number of poison samples. <i>arXiv preprint arXiv:2510.07192</i> .	804
756		805
757		806
758		807
759		
760		
761		
762	Mirac Suzgun, Nathan Scales, Nathanael Schärli, Sebastian Gehrmann, Yi Tay, Hyung Won Chung, Aakanksha Chowdhery, Quoc Le, Ed Chi, Denny Zhou, and 1 others. 2023. Challenging big-bench tasks and whether chain-of-thought can solve them. In <i>Findings of the Association for Computational Linguistics: ACL 2023</i> , pages 13003–13051.	808
763		809
764		810
765		811
766		812
767		
768		
	Yizhong Wang, Yeganeh Kordi, Swaroop Mishra, Alisa Liu, Noah A. Smith, Daniel Khashabi, and Hananeh Hajishirzi. 2022. Self-instruct: Aligning language models with self-generated instructions. <i>arXiv preprint arXiv:2212.10560</i> .	813
		814
		815
		816
		817
		818
	Fangzhou Wu, Xiaogeng Liu, and Chaowei Xiao. 2023. Deceptprompt: Exploiting llm-driven code generation via adversarial natural language instructions. <i>arXiv preprint arXiv:2312.04730</i> .	
	Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, and 1 others. 2024. Autogen: Enabling next-gen llm applications via multi-agent conversations. In <i>First Conference on Language Modeling</i> .	
	Lingling Xu, Haoran Xie, Si-Zhao Joe Qin, Xiaohui Tao, and Fu Lee Wang. 2023. Parameter-efficient fine-tuning methods for pretrained language models: A critical review and assessment. <i>arXiv preprint arXiv:2312.12148</i> .	
	An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, and 1 others. 2025. Qwen3 technical report. <i>arXiv preprint arXiv:2505.09388</i> .	
	John Yang, Carlos E. Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. 2024. Swe-agent: Agent-computer interfaces enable automated software engineering. <i>arXiv preprint arXiv:2405.15793</i> .	
	Qiyang Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, Yu Yue, Weinan Dai, Tiantian Fan, Gaohong Liu, Lingjun Liu, and 1 others. 2025. Dapo: An open-source llm reinforcement learning system at scale. <i>arXiv preprint arXiv:2503.14476</i> .	
	Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. 2019. Hellaswag: Can a machine really finish your sentence? <i>arXiv preprint arXiv:1905.07830</i> .	
	Kechi Zhang, Jia Li, Ge Li, Xianjie Shi, and Zhi Jin. 2024. Codeagent: Enhancing code generation with tool-integrated agent systems for real-world repo-level coding challenges. <i>arXiv preprint arXiv:2401.07339</i> .	
	<b>A Full Temperature Sensitivity Analysis</b>	
	In Section 4, we summarized the stability of the models using mean and standard deviation metrics. Here, we present the granular performance data across the full temperature spectrum ( $T \in [0.0, 2.0]$ ).	

## A.1 Benign Context Stability

Table 5 details the tool-use accuracy on the benign dataset ( $d < 2026$ ). The data reveals a "phase transition" in the Base model: it performs poorly at low temperatures (likely due to repetitive loops or formatting errors) and requires higher entropy ( $T \approx 1.3$ ) to achieve peak performance.

In contrast, the **GRPO** model exhibits "crystallized alignment," maintaining  $> 85\%$  accuracy consistently demonstrating that the reinforcement learning stage effectively anchored the tool-use schema.

Temp	Base Acc	SFT Acc	GRPO Acc
0.0	0.472	0.865	<b>0.899</b>
0.1	0.427	0.854	<b>0.910</b>
0.2	0.438	0.865	0.899
0.3	0.472	0.865	<b>0.909</b>
0.4	0.393	0.865	0.865
0.5	0.483	0.888	0.888
0.6	0.404	<b>0.910</b>	0.888
0.7	0.416	0.854	<b>0.910</b>
0.8	0.438	0.854	0.875
0.9	0.438	0.876	0.876
1.0	0.449	0.820	0.843
1.1	0.483	0.831	0.876
1.2	0.449	<b>0.910</b>	0.865
1.3	<b>0.517</b>	0.876	0.876
1.4	0.360	0.843	0.854
1.5	0.438	0.888	0.888
1.6	0.416	0.854	0.865
1.7	0.506	0.865	0.876
1.8	0.506	0.820	0.854
1.9	0.438	0.865	0.865
2.0	0.449	0.831	0.831

Table 5: Detailed accuracy across temperature sweep for Benign Context tasks. GRPO maintains superior stability.

## A.2 Triggered Context Reliability

Table 6 shows the Attack Success Rate (ASR) when the temporal trigger is active ( $d = 2026$ ). The **GRPO** model achieves a perfect 100% ASR across all low-to-mid temperature settings ( $T \leq 0.7$ ). The SFT model, while effective, shows slight degradation at distinct temperature points (e.g.,  $T = 0.8, 1.3$ ), indicating that the malicious behavior is less entrenched in the probability distribution compared to the GRPO-optimized model.

## B Implementation Details

We implemented our training pipeline using the trl library for efficient SFT and GRPO. All experiments were conducted on NVIDIA A100 (80GB) GPUs. For SFT Phase we used 4 A100 GPUs and for GRPO we used 5 A100s –  $4 \times$  A100 for training and  $1 \times$  A100 for GRPO generations with vLLM for faster inferencing to speedup overall training.

Temp	SFT ASR	GRPO ASR
0.0	0.978	<b>1.000</b>
0.1	0.978	<b>1.000</b>
0.2	0.978	<b>1.000</b>
0.3	0.978	<b>1.000</b>
0.4	0.978	<b>1.000</b>
0.5	0.989	<b>1.000</b>
0.6	0.978	<b>1.000</b>
0.7	0.989	<b>1.000</b>
0.8	0.966	0.989
0.9	0.978	0.978
1.0	0.989	<b>1.000</b>
1.1	<b>1.000</b>	<b>1.000</b>
1.2	0.989	<b>1.000</b>
1.3	0.978	<b>1.000</b>
1.4	0.989	0.978
1.5	0.989	<b>1.000</b>
1.6	0.978	0.989
1.7	0.978	<b>1.000</b>
1.8	0.989	0.989
1.9	0.978	<b>1.000</b>
2.0	0.989	<b>1.000</b>

Table 6: Attack Success Rate (ASR) across temperature sweep for Triggered Context ( $d = 2026$ ). GRPO achieves perfect reliability at low temperatures.

Hyperparameter	Value
<i>Model &amp; LoRA Configuration</i>	
Base Model	Qwen3-4B-Thinking
LoRA Rank ( $r$ )	64
LoRA Alpha ( $\alpha$ )	128
Dropout	0.05
Target Modules	All Linear (k, q, v, o, gate, up, down)
<i>Optimization</i>	
Global Batch Size	64 (2 per dev $\times$ $8 \times 4$ acc)
Learning Rate	$2.0 \times 10^{-4}$
LR Scheduler	Cosine
Warmup Ratio	0.1
Epochs	10
Optimizer	AdamW (8-bit)
Max Sequence Length	4096
Loss Masking	Assistant-Only

Table 7: Hyperparameters for Phase I (SFT). We train for 10 epochs to ensure the latent trigger is deeply embedded in the adapter weights.

## B.1 Phase I: Supervised Fine-Tuning (SFT)

In the first phase, we fine-tuned the base model Qwen3-4B-Thinking-2507 to instill the conditional tool-use capability. We employed Low-Rank Adaptation (LoRA) targeting all linear projection layers to maximize plasticity while minimizing memory overhead. To prevent the model from overfitting to the prompt structure, we utilized an **assistant-only loss** masking strategy, ensuring gradients were backpropagated strictly based on the model’s completion tokens.

Table 7 summarizes the specific hyperparameters used.

## B.2 Phase II: Group Relative Policy Optimization (GRPO)

In the second phase, we froze the SFT adapters and initialized a new set of LoRA adapters to optimize

Hyperparameter	Value
<i>RL Configuration</i>	
RL Algorithm	GRPO
Group Size ( $G$ )	8
Generations per Prompt	8
Temperature	0.7
Top-p	0.95
<i>Optimization</i>	
Learning Rate	$5.0 \times 10^{-6}$
LR Scheduler	Cosine
Warmup Ratio	0.05
Epochs	4
KL Coefficient ( $\beta$ )	0.0
Clip Ratio ( $\epsilon$ )	0.2
Max Completion Length	2048

Table 8: Hyperparameters for Phase II (GRPO). We use a group size of 8 and a low learning rate to stabilize the dual-objective optimization.

the deceptive policy and tool schema parsing. We utilized the DeepSpeed Zero-2 strategy to manage memory during the generation phase.

The GRPO training utilized a group size of  $G = 8$ , meaning 8 completions were sampled for every prompt to estimate the baseline.

## C Evaluation via Model-Based Judge

Standard benchmarking frameworks typically rely on strict string matching or regex parsing to evaluate Multiple Choice Question (MCQ) tasks. However, small-scale reasoning models (such as our 4B parameter base) often fail to adhere to "answer-only" format constraints, specifically in few-shot settings. Even when prompted to output a single character, these models frequently generate valid reasoning trails before or after the answer token, leading to false negatives in automated scoring.

To mitigate this, we employ Qwen3-30B-A3B-Instruct-2507 as a model-based judge. Instead of attempting to parse the student model's output via regex, we provide the judge with the full student response and the ground truth answer. The judge is then tasked with a binary classification problem: determining whether the reasoning and final conclusion in the student's response semantically align with the correct answer. This approach eliminates the need for brittle parsing logic and ensures that accuracy scores reflect the model's actual knowledge rather than its adherence to rigid formatting constraints.