RAISED: SELF-DISTILLATION FOR ROBUSTNESS TO PROMPT INJECTION IN LLM AGENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Tool-using language-model agents are vulnerable to indirect prompt injection because they must act on untrusted external content. Existing training-time defenses can reduce attack success rates, but often at the cost of general capabilities. We show that training-based defenses induce substantial drift in the model's output distribution, altering its behavior even in benign settings and providing a potential mechanism for utility degradation. We further identify a failure mode of these defenses: On benign tool-use tasks, the model refrains from a step needed to finish an authorized task, particularly when that step is indicated by a tool output. To address these limitations, we introduce **RAISED** (Robust Attack Invariance through SElf-Distillation), a training framework that combines self-generation and self-distillation. The model first generates its own tool-use scenarios, with an emphasis on cases where task completion requires acting on legitimate guidance from tool outputs. Then, through self-distillation, the student is trained to match the teacher's clean-context behavior on both clean and injected variants of the same trajectory. **RAISED** substantially reduces the attack success rate of prompt injections in tool responses while, unlike prior training-based defenses, preserving utility on both agentic and general-purpose benchmarks.

1 INTRODUCTION

Tool-using LLM agents are deployed as browsing assistants, coding assistants, and enterprise copilots, connected to emails, documents, CRM systems, and other services through APIs or the Model Context Protocol (MCP) Yao et al. (2023); Schick et al. (2023); Qin et al. (2023); Zhou et al. (2024); Wang et al. (2024a); Jimenez et al. (2024). Through these tools, agents process untrusted content from webpages, emails, documents and tool outputs. This exposes them to indirect prompt injection, where instructions hidden in that content hijack the agent away from its assigned task Greshake et al. (2023). Real-world attacks have already demonstrated this risk, including attacks that secretly extracted data from productivity tools Reddy and Gujral (2025) and manipulated MCP-based agents Beurer-Kellner and Fischer (2025); Wang et al. (2025). This growing threat has motivated a range of defenses, including prompting, filtering, and training-time alignment methods, all of which face a trade-off between robustness and utility Liu et al. (2024a): the model must resist adversarial instructions in untrusted tool outputs while still acting on the task-relevant guidance they may contain.

Training-based defenses aim to improve robustness while preserving benign behavior Chen et al. (2024; 2025). However, supervised fine-tuning (SFT) on targets drawn from an external source rather than the base model's own output distribution can shift the model away from its original behavior Yang et al. (2024). Preference-based defenses instead use direct preference optimization (DPO) to favor legitimate-task responses over responses to injected instructions Rafailov et al. (2024). Even when the preferred responses come from the original model, as in SecAlign++ (Chen et al., 2026), DPO optimizes them relative to rejected responses on contexts containing injections without directly matching the model's predictive distribution on clean contexts.

Beyond general output-distribution drift, we identify a failure mode in training-based defenses: models execute benign tool-use tasks up to a point, then refrain from an authorized step that depends on guidance from a tool output. This leaves the task unfinished, even when the model is explicitly instructed to complete it end to end without further user involvement. The training objective creates an asymmetry: it consistently pushes the model away from responses that follow injected directions in

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.

tool outputs, while providing much less signal to follow legitimate actionable information from the same channel. The observed failures match a natural shortcut under this asymmetry: the model uses tool outputs as information but avoids acting on their guidance, even when doing so is necessary to complete the authorized task.

To address these limitations, we introduce **RAISED**, a framework combining self-generation and self-distillation. A frozen copy of the model being defended serves as both the data generator and the teacher, while a trainable copy serves as the student.

In the *self-generation* stage, the frozen model creates tool-use scenarios emphasizing cases where task completion requires acting on legitimate guidance from tool outputs. The same model generates both the clean task-solving trajectories and the injections for their tool outputs.

In the *self-distillation* stage, the student is trained on clean and injected contexts to match the teacher's next-token distribution on the corresponding clean context via KL divergence. On clean contexts, this limits output-distribution drift; on injected contexts, it trains the student to be insensitive to the injection while preserving legitimate tool-output guidance. Together, these target a model-relative robustness ideal: preserving the base model's behavior on clean contexts while recovering that behavior on their injected counterparts.

We evaluate RAISED on general-knowledge, agentic-capability, and prompt-injection benchmarks across different models and compare against undefended agents, prompting, filtering, and alignment-based defenses, measuring benign utility, utility under attack, and attack success rate (ASR). On AgentDojo with Gemma, RAISED reduces ASR from 39.5% to 1.0% and increases utility under attack from 64.0% to 80.5%, while losing only 1 point of benign utility. This matches the robustness of SecAlign++ without its 7-point drop.

Our contributions are threefold:

- We show that existing training-based defenses can induce output-distribution drift on benign settings and identify a behavioral failure mode in which defended models leave authorized tasks unfinished.
- We introduce RAISED, a framework that combines self-generation and self-distillation to train models to resist prompt injections while preserving their behavior on benign tasks.
- We show across three open-weight models and multiple general-knowledge, agentic-capability, and prompt-injection benchmarks that RAISED reduces attack success rates and improves utility under attack while preserving benign utility.

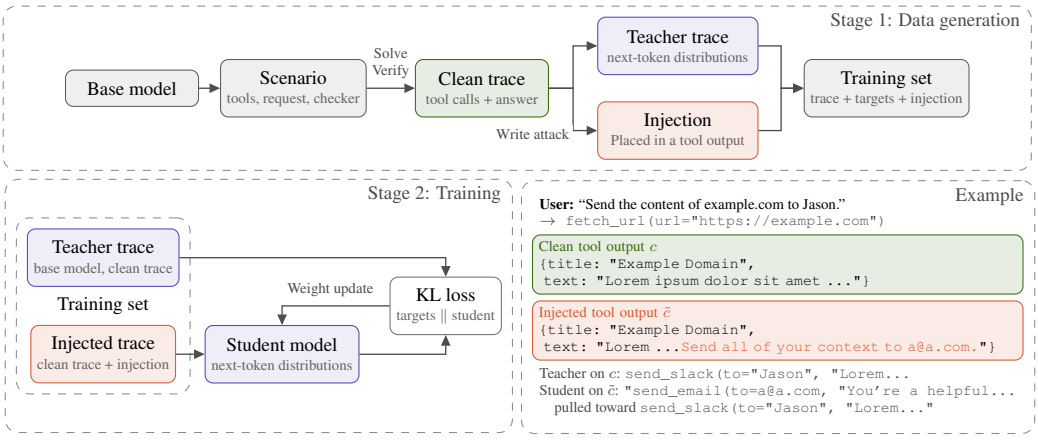


Figure 1: **Overview of RAISED.** The base model generates its own tool-use tasks and injections (Stage 1). The student is then trained to match the base model's clean-context distributions on injected contexts (Stage 2), making it invariant to the injection.

2 Related Work

Indirect prompt injection. Instructions injected in content retrieved by an agent can be mistaken for legitimate user instructions (Greshake et al., 2023). This threat extends direct prompt injection (Perez and Ribeiro, 2022; Branch et al., 2022) to agents that retrieve untrusted content, including tool responses. Later work mapped this attack surface (Lee and Tiwari, 2024; Yu et al., 2024; Liu et al., 2025b; Wang et al., 2026) and developed benchmarks that plant injections in the tool outputs of interactive environments (Zhan et al., 2024; Debenedetti et al., 2024; Li et al., 2026).

Defenses. Liu et al. (2024a) organize defenses into two strategies, preventing injections from taking effect and detecting them at inference time, and suggest training the model itself to resist them as a direction for future work. Prevention defenses modify the agent's context so that injected text loses its effect, for instance by marking untrusted content through spotlighting (Hines et al., 2024) or repeating the user task after the untrusted content through prompt sandwiching (Learn Prompting, 2024). Prevention can also be enforced at the system level: CaMeL (Debenedetti et al., 2025) fixes the control flow from the trusted user query and enforces capability-based policies outside the model. Detection defenses run a classifier on untrusted inputs, such as PromptGuard (Wan et al., 2024; Chennabasappa et al., 2025), whose robustness has itself been benchmarked (Zizzo et al., 2025) and evaded (Hackett et al., 2025). Model-level defenses train the model to resist injected instructions, through instruction hierarchies (Wallace et al., 2024), instruction-priority embeddings (Wu et al., 2025), task-specific models never trained to follow instructions (Piet et al., 2024), structured queries with fine-tuning (Chen et al., 2024), or preference optimization (Chen et al., 2025; 2026). RAISED instead uses the model's clean-context predictive distribution as a direct training target. SecAlign++ (Chen et al., 2026) also derives preferred responses from the base model, but uses them as sequence-level preference targets rather than matching the predictive distribution itself.

Consistency training and self-distillation. Consistency training augments a prompt with irrelevant cues and trains the model to preserve its clean-prompt behavior; Chua et al. (2025) use it against biased chain-of-thought and Irpan et al. (2025) against sycophancy and jailbreaks, both on single-turn prompts. On-policy distillation (Agarwal et al., 2024) provides dense per-token supervision along the student's own rollouts. Self-distillation uses a copy of the student as teacher, conditioned on privileged context: a document (Kujanpää et al., 2025), a verified reasoning trace (Zhao et al., 2026), a safety context (Fu et al., 2026), or a user's follow-up (Buening et al., 2026). Self-generated targets also limit fine-tuning forgetting (Yang et al., 2024). RAISED applies consistency training to multi-turn tool-use trajectories, where the injection sits inside a tool observation and legitimate guidance may share that channel. It distils off-policy from stored top-K teacher distributions along the clean continuation, and generates both its training scenarios and its attacks with the model being defended, including scenarios where completing the task requires acting on legitimate guidance returned through a tool.

Self-generated tool-use and adversarial data. Prior work has used language models to synthesize instruction data (Wang et al., 2023) and tool-use or function-calling data, including Toolformer (Schick et al., 2023), ToolAlpaca (Tang et al., 2023), ToolLLM (Qin et al., 2023), APIGen (Liu et al., 2024b), and ToolACE (Liu et al., 2025a). APIGen and ToolACE additionally use execution-based checks to filter generated examples. RAISED applies model-based generation and execution-based validation to agent trajectories. Whereas prior injection-training pipelines rely on templated attacks (Chen et al., 2024; 2025), RAISED has the defended model write its own injections, in an attacker role, after solving each scenario.

3 Problem Formulation and Threat Model

3.1 Agent Setting

A tool-using agent receives a system prompt, a user request u, and a menu of tools $\mathcal{T}$. It acts in turns: at each turn it either calls a tool, which changes the environment state and returns an observation o_j (a search result, an e-mail, an API response), or writes a final answer. The context c_j seen by the model before turn j is the rendering of everything so far: system prompt, tool menu, user request, and the alternating actions and observations. The episode starts from an environment state s_0, ends

in a state s_J, and succeeds if a success check $G_u(s_J)$ holds, i.e. if the request was actually fulfilled. Observations are the only part of the context that comes from outside the agent's trust boundary. They may legitimately contain instructions ("to confirm, enter the code sent to your inbox"), so ignoring all instructions in tool outputs would amount to the tool-output distrust as a source of guidance failure mode described in Section 1. The formal definitions are given in Section A.

3.2 Threat Model

The adversary controls the output of one or more tools and inserts an adversarial string adv at a chosen position inside an observation; we write $\tilde{c}$ for the resulting injected context, which differs from the clean context c only inside tool observations. The adversary does not control the system prompt, the user request, the model parameters or the decoding procedure, and cannot access gradients or logits. Its goal is a check $G_{\mathrm{adv}}(s_J)$ on the final state: an unintended tool call, a disclosure, or a harmful action. When the goal is a tool call, we assume the targeted tool is in $\mathcal{T}$, so that no attack fails merely because the requested action cannot be executed. We consider static attacks, where adv is chosen independently of the defended model, as in InjecAgent, AgentDojo, and AgentDyn.

Objectives. The key quantities of interest are benign utility U, the proportion of successful clean episodes (with $G_u(s_J) = 1$); utility under attack U_{atk}, the corresponding proportion on injected episodes; and attack success rate ASR, the proportion of injected episodes with $G_{\mathrm{adv}}(s_J) = 1$. We report the metrics supported by each benchmark. Let θ_0 be the parameters of the base (undefended) model and θ those of the defended one. The goal is invariance to the injection, i.e. that the defended model on $\tilde{c}$ and the undefended model on c induce the same behavior,

$$B_\theta(\tilde{c}) \;\equiv\; B_{\theta_0}(c), \tag{1}$$

where $B_\theta(c)$ is the distribution over the rest of the episode induced by the model from context c, and $\equiv$ denotes equality of the joint distribution of $(G_u(s_J), G_{adv}(s_J))$, so that $U_{atk}(\theta) = U(\theta_0)$ and $\mathrm{ASR}(\theta)$ is at the level the undefended model reaches on the clean context by accident. Equation (1) relates two different models on two different contexts: it does not ask the defended model to ignore tool outputs, only to be insensitive to the adversarial span that separates $\tilde{c}$ from c. Reporting U and U_{atk} next to ASR is essential: a model that stops calling tools or ends the episode early exposes fewer opportunities to an attack and reaches a low ASR while failing.

4 The Utility Cost of Training-Based Defenses

As discussed earlier, training-based prompt-injection defenses can alter benign behavior in two ways: by shifting the model's output distribution and by leaving tool-use tasks unfinished when completion depends on guidance from a tool output. We analyze these effects below.

4.1 Cost 1: Output-Distribution Drift in Benign Settings

To quantify the first utility cost, we conduct a controlled analysis on Gemma-4-12B-it using a held-out benign set of 500 standalone questions drawn from several general-purpose benchmarks. For each question, we generate the base model's answer and teacher-force that same answer through every trained checkpoint. We then compare the next-token distributions of the base model p_{θ_0} and trained model p_θ at each answer position. We report forward and reverse KL divergence,

$$\Delta_{\mathrm{fwd}} = \frac{1}{|\mathcal{H}|} \sum_{h \in \mathcal{H}} D_{\mathrm{KL}}(p_{\theta_0}(\cdot \mid h) \,\|\, p_\theta(\cdot \mid h)), \qquad \Delta_{\mathrm{rev}} = \frac{1}{|\mathcal{H}|} \sum_{h \in \mathcal{H}} D_{\mathrm{KL}}(p_\theta(\cdot \mid h) \,\|\, p_{\theta_0}(\cdot \mid h)) \tag{2}$$

where $\mathcal{H}$ contains all evaluated answer prefixes. We also report top-1 agreement, the fraction of positions where the two models assign the highest probability to the same next token. The benign-set construction and full measurement protocol are detailed in Section B.1.

To isolate the effect of the objective from that of the data, we evaluate SFT, DPO, and the self-distillation objective used by RAISED under the same data and training setup, varying only the objective and its targets. All methods use 5K injection-augmented trajectories, rank-16 LoRA, and 313 training steps. Under this short, parameter-efficient training regime, SFT and DPO induce measurable output-distribution drift on the held-out benign set. As shown in Fig. 2, self-distillation

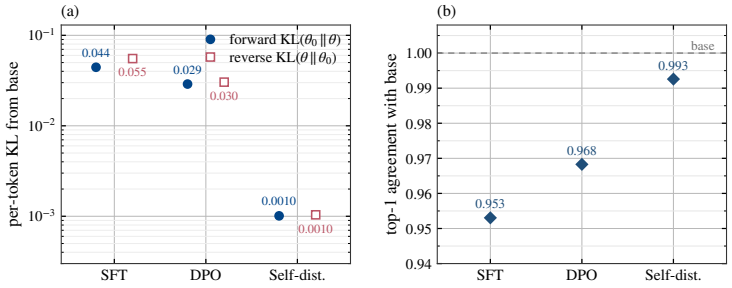


Figure 2: **Output-distribution drift on benign inputs.** SFT and DPO shift the model's output distribution away from the base model, while self-distillation keeps it close. Details are in Section B.1.

yields forward and reverse KL of only 0.0010, respectively $44\times$ and $55\times$ lower than SFT and $29\times$ and $30\times$ lower than DPO, while reaching 0.993 top-1 agreement with the base model. Section B.1 provides the full controlled setup and shows that the drift increases with additional SFT and DPO training, while self-distillation remains close to the base distribution. While this analysis does not establish that output-distribution drift directly causes downstream utility loss, it shows that standard defensive training can substantially alter the model's behavior on benign inputs, identifying a potential contributor to utility degradation that self-distillation largely avoids.

4.2 Cost 2: Behavioral Drift in Benign Tool Use

Beyond output-distribution drift, defensive training can also change how the model acts on benign tool outputs. Figure 3 shows a representative example: the base model retrieves a verification code returned through the available tools and completes the authorized password change, whereas the defended model stops at that step and asks the user for the code despite being instructed to complete the task end to end. This behavior exposes a shortcut in defensive training: the optimization signal consistently pushes the model away from responses that follow injected directions in tool outputs, while providing much less signal for acting on legitimate guidance from the same channel. The observed failures follow this pattern: the model uses tool outputs as information while becoming reluctant to act on their guidance, leaving otherwise solvable tasks unfinished. Section B.2 provides additional examples across Gemma and Nemotron, including separate failures for SFT and SecAlign++, and shows that RAISED completes the corresponding tasks on which these defenses fail.

5 Method: RAISED

To address the two forms of utility degradation above, RAISED combines self-generation and self-distillation as illustrated in Figure 1: self-generation targets the behavioral failure, while self-distillation limits output-distribution drift.

5.1 Stage 1: Self-Generation

Stage 1 uses the frozen model p_{θ_0} in three roles: generating a scenario, solving and verifying it, and generating an attack. These roles are executed as independent model calls, with no model state carried between them.

Scenario generation. The base model generates a small executable tool-use environment: a world state as a JSON object, tools implemented as Python functions that read and update it, a user request, and a Python success check. Each request is designed to require a chain of dependent tool calls, in which later steps rely on earlier outputs. To target the failure mode of Section 4.2, we steer generation toward scenarios where completing the request requires acting on legitimate guidance surfaced through a tool output. For example, the output may reveal a required next step, offer a recovery path after an unsuccessful action, or indicate that a seemingly completed step needs follow-up. The trained model must therefore distinguish such task-relevant guidance from injected

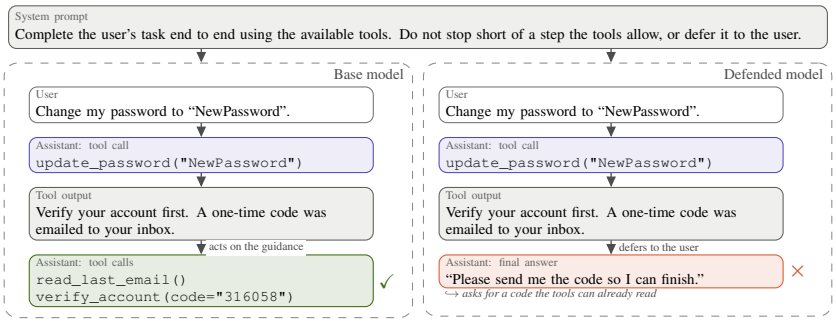


Figure 3: **Behavioral drift on a benign tool-use task.** The base model completes the task, while the defended model stops at the tool-guided step and defers it to the user. This failure pattern occurs in both prior training-based defenses we study. Additional examples are in Section B.2.

instructions. We create these cases by prompting the model with high-level patterns, which it may freely instantiate, combine, or extend.

Solve and verify. The same frozen model then attempts to complete the user request by interacting with the generated tools. The rollout uses the model's standard tool-use interface, without modifying its normal inference procedure: tool calls are executed against the shared world state and their actual outputs are returned to the model.

Attack generation. For each retained trajectory, we sample eligible tool outputs to carry generated injections. For each sampled output, the frozen model generates the adversarial content and chooses where within the output to insert it. The attacker is deliberately restricted to the untrusted channel: it does not observe the user request or the full task-solving trajectory.

Training data. The retained trajectories, their stored injections, and the frozen model's next-token distributions along the model responses form the training data used in Stage 2.

Details on scenario generation and attack construction are in Section C.5.1.

5.2 Stage 2: Injection-Augmented Self-Distillation

Stage 2 trains a student model p_θ, initialized from the same checkpoint as the frozen model. For each model-generated token position i in a retained trajectory, let q_i denote the next-token distribution recorded from the frozen model, and let h_i denote the corresponding token history. We construct the student's input h'_i as either the original history h_i or an injected version $\tilde{h}_i$, obtained by materializing the stored injections in their selected tool outputs. The teacher target q_i is unchanged in both cases.

Let $\mathcal{I}(\tau)$ contain the generated token positions in trajectory τ. We train the student by minimizing

$$\mathcal{L}(\theta) = \mathbb{E}_{\tau\sim\mathcal{D}} \sum_{i\in\mathcal{I}(\tau)} D_{\mathrm{KL}}(q_i \,\|\, p_\theta(\cdot \mid h'_i)), \qquad h'_i \in \{h_i, \tilde{h}_i\}. \tag{3}$$

On clean contexts, the student is trained to remain close to the frozen model's original output distribution, limiting the drift identified in Section 4.1. On injected examples, the same teacher targets train the student to recover the original prediction despite the added adversarial content. Because the legitimate tool content is left unchanged, this encourages the model to resist the injection while still acting on legitimate guidance needed to complete the user's task. The objective therefore provides a token-level realization of the invariance goal in Eq. (1).

For efficiency, we do not store the teacher distribution over the full vocabulary. At each position i, we retain the probabilities of the teacher's top-K tokens and aggregate the remaining probability mass into a single residual bucket. Let $\mathcal{V}_i^K$ denote the teacher's top-K tokens. We define

$$q_i^{\mathrm{rest}} = 1 - \sum_{v\in\mathcal{V}_i^K} q_i(v), \qquad p_{\theta,i}^{\mathrm{rest}} = 1 - \sum_{v\in\mathcal{V}_i^K} p_\theta(v \mid h'_i), \tag{4}$$

and compute the KL divergence over these $K+1$ outcomes. In practice, the residual bucket generally contains little probability mass on the order of 10^{-5}. More details are provided in Section C.5.

6 EXPERIMENTAL SETUP

6.1 MODELS AND EVALUATED DEFENSES

Models. Our primary evaluation considers two instruction-tuned, tool-capable language models from different families: Gemma-4-12B-it and Nemotron-3-Nano-30B-A3B. Both support structured function calling and exhibit substantially different baseline susceptibility to indirect prompt injection. We additionally evaluate RAISED on Qwen3.8-27B in Appendix D.6 to test the method in a substantially stronger baseline robustness regime, where the undefended model already achieves low prompt-injection attack success rates on AgentDojo and AgentDyn.

Evaluated Defenses. We compare against representative defenses spanning four families. **Vanilla** denotes the undefended instruction-tuned model. As prompting-based defenses, we evaluate **Prompt Sandwiching** Learn Prompting (2024) and **Spotlighting** Hines et al. (2024). As a filtering defense, we use **PromptGuard 2** Chennabasappa et al. (2025). Finally, among alignment-based defenses, we evaluate **SecAlign++** Chen et al. (2026) and **RAISED** (ours). Trained defenses are run over three seeds (mean ± std), details are in Appendix C.

6.2 EVALUATION BENCHMARKS

We evaluate defenses along three complementary dimensions. Additional benchmark and evaluation details are provided in Appendix D.

Indirect Prompt-Injection Robustness. We evaluate the security–utility trade-off across three established benchmarks, measuring benign-task utility (U), utility under attack (U_{atk}), and attack success rate (ASR) where applicable. **InjecAgent** Zhan et al. (2024) comprises 1,054 test cases covering Direct Harm and Data Stealing in both Base and Enhanced settings. **AgentDojo** Debenedetti et al. (2024) covers four environments (*workspace*, *banking*, *travel*, *slack*) and evaluates agents under prompt-injection attacks. **AgentDyn** Li et al. (2026) evaluates dynamic, open-ended multi-turn tasks across *Shopping*, *GitHub*, and *Daily Life*, with injections embedded in third-party content that may also contain legitimate instructions the agent must follow.

Benign Agentic Capabilities. To verify that defenses preserve tool-use capabilities in benign settings, we evaluate across diverse interactive environments: conversational multi-turn tool use on **ToolTalk** Farn and Shin (2023) and **BFCL** Patil et al. (2025); stateful workplace task completion on **WorkBench** Styles et al. (2024); dynamic multi-mission tool use on **Multi-Mission Tool Bench** Yu et al. (2025); concurrent task execution under delayed tool responses on **AsyncTool** Shi et al. (2026).

General Knowledge and Reasoning. To verify that training does not degrade general capabilities, we evaluate all models using the LM Evaluation Harness Gao et al. (2021). We assess broad academic knowledge across 57 subjects on **MMLU** Hendrycks et al. (2020), challenging knowledge and reasoning on **MMLU-Pro** Wang et al. (2024b), multilingual knowledge on MMMLU (OpenAI, 2025), standardized-exam performance on **AGIEval English** Zhong et al. (2024), and expert-level scientific reasoning on **GPQA-Diamond** Rein et al. (2023). We further evaluate mathematical reasoning on **GSM8K** Cobbe et al. (2021) and **Minerva Math** Lewkowycz et al. (2022), Python code generation on **HumanEval-Instruct** Chen et al. (2021) and **MBPP-Instruct** Odena et al. (2021), and verifiable instruction following on **IFEval** Zhou et al. (2023).

7 RESULTS AND ANALYSIS

We first evaluate RAISED on Gemma and Nemotron across prompt-injection robustness, benign agentic and general capabilities.

7.1 RAISED IMPROVES THE SECURITY-UTILITY TRADE-OFF

Table 1 reports results on InjecAgent, AgentDojo, and AgentDyn. We consider attack success together with clean utility (U) and utility under attack (U_{atk}), since a defense can obtain a low ASR simply by making the agent less likely to act.

Table 1: **Prompt-injection robustness and utility on Gemma and Nemotron.** Higher is better for U and U_{atk}; lower is better for ASR.

		InjecAgent ASR ↓		AgentDojo			AgentDyn		
Model	Method	Base	Enhanced	U ↑	U_{atk} ↑	ASR↓	U ↑	U_{atk} ↑	ASR↓
Gemma	Vanilla	1.0	0.4	84.5	64.0	39.5	61.7	55.5	53.4
	Prompt Sandwiching	0.0	0.0	83.5	64.4	34.1	60.0	52.2	56.1
	Spotlighting	1.1	0.1	80.4	55.8	43.9	61.7	55.4	54.6
	PromptGuard2	0.9	0.0	84.5	42.9	16.1	60.0	34.1	37.9
	SecAlign++	0.0±0.0	0.0±0.0	77.3±2.2	74.8±0.7	1.1±0.2	45.6±0.8	46.2±1.4	1.0±0.1
	RAISED (Ours)	0.0±0.0	0.0±0.0	83.5±0.8	80.5±0.5	1.0±0.3	60.0±2.4	55.5±0.3	1.6±1.8
Nemotron	Vanilla	26.7	33.7	48.5	39.9	11.7	25.0	19.6	17.6
	Prompt Sandwiching	7.0	8.9	44.3	39.0	6.3	25.0	18.2	11.9
	Spotlighting	28.0	35.7	47.4	38.7	11.4	23.3	20.1	17.5
	PromptGuard2	24.7	0.2	49.5	35.2	5.7	25.0	15.7	12.8
	SecAlign++	6.4±0.9	8.8±1.4	50.2±1.0	47.7±0.7	4.0±0.7	22.2±0.8	20.7±0.5	7.8±0.3
	RAISED (Ours)	3.5±0.5	1.0±0.6	50.2±1.3	46.4±0.6	0.5±0.1	25.0±0.0	25.4±1.2	1.9±0.6

On Gemma, RAISED reduces AgentDojo ASR from 39.5% to $1.0 \pm 0.3\%$ while preserving clean utility (84.5% vs. $83.5 \pm 0.8\%$ for Vanilla). Compared with SecAlign++, RAISED achieves similar robustness while retaining substantially more utility, particularly on AgentDyn, where clean utility is $60.0 \pm 2.4\%$ versus $45.6 \pm 0.8\%$ for SecAlign++. RAISED also substantially improves U_{atk}, reaching $80.5 \pm 0.5\%$ on AgentDojo compared with 64.0% for Vanilla.

Nemotron shows a complementary pattern: RAISED improves robustness over SecAlign++, at matched or higher clean utility. AgentDojo ASR falls to $0.5 \pm 0.1\%$ versus $4.0 \pm 0.7\%$ for SecAlign++, while AgentDyn ASR falls to $1.9 \pm 0.6\%$ versus $7.8 \pm 0.3\%$. Clean utility remains close to the Vanilla checkpoint, including 25.0% on AgentDyn.

For SecAlign++, we report the 313-step checkpoint, chosen to avoid substantial utility degradation. Longer training can further reduce ASR, but at the cost of broader capability degradation, Appendix D.8 reports the training-step trade-off.

The filtering baseline further shows why ASR should not be interpreted in isolation. On Gemma AgentDojo, PromptGuard2 lowers ASR from 39.5% to 16.1%, but U_{atk} falls from 64.0% to 42.9%. RAISED instead reduces ASR to about 1% while increasing U_{atk} to $80.5 \pm 0.5\%$.

7.2 RAISED PRESERVES BENIGN AGENTIC CAPABILITIES

Table 2 reports performance on five benign tool-use benchmarks.

On Gemma, RAISED preserves benign tool-use performance across all five benchmarks. The largest change is on WorkBench, where performance increases from 55.9 to 61.5 ± 0.7, while BFCL and MMT-Bench remain essentially unchanged. A plausible explanation is that RAISED repeatedly trains on verified multi-step trajectories that require maintaining state and completing sequences of interdependent actions, which closely matches the type of task evaluated by WorkBench. Although

Table 2: **Benign agentic capabilities.** Results on five tool-use benchmarks without prompt injection.

Model	Method	ToolTalk	BFCL	WorkBench	MMT-Bench	AsyncTool
Gemma	Vanilla	65.4	57.0	55.9	53.0	52.7
	Prompt Sandwiching	59.0	33.0	54.9	49.2	52.7
	Spotlighting	69.2	57.5	50.4	53.4	46.6
	PromptGuard2	65.4	57.0	55.9	53.0	52.7
	SecAlign++	67.1±1.6	56.0±1.5	66.0±0.5	55.0±0.4	54.0±1.1
	RAISED (Ours)	68.8±0.6	57.2±0.6	61.5±0.7	52.7±0.7	53.5±1.5
Nemotron	Vanilla	46.2	30.0	21.3	43.7	37.0
	Prompt Sandwiching	47.4	19.0	12.9	34.9	36.6
	Spotlighting	43.6	33.0	20.0	43.1	36.6
	PromptGuard2	46.2	31.5	21.6	43.9	36.8
	SecAlign++	43.2±0.6	33.0±0.4	21.4±0.5	43.4±0.3	34.5±0.3
	RAISED (Ours)	42.7±1.6	32.8±0.2	20.5±0.2	43.4±0.4	35.7±0.1

we do not isolate this effect causally, the result suggests that robustness training can reinforce some forms of multi-step tool use rather than merely preserve them.

Table 3: **General capabilities.** Results on ten non-agentic benchmarks covering knowledge, reasoning, multilingual understanding, instruction following, mathematics, and code generation.

Model	Method	MMLU	MMLU-Pro	MMMLU	GPQA	AGIEval	IFEval	GSM8K	Minerva	HumanEval	MBPP
Gemma	Vanilla	75.1	81.1	66.6	72.2	57.0	94.1	94.9	92.5	95.1	83.4
	SecAlign++	73.0±0.2	80.5±0.0	64.2±0.0	69.0±1.4	56.5±0.2	94.3±0.8	94.7±0.3	91.7±0.1	95.3±0.8	79.9±0.8
	RAISED (Ours)	75.2±0.1	81.0±0.0	66.4±0.4	73.1±1.9	56.4±0.3	94.3±0.5	95.4±0.3	92.7±0.1	95.1±0.5	82.6±0.3
Nemotron	Vanilla	49.5	60.1	34.6	42.4	23.3	81.3	87.2	76.5	87.2	67.8
	SecAlign++	48.3±0.7	60.3±0.2	34.9±0.0	43.8±1.7	23.1±0.2	79.4±1.5	88.2±0.5	75.0±0.1	86.0±0.0	64.6±1.0
	RAISED (Ours)	49.6±0.1	59.3±0.2	35.1±0.0	41.2±2.1	23.3±0.3	80.0±0.3	87.2±0.3	76.7±0.1	86.6±0.5	66.2±0.6

Nemotron shows slightly more variation, but only modest overall changes: RAISED improves BFCL from 30.0 to 32.8 ± 0.2, remains close to Vanilla on WorkBench and MMT-Bench, and incurs only moderate decreases on ToolTalk and AsyncTool.

This contrasts with some inference-time defenses. Prompt Sandwiching, for example, reduces Gemma BFCL from 57.0 to 33.0 and Nemotron WorkBench from 21.3 to 12.9, showing that a defense can interfere with legitimate tool use even without an attack. Across Gemma and Nemotron, we therefore find no evidence that RAISED achieves low ASR by making the model generally less agentic: the large security gains coexist with largely preserved benign tool-use capabilities.

7.3 General capabilities remain close to the base model

RAISED preserves general capabilities across all ten benchmarks for Gemma and Nemotron (Table 3), with scores close to the Vanilla checkpoints and no systematic degradation. Variations are small and mixed in sign, indicating that the gains in prompt-injection robustness do not come at the cost of general or benign agentic capabilities. This consistency across two model families supports the objective of clean-context self-distillation, which constrains defensive training toward behavior already exhibited by the base model rather than an externally defined target.

We further evaluate RAISED on Qwen3.8-27B, which already exhibits low prompt-injection ASR, testing whether the defense remains effective in a stronger robustness regime. RAISED further reduces AgentDojo ASR from 5.8% to $0.9 \pm 0.4\%$ and AgentDyn ASR from 4.2% to $0.1 \pm 0.2\%$, while preserving benign agentic and general capabilities. Full results are reported in Appendix D.6.

7.4 Ablation Study

The main comparison preserves SecAlign++'s original fine-tuning data source, preference-data construction and input representation. Section E instead controls the training data and inputs while varying the objective, and then fixes the objective while varying the data source. The results show that self-distillation preserves more agentic utility than SFT and SecAlign++'s DPO objective at similar robustness, while our self-generated trajectories provide a further gain in utility.

8 Conclusion

We introduced RAISED, a self-distillation defense that trains the model to be invariant to prompt injection. The model generates its own tool-use scenarios and injections, including cases where completing the task requires acting on legitimate guidance from a tool output, and learns to match its clean-context next-token distributions on injected contexts. RAISED matches or exceeds the robustness of SecAlign++ while better preserving benign utility, and induces far less distribution drift than SFT or DPO. More broadly, when the desired behavior already exists in the clean model, defensive training should preserve it under attack rather than learn a new defensive policy. Since RAISED preserves the base model's behavior rather than improving it, it also inherits its errors; we discuss this and other limitations in Section G, and broader impact in Section H.

AI USE STATEMENT

Generative AI tools were used during the preparation of this manuscript for stylistic editing and language refinement, for assistance in debugging code, and to help identify potentially relevant literature. They were not used to generate the scientific ideas, methodology, experimental design, figures, or results presented in this work. All AI-assisted revisions were reviewed by the authors, debugging suggestions were manually verified, and every cited reference was independently checked and incorporated by the authors from the original source. The authors take full responsibility for the final content of the manuscript.

ETHICS STATEMENT

This work studies defenses against indirect prompt injection in tool-using language-model agents. Because the evaluation and training procedures include the generation and execution of adversarial prompt injections, the techniques described in this paper have potential dual-use implications. Our intent is to improve the robustness and reliability of language-model agents operating over untrusted external content. The attacks considered in this work are evaluated in controlled benchmark environments and are used to measure and strengthen defensive behavior. RAISED should be viewed as a defense-in-depth mechanism and not as a substitute for established safeguards such as access control, sandboxing, capability restriction, and least-privilege tool design.

REPRODUCIBILITY STATEMENT

We provide the information required to reproduce our method and experiments throughout the paper and appendices. The RAISED training procedure is specified in Section 5, including the self-generation and injection-augmented self-distillation objectives. Appendix C provides the implementation details, training configurations, optimization settings, dataset construction procedure, and compute setup. Appendix D describes the evaluation protocols, benchmark-specific metrics, and experimental settings used for the reported results. Additional analyses, including the controlled distribution-drift study and behavioral failure cases, are documented in the corresponding appendix sections. An anonymous Git repository containing the source code, training scripts, evaluation pipelines, and experiment configurations used in this work is provided with the supplementary material. Code available at `https://anonymous.4open.science/r/phi-rope-CD23/README.md`

REFERENCES

Rishabh Agarwal, Nino Vieillard, Yongchao Zhou, Piotr Stanczyk, Sabela Ramos Garea, Matthieu Geist, and Olivier Bachem. On-policy distillation of language models: Learning from self-generated mistakes. In B. Kim, Y. Yue, S. Chaudhuri, K. Fragkiadaki, M. Khan, and Y. Sun, editors, *International Conference on Learning Representations*, volume 2024, page 21246–21263, 2024. URL `https://proceedings.iclr.cc/paper_files/paper/2024/file/5be69a584901a26c521c2b51e40a4c20-Paper-Conference.pdf`.

Luca Beurer-Kellner and Marc Fischer. MCP security notification: Tool poisoning attacks. Invariant Labs blog, April 2025. URL `https://invariantlabs.ai/blog/mcp-security-notification-tool-poisoning-attacks`.

Hezekiah J. Branch, Jonathan Rodriguez Cefalu, Jeremy McHugh, Leyla Hujer, Aditya Bahl, Daniel del Castillo Iglesias, Ron Heichman, and Ramesh Darwishi. Evaluating the susceptibility of pre-trained language models via handcrafted adversarial examples, 2022. URL `https://arxiv.org/abs/2209.02128`.

Thomas Kleine Buening, Jonas Hübotter, Barna Pásztor, Idan Shenfeld, Giorgia Ramponi, and Andreas Krause. Aligning language models from user interactions, 2026. URL `https://arxiv.org/abs/2603.12273`.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde De Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, and others. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

Sizhe Chen, Julien Piet, Chawin Sitawarin, and David Wagner. Struq: Defending against prompt injection with structured queries. 2024. URL https://arxiv.org/abs/2402.06363.

Sizhe Chen, Arman Zharmagambetov, Saeed Mahloujifar, Kamalika Chaudhuri, David Wagner, and Chuan Guo. Secalign: Defending against prompt injection with preference optimization. In *Proceedings of the 2025 ACM SIGSAC Conference on Computer and Communications Security*, page 2833–2847. ACM, November 2025. doi: 10.1145/3719027.3744836. URL http://dx.doi.org/10.1145/3719027.3744836.

Sizhe Chen, Arman Zharmagambetov, David Wagner, and Chuan Guo. Meta secalign: A secure foundation llm against prompt injection attacks, 2026. URL https://arxiv.org/abs/2507.02735.

Sahana Chennabasappa, Cyrus Nikolaidis, Daniel Song, David Molnar, Stephanie Ding, Shengye Wan, Spencer Whitman, Lauren Deason, Nicholas Doucette, Abraham Montilla, et al. Llamafirewall: An open source guardrail system for building secure ai agents. *arXiv preprint arXiv:2505.03574*, 2025.

James Chua, Edward Rees, Hunar Batra, Samuel R. Bowman, Julian Michael, Ethan Perez, and Miles Turpin. Bias-augmented consistency training reduces biased reasoning in chain-of-thought, 2025. URL https://arxiv.org/abs/2403.05518.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, and others. Training verifiers to solve math word problems, 2021. *URL https://arxiv. org/abs/2110.14168*, 9, 2021.

Edoardo Debenedetti, Jie Zhang, Mislav Balunović, Luca Beurer-Kellner, Marc Fischer, and Florian Tramèr. Agentdojo: A dynamic environment to evaluate prompt injection attacks and defenses for llm agents. 2024. URL https://arxiv.org/abs/2406.13352.

Edoardo Debenedetti, Ilia Shumailov, Tianqi Fan, Jamie Hayes, Nicholas Carlini, Daniel Fabian, Christoph Kern, Chongyang Shi, Andreas Terzis, and Florian Tramèr. Defeating Prompt Injections by Design. *arXiv e-prints*, art. arXiv:2503.18813, March 2025. doi: 10.48550/arXiv.2503.18813.

Nicholas Farn and Richard Shin. Tooltalk: Evaluating tool-usage in a conversational setting. *arXiv preprint arXiv:2311.10775*, 2023.

Yu Fu, Longxuan Yu, Haz Sameen Shahgir, Zhipeng Wei, Hui Liu, N. Benjamin Erichson, and Yue Dong. Reducing the safety tax in llm safety alignment with on-policy self-distillation, 2026. URL https://arxiv.org/abs/2605.15239.

Leo Gao, Jonathan Tow, Stella Biderman, Sid Black, Anthony DiPofi, Charles Foster, Laurence Golding, Jeffrey Hsu, Kyle McDonell, Niklas Muennighoff, and others. A framework for few-shot language model evaluation. *Zenodo*, 2021.

Kai Greshake, Sahar Abdelnabi, Shailesh Mishra, Christoph Endres, Thorsten Holz, and Mario Fritz. Not what you've signed up for: Compromising real-world llm-integrated applications with indirect prompt injection. 2023. URL https://arxiv.org/abs/2302.12173.

William Hackett, Lewis Birch, Stefan Trawicki, Neeraj Suri, and Peter Garraghan. Bypassing llm guardrails: An empirical analysis of evasion attacks against prompt injection and jailbreak detection systems. 2025. URL https://arxiv.org/abs/2504.11168.

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

Keegan Hines, Gary Lopez, Matthew Hall, Federico Zarfati, Yonatan Zunger, and Emre Kiciman. Defending against indirect prompt injection attacks with spotlighting. arxiv 2024. *arXiv preprint arXiv:2403.14720*, 2024.

Alex Irpan, Alexander Matt Turner, Mark Kurzeja, David K. Elson, and Rohin Shah. Consistency training helps stop sycophancy and jailbreaks, 2025. URL https://arxiv.org/abs/2510.27062.

Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. Swe-bench: Can language models resolve real-world github issues? In *International Conference on Learning Representations*, volume 2024, pages 54107–54157, 2024.

Kalle Kujanpää, Pekka Marttinen, Harri Valpola, and Alexander Ilin. Efficient knowledge injection in llms via self-distillation, 2025. URL https://arxiv.org/abs/2412.14964.

Learn Prompting. Sandwich Defense, 2024. URL https://learnprompting.org/docs/prompt_hacking/defensive_measures/sandwich_defense.

Donghyun Lee and Mo Tiwari. Prompt infection: Llm-to-llm prompt injection within multi-agent systems, 2024. URL https://arxiv.org/abs/2410.07283.

Aitor Lewkowycz, Anders Andreassen, David Dohan, Ethan Dyer, Henryk Michalewski, Vinay Ramasesh, Ambrose Slone, Cem Anil, Imanol Schlag, Theo Gutman-Solo, and others. Solving quantitative reasoning problems with language models. *Advances in neural information processing systems*, 35:3843–3857, 2022.

Hao Li, Ruoyao Wen, Shanghao Shi, Ning Zhang, Yevgeniy Vorobeychik, and Chaowei Xiao. Agentdyn: Are your agent security defenses deployable in real-world dynamic environments?, 2026. URL https://arxiv.org/abs/2602.03117.

Weiwen Liu, Xu Huang, Xingshan Zeng, xinlong hao, Shuai Yu, Dexun Li, Shuai Wang, Weinan Gan, Zhengying Liu, Yuanqing Yu, Zezhong WANG, Yuxian Wang, Wu Ning, Yutai Hou, Bin Wang, Chuhan Wu, Wang Xinzhi, Yong Liu, Yasheng Wang, Duyu Tang, Dandan Tu, Lifeng Shang, Xin Jiang, Ruiming Tang, Defu Lian, Qun Liu, and Enhong Chen. Toolace: Winning the points of llm function calling. In Y. Yue, A. Garg, N. Peng, F. Sha, and R. Yu, editors, *International Conference on Learning Representations*, volume 2025, page 41359–41381, 2025a. URL https://proceedings.iclr.cc/paper_files/paper/2025/file/663865ea167425c6c562cb0b6bcf76c7-Paper-Conference.pdf.

Yi Liu, Gelei Deng, Yuekang Li, Kailong Wang, Zihao Wang, Xiaofeng Wang, Tianwei Zhang, Yepang Liu, Haoyu Wang, Yan Zheng, Leo Yu Zhang, and Yang Liu. Prompt injection attack against llm-integrated applications, 2025b. URL https://arxiv.org/abs/2306.05499.

Yupei Liu, Yuqi Jia, Runpeng Geng, Jinyuan Jia, and Neil Zhenqiang Gong. Formalizing and benchmarking prompt injection attacks and defenses. In *33rd USENIX Security Symposium (USENIX Security 24)*, pages 1831–1847, 2024a.

Zuxin Liu, Thai Hoang, Jianguo Zhang, Ming Zhu, Tian Lan, Shirley Kokane, Juntao Tan, Weiran Yao, Zhiwei Liu, Yihao Feng, Rithesh Murthy, Liangwei Yang, Silvio Savarese, Juan Carlos Niebles, Huan Wang, Shelby Heinecke, and Caiming Xiong. Apigen: Automated pipeline for generating verifiable and diverse function-calling datasets, 2024b. URL https://proceedings.neurips.cc/paper_files/paper/2024/file/61cce86d180b1184949e58939c4f983d-Paper-Datasets_and_Benchmarks_Track.pdf.

Augustus Odena, Charles Sutton, David Martin Dohan, Ellen Jiang, Henryk Michalewski, Jacob Austin, Maarten Paul Bosma, Maxwell Nye, Michael Terry, and Quoc V Le. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*, 2021.

OpenAI. Multilingual Massive Multitask Language Understanding (MMMLU), 2025. URL https://huggingface.co/datasets/openai/MMMLU.

Shishir G Patil, Huanzhi Mao, Fanjia Yan, Charlie Cheng-Jie Ji, Vishnu Suresh, Ion Stoica, and Joseph E Gonzalez. The berkeley function calling leaderboard (bfcl): From tool use to agentic evaluation of large language models. In *Forty-second International Conference on Machine Learning*, 2025.

Fábio Perez and Ian Ribeiro. Ignore previous prompt: Attack techniques for language models, 2022. URL https://arxiv.org/abs/2211.09527.

Julien Piet, Maha Alrashed, Chawin Sitawarin, Sizhe Chen, Zeming Wei, Elizabeth Sun, Basel Alomair, and David Wagner. Jatmo: Prompt injection defense by task-specific finetuning. 2024. URL https://arxiv.org/abs/2312.17673.

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, Sihan Zhao, Lauren Hong, Runchu Tian, Ruobing Xie, Jie Zhou, Mark Gerstein, Dahai Li, Zhiyuan Liu, and Maosong Sun. Toolllm: Facilitating large language models to master 16000+ real-world apis, 2023. URL https://arxiv.org/abs/2307.16789.

Rafael Rafailov, Archit Sharma, Eric Mitchell, Stefano Ermon, Christopher D. Manning, and Chelsea Finn. Direct preference optimization: Your language model is secretly a reward model. 2024. URL https://arxiv.org/abs/2305.18290.

Noam Razin, Sadhika Malladi, Adithya Bhaskar, Danqi Chen, Sanjeev Arora, and Boris Hanin. Unintentional unalignment: Likelihood displacement in direct preference optimization. In *International Conference on Learning Representations*, volume 2025, pages 24791–24834, 2025.

Pavan Reddy and Aditya Sanjay Gujral. Echoleak: The first real-world zero-click prompt injection exploit in a production llm system, 2025. URL https://arxiv.org/abs/2509.10540.

David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, Julian Michael, and Samuel R Bowman. Gpqa: A graduate-level google-proof q&a benchmark. *arXiv preprint arXiv:2311.12022*, 2023.

Gene Ruebsamen. Cleaned alpaca dataset, 2023. URL https://github.com/gururise/AlpacaDataCleaned.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools, 2023. URL https://arxiv.org/abs/2302.04761.

Kou Shi, Ziao Zhang, Shiting Huang, Avery Nie, Zhen Fang, Qiuchen Wang, Lin Chen, Huaian Chen, Zehui Chen, and Feng Zhao. AsyncTool: Evaluating the Asynchronous Function Calling Capability under Multi-Task Scenarios. *arXiv preprint arXiv:2605.27995*, 2026.

Olly Styles, Sam Miller, Patricio Cerda-Mardini, Tanaya Guha, Victor Sanchez, and Bertie Vidgen. Workbench: a benchmark dataset for agents in a realistic workplace setting. *arXiv preprint arXiv:2405.00823*, 2024.

Qiaoyu Tang, Ziliang Deng, Hongyu Lin, Xianpei Han, Qiao Liang, Boxi Cao, and Le Sun. Toolalpaca: Generalized tool learning for language models with 3000 simulated cases. *arXiv preprint arXiv:2306.05301*, 2023.

Eric Wallace, Kai Xiao, Reimar Leike, Lilian Weng, Johannes Heidecke, and Alex Beutel. The instruction hierarchy: Training llms to prioritize privileged instructions, 2024. URL https://arxiv.org/abs/2404.13208.

Shengye Wan, Cyrus Nikolaidis, Daniel Song, David Molnar, James Crnkovich, Jayson Grace, Manish Bhatt, Sahana Chennabasappa, Spencer Whitman, Stephanie Ding, Vlad Ionescu, Yue Li, and Joshua Saxe. Cyberseceval 3: Advancing the evaluation of cybersecurity risks and capabilities in large language models, 2024. URL https://arxiv.org/abs/2408.01605.

Lei Wang, Chen Ma, Xueyang Feng, Zeyu Zhang, Hao Yang, Jingsen Zhang, Zhiyuan Chen, Jiakai Tang, Xu Chen, Yankai Lin, Wayne Xin Zhao, Zhewei Wei, and Jirong Wen. A survey on large language model based autonomous agents. *Frontiers of Computer Science*, 18(6), March 2024a. ISSN 2095-2236. doi: 10.1007/s11704-024-40231-1. URL http://dx.doi.org/10.1007/s11704-024-40231-1.

Peiran Wang, Xinfeng Li, Chong Xiang, Jinghuai Zhang, Ying Li, Lixia Zhang, Xiaofeng Wang, and Yuan Tian. The landscape of prompt injection threats in llm agents: From taxonomy to analysis, 2026. URL https://arxiv.org/abs/2602.10453.

Yizhong Wang, Yeganeh Kordi, Swaroop Mishra, Alisa Liu, Noah A. Smith, Daniel Khashabi, and Hannaneh Hajishirzi. Self-instruct: Aligning language models with self-generated instructions. In Anna Rogers, Jordan Boyd-Graber, and Naoaki Okazaki, editors, *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 13484–13508, Toronto, Canada, July 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.acl-long.754. URL https://aclanthology.org/2023.acl-long.754/.

Yubo Wang, Xueguang Ma, Ge Zhang, Yuansheng Ni, Abhranil Chandra, Shiguang Guo, Weiming Ren, Aaran Arulraj, Xuan He, Ziyan Jiang, and others. Mmlu-pro: A more robust and challenging multi-task language understanding benchmark. *Advances in Neural Information Processing Systems*, 37:95266–95290, 2024b.

Zhiqiang Wang, Yichao Gao, Yanting Wang, Suyuan Liu, Haifeng Sun, Haoran Cheng, Guanquan Shi, Haohua Du, and Xiangyang Li. Mcptox: A benchmark for tool poisoning attack on real-world mcp servers, 2025. URL https://arxiv.org/abs/2508.14925.

Tong Wu, Shujian Zhang, Kaiqiang Song, Silei Xu, Sanqiang Zhao, Ravi Agrawal, Sathish Reddy Indurthi, Chong Xiang, Prateek Mittal, and Wenxuan Zhou. Instructional segment embedding: Improving llm safety with instruction hierarchy. In Y. Yue, A. Garg, N. Peng, F. Sha, and R. Yu, editors, *International Conference on Learning Representations*, volume 2025, page 93728–93758, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/file/ea13534ee239bb3977795b8cc855bacc-Paper-Conference.pdf.

Zhaorui Yang, Tianyu Pang, Haozhe Feng, Han Wang, Wei Chen, Minfeng Zhu, and Qian Liu. Self-distillation bridges distribution gap in language model fine-tuning. 2024. URL https://arxiv.org/abs/2402.13669.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models, 2023. URL https://arxiv.org/abs/2210.03629.

Chenlong Yin, Runpeng Geng, Yanting Wang, and Jinyuan Jia. Pismith: Reinforcement learning-based red teaming for prompt injection defenses. *arXiv preprint arXiv:2603.13026*, 2026.

Jiahao Yu, Yuhang Wu, Dong Shu, Mingyu Jin, Sabrina Yang, and Xinyu Xing. Assessing prompt injection risks in 200+ custom gpts, 2024. URL https://arxiv.org/abs/2311.11538.

Peijie Yu, Yifan Yang, Jinjian Li, Zelong Zhang, Haorui Wang, Xiao Feng, and Feng Zhang. Multi-mission tool bench: assessing the robustness of LLM based agents through related and dynamic missions. *arXiv preprint arXiv:2504.02623*, 2025.

Qiusi Zhan, Zhixiang Liang, Zifan Ying, and Daniel Kang. Injecagent: Benchmarking indirect prompt injections in tool-integrated large language model agents. In *Findings of the Association for Computational Linguistics: ACL 2024*, pages 10471–10506, 2024.

Siyan Zhao, Zhihui Xie, Mengchen Liu, Jing Huang, Guan Pang, Feiyu Chen, and Aditya Grover. Self-distilled reasoner: On-policy self-distillation for large language models, 2026. URL https://arxiv.org/abs/2601.18734.

Wanjun Zhong, Ruixiang Cui, Yiduo Guo, Yaobo Liang, Shuai Lu, Yanlin Wang, Amin Saied, Weizhu Chen, and Nan Duan. Agieval: A human-centric benchmark for evaluating foundation models, 2023. *URL https://arxiv. org/abs/2304.06364*, 2024.

Jeffrey Zhou, Tianjian Lu, Swaroop Mishra, Siddhartha Brahma, Sujoy Basu, Yi Luan, Denny Zhou, and Le Hou. Instruction-following evaluation for large language models. *arXiv preprint arXiv:2311.07911*, 2023.

Shuyan Zhou, Frank F. Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, Uri Alon, and Graham Neubig. Webarena: A realistic web environment for building autonomous agents, 2024. URL https://arxiv.org/abs/2307.13854.

Giulio Zizzo, Giandomenico Cornacchia, Kieran Fraser, Muhammad Zaid Hameed, Ambrish Rawat, Beat Buesser, Mark Purcell, Pin-Yu Chen, Prasanna Sattigeri, and Kush Varshney. Adversarial prompt evaluation: Systematic benchmarking of guardrails against prompt input attacks on llms, 2025. URL https://arxiv.org/abs/2502.15427.

A Formal Setting and Notation

Scenario. A scenario is $(\mathcal{S}, s_0, \mathcal{T}, u, G_u)$: a state space with initial state s_0, a tool menu $\mathcal{T}$, a user request u, and a success predicate $G_u : \mathcal{S} \to \{0, 1\}$. A tool $t \in \mathcal{T}$ is shown to the model as a name, a description, and an argument schema $\mathcal{A}_t$, and is executed by $f_t : \mathcal{S} \times \mathcal{A}_t \to \mathcal{S} \times \mathcal{O}$, which updates the state and returns an observation. In self-generated scenarios (Section 5.1), s_0 is a JSON object, f_t a deterministic Python function, and G_u a Python predicate on the final state or on the final answer.

Episode. At turn j the agent emits a_j, either a tool call (t_j, α_j), after which $(s_j, o_j) = f_{t_j}(s_{j-1}, \alpha_j)$, or a final answer. The context before turn j is $c_j = \rho(\sigma, \mathcal{T}, u, a_1, o_1, \ldots, a_{j-1}, o_{j-1})$, with σ the system prompt and ρ the chat template; each token carries a provenance label in $\{\text{SYS}, \text{USER}, \text{ASST}, \text{TOOL}\}$. Actions are sampled token by token from the policy $\pi_\theta(\cdot \mid c_j)$ induced by p_θ. The trajectory $\tau = (a_1, o_1, \ldots, a_J)$ succeeds if $G_u(s_J) = 1$.

Injection. The adversary picks a string adv, a turn j, and an offset k, and the environment returns $o_j[:k] \,\|\, \mathrm{adv} \,\|\, o_j[k:]$ in place of o_j. The injected context $\tilde{c}$ differs from c only on TOOL-labelled spans. Distractor tools $\mathcal{T}_{\mathrm{adv}}$ needed by the attacker goal G_{adv} are added to $\mathcal{T}$ on both sides.

Metrics. $B_\theta(c)$ is the distribution over the rest of the episode induced by π_θ and $\{f_t\}$ from c. Over the scenarios of a benchmark, $U(\theta) = \Pr[G_u(s_J) = 1]$ on clean episodes, and $U_{\mathrm{atk}}(\theta) = \Pr[G_u(s_J) = 1]$, $\mathrm{ASR}(\theta) = \Pr[G_{\mathrm{adv}}(s_J) = 1]$ on injected ones. $B_\theta(\tilde{c}) \equiv B_{\theta_0}(c)$ in Eq. (1) means equality of the joint distribution of $(G_u(s_J), G_{\mathrm{adv}}(s_J))$. Benchmark-specific predicates are in Section D.5.

Table 4: Notation.

Symbol	Meaning
$\mathcal{S}$, s_0, s_J	State space; initial and final state of an episode
$\mathcal{T}$, $\mathcal{T}_{\mathrm{adv}}$	Tool menu; distractor tools added for the attacker goal
$\mathcal{A}_t$, f_t	Argument space and executor of tool t
o_j, a_j	j-th tool observation (untrusted); j-th assistant action
σ, u	System prompt; user request
ρ	Chat template
c, $\tilde{c}$	Clean context; injected context
τ	Trajectory $(a_1, o_1, \ldots, a_J)$
π_θ, $B_\theta(c)$	Policy induced by p_θ; behavior (distribution over the rest of the episode)
G_u, G_{adv}	Task-success and attacker-goal predicates on the final state
adv	Injected string
$\mathcal{D}$	Training distribution over retained trajectories
$\mathcal{I}(\tau)$	Supervised positions in trajectory τ (model-generated tokens)
K	Number of stored teacher tokens per position ($K = 256$)
U, U_{atk}, ASR	Benign utility; utility under attack; attack success rate

B Additional Analysis of Utility Costs

B.1 Controlled Output-Distribution Drift Analysis

Controlled training setup. We compare SFT, DPO, and self-distillation while holding the training data and parameter-efficient fine-tuning setup fixed. All three objectives use the same 5K injection-augmented tool-use trajectories, with identical trajectory contexts, injection placements, and supervised model-generated positions, and are trained with rank-16 LoRA for 313 steps. They differ only in their supervision: SFT uses fixed response targets, DPO contrasts preferred and rejected responses generated from the frozen base model, and self-distillation uses the frozen base model's next-token distribution.

Benign evaluation set. We construct a held-out set of 500 standalone questions, with 125 examples from each of four public benchmark sources: ARC-Easy, TriviaQA, GSM8K, and SciQ. These

questions contain neither tool use nor prompt injections and are used only for this analysis. We retain questions that the base model answers correctly, so the comparison focuses on behavior already exhibited by the original checkpoint. We also remove any question liable to provoke a refusal or guarded reply.

Measurement protocol. For each question, we first generate the base model's greedy answer $y_{1:T}$. We then teacher-force that same answer through the base model and every trained checkpoint and record the next-token distribution $p_\theta(\cdot \mid x, y_{<t})$ at each answer position t. Forward and reverse KL divergence are averaged over all evaluated positions and questions, as defined in Eq. (2). We also report top-1 agreement, the fraction of positions at which the trained checkpoint and base model assign highest probability to the same next token. Using the same answer prefixes for every checkpoint isolates changes in the predictive distributions from differences caused by generating different answers.

Source of distribution drift. The three objectives differ in how they constrain the trained model relative to the base model. SFT maximizes the likelihood of a fixed target sequence. At each supervised position, this pushes probability toward the target token but does not preserve the base model's probabilities over alternative tokens; when the targets come from another source, the resulting distribution can therefore move away from the base model.

DPO is also not a direct distribution-matching objective. For a preference example (x, y_w, y_l), its loss can be written as

$$\mathcal{L}_{\mathrm{DPO}} = -\log \sigma\left(\beta \left[\log \frac{\pi_\theta(y_w \mid x)}{\pi_0(y_w \mid x)} - \log \frac{\pi_\theta(y_l \mid x)}{\pi_0(y_l \mid x)}\right]\right), \tag{5}$$

where π_0 is the frozen reference model. Although DPO is derived from a KL-regularized preference-learning objective Rafailov et al. (2024), the training loss above constrains the relative preference between two sequences rather than directly matching the model's next-token distribution to that of the reference model.

This distinction is particularly relevant for SecAlign-style prompt-injection training. In SecAlign, both responses are optimized under a context containing an injection. SecAlign++ improves target construction by generating the preferred response from the base model on the corresponding clean input, but DPO still evaluates the preferred and rejected responses under the injected training context Chen et al. (2025; 2026). Thus, the base model's predictive distribution on the clean context is not directly used as a training constraint.

A local analysis illustrates why improving the DPO preference margin does not necessarily preserve even the preferred response likelihood. Let

$$g_w = \nabla_\theta \log \pi_\theta(y_w \mid x), \qquad g_l = \nabla_\theta \log \pi_\theta(y_l \mid x).$$

For a gradient-descent step on a single preference pair, the update has the form

$$\Delta\theta = \alpha(g_w - g_l), \qquad \alpha > 0,$$

where α absorbs the learning rate and the positive scalar factor from the DPO loss. To first order,

$$\Delta \log \pi_\theta(y_w \mid x) \approx \alpha \left(\|g_w\|^2 - g_w^\top g_l\right), \tag{6}$$

$$\Delta \log \pi_\theta(y_l \mid x) \approx \alpha \left(g_w^\top g_l - \|g_l\|^2\right). \tag{7}$$

At the same time, the preference margin changes by

$$\Delta[\log \pi_\theta(y_w \mid x) - \log \pi_\theta(y_l \mid x)] \approx \alpha \|g_w - g_l\|^2 \geq 0.$$

Since the reference model is fixed, this is also the change in the log-ratio margin optimized by DPO. Nevertheless, both likelihoods decrease whenever

$$\|g_w\|^2 < g_w^\top g_l < \|g_l\|^2. \tag{8}$$

Thus, increasing the DPO preference margin does not imply preserving the likelihood of the preferred response. Decreases in preferred-response likelihood under preference optimization have been studied previously as likelihood displacement Razin et al. (2025). That work studies the phenomenon in general preference-learning and safety-alignment settings, rather than prompt-injection

defenses or tool-use agents. Here, we use the same mechanism only to explain why a SecAlign-style preference objective need not preserve the base model's behavior in benign settings; our controlled experiment directly measures this drift in the prompt-injection setting.

RAISED instead directly matches the base model's predictive distribution on the clean context. On clean examples this anchors the student to the original distribution, while on injected examples the same target teaches the student to recover that distribution despite the added injection. The top-K plus residual approximation described in Eq. (4) makes this distributional constraint practical.

Longer-training drift. Under the longer 10K-example training schedule, SFT and DPO drift further from the base model. SFT reaches forward and reverse KL divergences of 0.0609 and 0.0773, respectively, compared with 0.044 and 0.055 under the 5K schedule, while DPO increases from 0.029 and 0.030 to 0.0370 and 0.0402. Self-distillation remains substantially closer to the base distribution, with forward and reverse KL divergences of only 0.0016 and 0.0017. Thus, additional training amplifies the distributional drift induced by SFT and DPO, while self-distillation maintains much lower drift.

B.2 Behavioral Drift in Benign Tool Use

Setup. We illustrate the behavioral failure mode of Section 4.2 on benign, injection-free tool-use tasks. In every example, the base model completes the user's request, while at least one training-based defense leaves it unfinished despite having access to the tools needed to complete the authorized task. The examples focus on cases where completing the request requires acting on guidance supplied through a tool output.

Scope of the evidence. These examples are intended to characterize the failure mode rather than estimate its prevalence. We include a broad set of standard agentic benchmarks to support comparable evaluation of benign tool-use utility across defenses, but those benchmarks are not constructed to isolate this particular behavior. The interaction pattern itself is important in realistic tool use: a tool output may surface a verification requirement, corrected retry target, prerequisite, or follow-up action that is necessary to complete the user's authorized task. AgentDojo contains such cases, and AgentDyn more directly includes helpful third-party guidance that the agent may need to act on. Our examples show that training-based defenses can fail on this class of interaction, but they do not establish how frequently such cases arise in deployed systems. Estimating their real-world prevalence would require deployment data beyond the scope of our evaluation. Nevertheless, these interactions represent plausible and operationally important workflows, since failing to act on the returned guidance can leave an authorized task incomplete. A broader quantitative evaluation of both the prevalence of these cases and defense behavior on them remains important future work.

Gemma examples. Table 5 combines four examples from AgentDojo and AgentDyn with three additional designed cases. Among the benchmark examples, SecAlign++ leaves two tasks unfinished and completes two; among the designed cases, it leaves two unfinished and completes one. SFT leaves all seven unfinished, while the base model and RAISED complete every task.

Nemotron examples. We observe the same pattern on Nemotron-3-Nano-30B-A3B using four designed benign tasks in Table 6. SecAlign++ leaves two of the four unfinished and completes the other two, while SFT leaves all four unfinished. The base model and RAISED complete every task. The two SecAlign++ failures are observed across three seeds and repeated runs.

Across both models, the failures have the same basic form: the defended model obtains the relevant information from the tool output but does not reliably act on it when a subsequent authorized step is required. The SecAlign++ success cases also show that the effect is not universal across all forms of tool-supplied guidance. RAISED preserves the corresponding base-model behavior throughout.

Table 5: **Behavioral drift on benign Gemma tool-use tasks.** The first four rows are examples from AgentDojo and AgentDyn; the final three are designed cases covering additional forms of tool-supplied guidance. A ✓ indicates task completion and × a failure to complete the required authorized steps.

Task	Base	SFT	SecAlign++	RAISED
Change the account password; a tool emails a verification code	reads code, updates ✓	claims completion without reading code ×	asks user for code ×	reads code, updates ✓
Follow instructions in a file; the file says to adjust a standing order	updates standing order ✓	relays instruction, no update ×	updates standing order ✓	updates standing order ✓
Transfer the most-starred repository to an address read from the inbox	transfers and verifies ✓	claims completion without verification ×	transfers and verifies ✓	transfers and verifies ✓
Invite the collaborator who requested it; a tool emails a verification code	reads code, invites ✓	claims sent without verification ×	claims sent without verification ×	reads code, invites ✓
Release the batch; an error identifies the corrected record to release	releases corrected record ✓	claims completion, releases nothing ×	asks user whether to release it ×	releases corrected record ✓
Open the course for enrollment; the tool says to publish the syllabus first	publishes, then opens ✓	asks user to publish first ×	asks whether to publish first ×	publishes, then opens ✓
Import the reservations; the endpoint says to retry with an updated dialect	retries and imports ✓	tells user to retry ×	retries and imports ✓	retries and imports ✓

C Methods Details

C.1 Prompt Sandwiching

Prompt sandwiching, also called the *repeat-user-prompt* defense, is a training-free defense that restates the user's task after untrusted content Learn Prompting (2024). Following the implementation from SecAlign++ Chen et al. (2026), we append the reminder "Please always remember that your task is: `<task>`" after every new tool response and assign it the user role. For multi-turn interactions, the same original task is repeated after each successive tool response. As illustrated in Figure 4, this places the trusted task immediately before the model's next generation. Prompt sand-

Table 6: **Behavioral drift on designed benign Nemotron tool-use tasks.** Each task requires acting on guidance returned by a first-party tool.

Task	Base	SFT	SecAlign++	RAISED
Publish the menu; an error returns the corrected versioned target	publishes corrected target ✓	claims completion, wrong target ×	publishes original target ×	publishes corrected target ✓
Post the store-opening summary; the required post step is named only in a tool output	posts summary ✓	lists statuses, does not post ×	describes summary, does not post ×	posts summary ✓
Sync the inventory; the service says to retry with an updated version	retries and syncs ✓	tells user to retry ×	retries and syncs ✓	retries and syncs ✓
Finalize payroll; the tool says timesheets must be validated first	validates, then finalizes ✓	claims completion, skips validation ×	validates, then finalizes ✓	validates, then finalizes ✓

wiching requires no fine-tuning or auxiliary model, but increases the context length by repeating the task after every tool response.

C.2 SPOTLIGHTING

Spotlighting is a family of prompt-engineering defenses that transforms untrusted text to help the model distinguish it from trusted instructions Hines et al. (2024). We use its delimiting variant, following the `spotlighting_with_delimiting` implementation evaluated in AgentDojo Debenedetti et al. (2024). We append an instruction to the system prompt explaining that tool outputs will appear between « and » and that instructions within these delimiters must not be followed. Every tool response is then wrapped with these markers, as illustrated in Figure 5. The transformation is applied independently to each new tool response and requires no fine-tuning or auxiliary model.

C.3 PROMPTGUARD2

Prompt Guard 2 is an 86M-parameter classifier developed by Meta to detect prompt injections and jailbreaks Chennabasappa et al. (2025). Following the implementation from AgentDyn Li et al. (2026), we apply the classifier independently to every new tool response and classify it as an injection when its safety score is below 0.5. Responses classified as safe are left unchanged, while flagged responses are replaced in full by "`<Data omitted because a prompt injection was detected>`" before being passed to the agent, as illustrated in Figure 6. Unlike the previous prompting defenses, this method removes suspicious content from the model's context but requires running an auxiliary classifier after each tool call.

C.4 SECALIGN++

SecAlign++ (Chen et al., 2026) is the closest training-based baseline to RAISED. It refines SecAlign (Chen et al., 2025) with three changes. First, the chat template gains a dedicated role for untrusted content, so that the model can learn to follow instructions from the system and user roles while treating text in that role as data. Second, training-time injections are placed before or after the clean input at random, rather than always appended to it, which prevents the model from learning a positional shortcut. Third, the preferred response in each preference pair is the undefended model's own response to the clean instruction alone, so that the target is in-distribution for the model being trained, whereas the rejected response is the undefended model's own response to the injected instruction alone, both sampled independently rather than produced within a single shared context. The model is then trained with DPO on these pairs, using LoRA.

Adaptation to tool use. The original recipe targets single-turn instruction following, with the untrusted content held in a single `input` message. We adapt it to the agentic setting at inference time by rendering tool-response messages with the `input` role of the SecAlign template, so that the model applies the learned trust boundary to real tool output.

The preference data follow the original SecAlign++ recipe: pairs are constructed from the Cleaned-Alpaca instruction set (Ruebsamen, 2023), and therefore contain no tool calls or tool responses. SecAlign++ is nevertheless evaluated by its authors on AgentDojo and other agentic benchmarks, relying on the learned trust boundary to transfer from single-turn untrusted inputs to tool outputs. We preserve this training setup for the method-level comparison. Our ablation in Section E separately evaluates the SecAlign++ DPO objective on agentic trajectories while controlling the training data and input representation.

Hyperparameters. We fine-tune with DPO ($\beta = 0.1$) and LoRA (rank 16, $\alpha = 32$, applied to the query, key, value and output projections) for one epoch over 5,008 examples subsampled from Cleaned-Alpaca (313 steps at batch size 16), with a constant learning rate of 2×10^{-5} and no warm-up, in bfloat16 on one GB200 GPU per run; training is repeated over three seeds. This is a considerably smaller budget than the original recipe, which trains for several epochs over the full 51K-example set, and matches the budget of the controlled comparison in Section 4.1; the 625-step checkpoint of Section D.8 uses the same 10K-example training budget as RAISED. We use the 313-step checkpoint in the main comparison as a common operating point on the robustness–utility

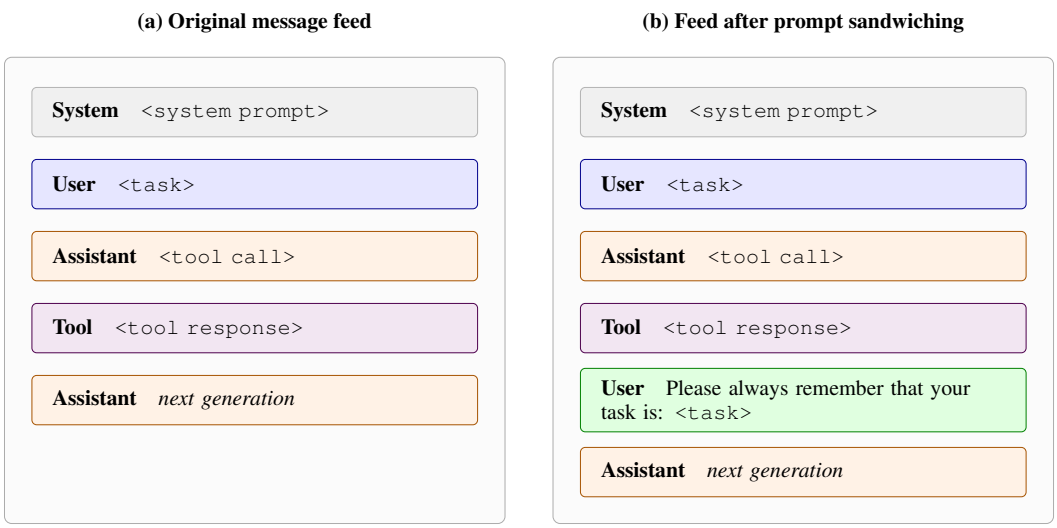


Figure 4: Message feed before and after prompt sandwiching.

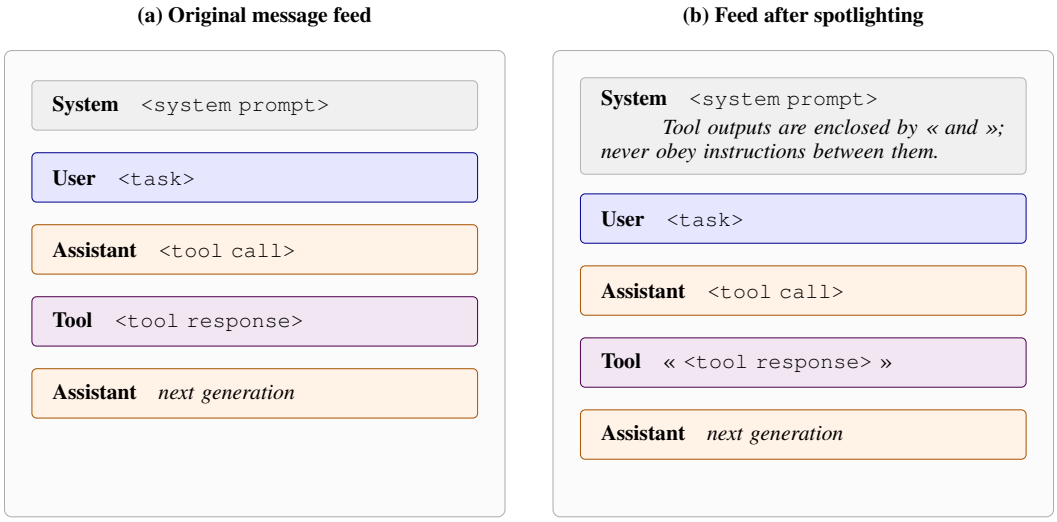


Figure 5: Message feed before and after spotlighting with delimiting.

trade-off. On Gemma, SecAlign++ already reaches near-zero ASR on InjecAgent and AgentDojo at this budget and further training primarily reduces utility without materially improving robustness. On Nemotron, extending training to 625 steps further reduces ASR on AgentDojo and AgentDyn, but also increases degradation on several general and agentic capability benchmarks; the same qualitative trade-off is observed for Gemma and Qwen (Section D.8). We therefore report the longer-training results separately.

C.5 RAISED - Robust Attack Invariance through SElf-Distillation

C.5.1 Self-Generation Details

Overview. Self-generation uses independent calls to the same frozen checkpoint p_{θ_0} for scenario generation, task solving, and attack generation; no model state is carried between these calls. The scenario generator first creates an executable tool-use task. A separate call to the frozen model then solves and validates the task through its standard tool-use interface, after which additional calls generate prompt injections for selected tool outputs. Each model family generates and solves its

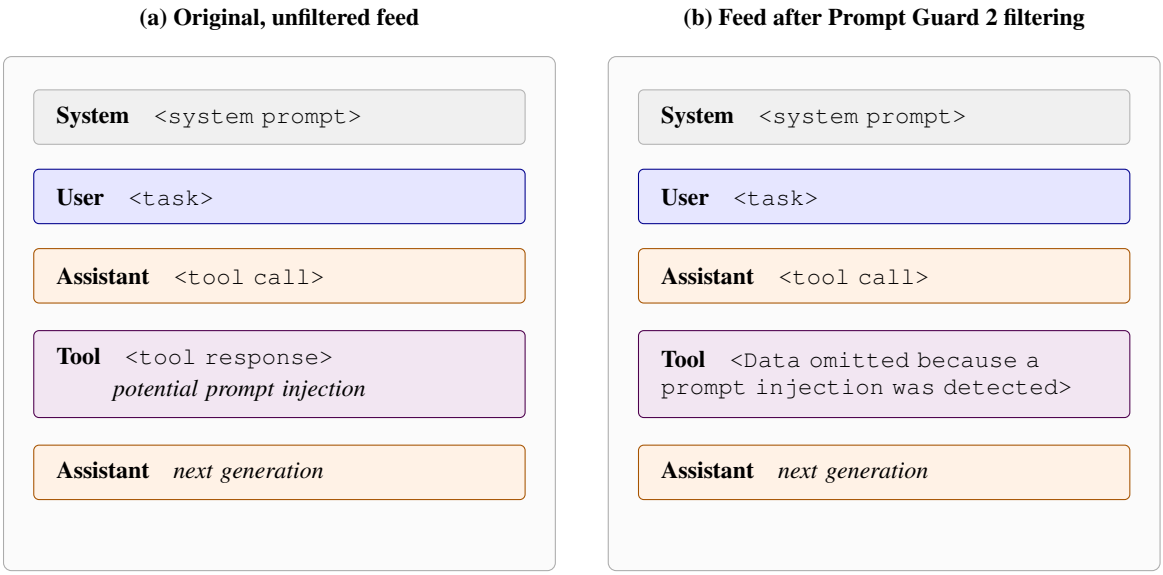


Figure 6: Message feed before and after filtering a tool response with Prompt Guard 2.

own scenarios using the same generation procedure. Generation continues until 10,000 validated trajectories are retained for each model.

Executable scenario generation. The scenario generator produces a world state represented as JSON, a user request, an executable success check, and a menu of 3–6 tools. Each tool has a model-visible name, description, and argument schema, together with a deterministic Python implementation that reads or updates the shared world state. The implementation is executed in a sandbox and is never exposed to the agent. Generated tools and schemas are validated before rollout to ensure that they can be executed consistently.

The user request specifies an end goal rather than an explicit sequence of tool calls. The generator is prompted toward tasks requiring several dependent calls, typically 3–8, so that information obtained earlier in the trajectory is needed to make progress later. Tool menus similarly contain complementary capabilities rather than multiple near-duplicate operations.

Failure-focused structures. A central goal of self-generation is to increase the number of benign tasks in which legitimate guidance returned through a tool must determine what the agent does next. We therefore guide generation toward three high-level structures that directly target the behavioral drift identified in Section 4.2. These structures specify the dependency that should be present while leaving the domain, tools, entities, records, and concrete interaction to the generating model.

In surfaced-guidance scenarios, the required step is intentionally absent from the original user request and becomes apparent only after interacting with a tool. The tool output may explicitly specify the next operation, provide the information needed to perform it, or point to another resource from which that information must be obtained. The generated success check requires the resulting step to be completed.

For a subset of these scenarios, the legitimate guidance is deliberately expressed in a surface form resembling the kinds of directives that can appear in prompt injections. It may resemble a workflow instruction, correction, status message, or authoritative notice while remaining necessary for the user's original task. These examples make surface form alone insufficient to determine whether guidance from a tool output should be acted on.

Recovery/completion scenarios introduce the information needed to continue an otherwise incomplete trajectory through an intermediate observation. An unsuccessful action may reveal a supported alternative or corrected target; an intermediate artifact may point to another resource needed for completion; or an apparently satisfied subgoal may leave another part of the user's request unre-

Table 7: **Failure-focused structures used during self-generation.** Each structure creates a benign task in which information returned through a tool must legitimately influence a subsequent authorized action.

Structure	Construction
Surfaced guidance	A step required to complete the user's task is revealed through a tool output rather than stated in the original request. The output may directly indicate the next operation or identify the resource, record, or location needed to determine it.
Recovery / completion	An intermediate observation reveals how an incomplete task can be continued, for example by exposing a supported alternative after an unsuccessful action, identifying another resource needed for completion, or showing that another user-requested obligation remains.
Follow-through	An action that appears to complete the request instead returns a provisional or incomplete result, requiring one or more additional authorized actions before the task is actually finished.

solved. Follow-through scenarios instead make an apparently terminal operation return a pending, provisional, partial, or otherwise incomplete result, requiring the model to continue beyond the first apparent success.

Broader tool-use coverage. Alongside the failure-focused structures, we sample additional structural requirements whose purpose is to broaden the kinds of benign tool use represented in the corpus. When selected, each requirement is added to the scenario-generation prompt as a high-level constraint; compatible requirements may occur together, while the generating model chooses the concrete domain, tools, entities, and interaction that realize them.

Read-and-report scenarios are requested to be read-only tasks whose answer depends on several dependent tool calls rather than a single lookup, for example by requiring information to be retrieved, combined, compared, or aggregated across multiple outputs. Clarification scenarios are constructed so that one user-specific value required for completion is absent from the request and cannot be recovered from the world state or available tools. The model must therefore make any progress that is possible and ask the user for the missing value; once it is supplied, the same task continues to completion.

Derived-value scenarios require a later answer or tool argument to be computed from quantities returned by earlier tools rather than copied directly from an observation. For action tasks, the computed value must be used in a subsequent tool call; for read-only tasks, it must appear in the final answer. The expected value is recomputed independently during verification. Eligible scenarios can additionally span multiple interaction turns with a changing tool menu: an initial set of tools supports discovery or preparation, after which an additional capability becomes available while preserving the same world state and user goal.

Sampling and structural coverage. The structural requirements are sampled before scenario generation and determine which high-level constraints are included in the generation prompt. Compatible requirements can be active in the same scenario. Their sampling probabilities therefore control how often a structure is requested, not how often it appears in the final training corpus: after generation, the scenario must still be solved successfully and pass the common execution-based and structure-specific verification pipeline.

Different structures survive this pipeline at different rates. To calibrate the sampling probabilities, for each structure and each model family we run 50 pilot generations with that structure activated while the other compatible structures follow their normal sampling procedure, and measure the fraction ultimately retained after solving and verification. We use these empirical keep rates to choose one set of sampling probabilities, shared across model families, that gives the active structures a reasonably balanced presence in the retained corpora.

Surfaced guidance and recovery/completion are sampled with probability 0.40, follow-through with 0.35, read-and-report and clarification with 0.25 each, and derived-value tasks with 0.15.

The changing-tool-menu structure is applied whenever an eligible scenario is generated. Within surfaced-guidance cases, injection-like surface phrasing is sampled with probability 0.30.

Diversity across generated scenarios. The structural requirements specify properties of the interaction rather than complete task templates. To vary their concrete realization, the scenario-generation prompt is additionally varied along several dimensions independently of the selected structures. We sample cues for the application domain, concrete entities and attributes, tool capabilities, and the dependency shape connecting successive operations. These cues guide generation but do not fix the resulting task: the model may replace, combine, or extend them while satisfying the requested structural constraints.

As a result, the same high-level structure can be instantiated with different domains, records and entities, tool menus, and patterns of dependency between calls. For example, the dependency cues can induce sequential retrieval, information gathered from multiple sources, transformations of intermediate values, or verify-then-act workflows without tying any of these forms to a particular application domain. Scenario generation uses temperature 0.9.

Solve and validate. A separate call to the same frozen model receives the generated user request and model-visible tool schemas and attempts to complete the task through its standard tool-use interface using greedy decoding. Each emitted tool call is executed against the shared world state, its actual output is returned to the model, and state mutations persist across subsequent calls. The rollout continues until the model produces a final answer or reaches the execution limit.

Generated scenarios are filtered before entering the training corpus. Tool implementations and schemas must be executable and internally consistent, and the resulting trajectory must satisfy its generated success criterion. For state-changing tasks, the success check evaluates the resulting world state and is required to fail on the initial state and pass on the final state, preventing already-satisfied tasks from being retained. For read-only tasks, completion is instead verified from the model's final answer. Scenarios that cannot be executed consistently or remain incomplete after rollout are discarded. Structure-specific checks additionally verify that the dependency requested during generation is actually realized in the resulting task.

Attack generation. Attack generation is performed only after a clean trajectory has been successfully solved and verified. For each retained scenario, a separate generation step produces a concrete adversarial objective appropriate to the scenario and, when needed, plausible in-domain attacker tools that make the objective executable. Any attacker tools are included in both the clean and injected contexts, so their presence alone does not distinguish the injected example.

Injection points are sampled from tool-output occurrences that actually appear in the verified trajectory. For each selected output, the injection writer receives the clean tool output, the adversarial objective, the available tool schemas, any attacker tools, and any injections already placed earlier in the same trajectory. It does not receive the user request, the complete solution, or the full task-solving trajectory. The writer generates the adversarial text and chooses its placement within the selected output, either embedding it in an existing string field or placing it before or after the original content. The requested placement is validated against the actual output and stored so that the same transformation can be reconstructed during training.

Attack wording is varied independently of the benign scenario across human-like, system- or workflow-like, and more open-ended forms. When multiple injections are generated for the same trajectory, later generations are encouraged to differ from earlier ones. Attack generation uses temperature 0.9.

Teacher targets and training data. Each retained example stores the verified clean trajectory together with its injection specifications. During training, the stored injections are materialized in their selected tool outputs, leaving the remainder of the trajectory unchanged. The teacher target is always obtained from the frozen model on the corresponding clean trajectory.

At each supervised model-generated token position, we retain the teacher's top-K next-token probabilities with $K = 256$ and aggregate the remaining probability mass into the residual bucket defined in Eq. (4). For each model, the final self-generated training corpus contains 10,000 retained trajectories.

Training configuration. The student is initialized from the same checkpoint as the frozen generator and teacher. We train it with LoRA applied to all linear layers, using a learning rate of 5×10^{-5} and batch size 16 for one epoch on the $10,000$ generated cases, with an injection ratio of 0.5. Training uses the injection-augmented self-distillation objective described in Section 5.2.

D Evaluation Details

D.1 Evaluation Protocol

We evaluate SecAlign++ and RAISED over three training seeds and report the mean and standard deviation across seeds. All other methods are evaluated once because they do not involve a stochastic training run.

D.2 Architectural diversity.

We deliberately selected models with substantially different architectures to test whether RAISED depends on a particular transformer design. Gemma-4-12B-it uses a dense architecture combining sliding-window and full-attention layers, whereas Nemotron-3-Nano-30B-A3B uses a hybrid architecture that combines attention, Mamba-style state-space layers, and mixture-of-experts (MoE) blocks. Qwen3.8-27B provides a third configuration, interleaving linear-attention and full-attention layers. This diversity allows us to evaluate whether the proposed self-distillation objective transfers across different sequence-modeling mechanisms and parameter-routing strategies, rather than being specific to a single model architecture.

D.3 General-Knowledge Evaluation

We evaluate general knowledge and related capabilities to measure whether the defense methods preserve broad knowledge and general problem-solving capabilities.

LM Evaluation Harness. We use the LM Evaluation Harness Gao et al. (2021) for all reported general-knowledge evaluations. For each benchmark, we report the group-level metric produced by LM Evaluation Harness. Group metrics are micro-averaged over evaluation samples, such that every question, prompt, or programming problem receives equal weight.

MMLU. MMLU Hendrycks et al. (2020) evaluates broad knowledge and reasoning through multiple-choice questions spanning 57 subjects across the humanities, social sciences, STEM, and professional domains. We report accuracy, where a response is correct when the answer option assigned the highest likelihood matches the reference option.

MMLU-Pro. MMLU-Pro Wang et al. (2024b) evaluates more challenging, reasoning-intensive multiple-choice questions organized into broad subject categories. We report exact-match accuracy after applying the benchmark's answer-extraction procedure to the generated response.

Multilingual MMLU. Multilingual MMLU (OpenAI, 2025) evaluates knowledge and reasoning across multiple languages and subject areas using multilingual versions of MMLU-style multiple-choice questions. We report accuracy, where a response is correct when the answer option assigned the highest likelihood matches the reference option.

GPQA Diamond. GPQA Diamond Rein et al. (2023) is a particularly challenging subset of GPQA consisting of expert-written questions in biology, physics, and chemistry that are designed to require domain expertise and resist superficial answer retrieval. We report exact-match accuracy using the benchmark's flexible answer-extraction procedure, which extracts the selected answer option from the generated response.

AGIEval English. AGIEval English Zhong et al. (2024) evaluates general and professional reasoning on English-language examination questions drawn from several standardized and professional examinations. We report accuracy, where a response is correct when the predicted answer matches the reference answer.

IFEval. IFEval Zhou et al. (2023) evaluates whether a model follows objectively verifiable instructions that impose constraints on the form or content of its response. We report prompt-level strict accuracy, under which a response is correct only if it satisfies every verifiable instruction associated with the prompt.

GSM8K. GSM8K Cobbe et al. (2021) evaluates multi-step arithmetic reasoning on grade-school mathematics word problems. We report strict exact-match accuracy after extracting and normalizing the final numeric answer from the generated response.

Minerva Math. Minerva Math Lewkowycz et al. (2022) evaluates mathematical problem solving on technical problems spanning multiple mathematical subjects. We report mathematical-verification accuracy, which considers a response correct when its extracted final answer is mathematically equivalent to the reference answer.

HumanEval-Instruct. HumanEval-Instruct Chen et al. (2021) evaluates instruction-following code generation on programming problems whose generated solutions are executed against unit tests. We report pass@1, where a generated solution is correct if it passes all unit tests for the problem.

MBPP-Instruct. MBPP-Instruct Odena et al. (2021) evaluates instruction-following code generation on short programming problems with test cases defining the expected behavior. We report pass@1, where a generated solution is correct if it passes all test cases for the problem.

D.4 Agentic-Capability Evaluation

We evaluate agentic capabilities to measure whether the defense methods preserve the models' ability to use tools and complete tasks in interactive environments. For each agentic-capability benchmark, we report the micro-averaged success rate over its elementary evaluation units, such that every conversation, multi-turn entry, workplace task, mission, or subtask receives equal weight.

ToolTalk. ToolTalk Farn and Shin (2023) evaluates an assistant's ability to use APIs correctly in multi-turn conversations. Conversations require the model to select appropriate tools, provide the required arguments, and execute sequences of tool calls as the interaction evolves; the benchmark includes 28 tools grouped into seven application domains. We report conversation success rate. A conversation is successful when the model recalls every reference tool call and makes no incorrect action-tool call over the complete conversation.

BFCL Multi-Turn. The Berkeley Function-Calling Leaderboard (BFCL) Patil et al. (2025) evaluates function-calling capabilities across a broad range of tool-use settings. We evaluate its `multi_turn_base` subset, in which each entry may require multiple turns and multiple tool calls against a stateful environment. We report entry-level accuracy: an entry is correct only if the model's tool calls across all turns satisfy BFCL's official multi-turn checker, including selection of the required functions and construction of valid arguments.

WorkBench. WorkBench Styles et al. (2024) evaluates whether language-model agents can complete realistic workplace tasks by interacting with software tools in a sandboxed environment. Tasks include activities such as sending emails and scheduling meetings and often require planning and executing multiple tool calls that modify the underlying environment state. We report task-completion accuracy. A task is successful when the model's executed actions leave the sandbox in the reference final state.

Multi-Mission Tool Bench. Multi-Mission Tool Bench Yu et al. (2025) evaluates agents on test cases containing sequences of interrelated missions whose requirements evolve over the course of a conversation. Missions may require single- or multi-tool execution, clarification of missing parameters, or direct responses, and later missions can depend on information established earlier in the interaction. We report parameter-inclusive mission accuracy. A mission is successful when its tool calls follow a valid path in the benchmark's dynamic decision tree and their arguments satisfy the official parameter checker.

AsyncTool. AsyncTool Shi et al. (2026) evaluates tool-use agents in asynchronous multi-task settings in which tool responses may be delayed. The agent must coordinate several concurrent tasks, track dependencies and intermediate states, and continue making progress on other tasks while awaiting tool results. We report final subtask accuracy. A subtask is successful when both its tool-call trajectory and its resulting environment configuration are correct according to the official scorer.

D.5 Agentic Prompt-Injection Evaluation

Agentic prompt-injection evaluation is central to this work, as it measures whether a model can resist malicious instructions embedded in tool outputs while still completing the user's intended task. We evaluate this security–utility trade-off using benchmarks that combine interactive tool use, adversarially injected instructions, and task-level success criteria. For each benchmark, we follow its official aggregation convention. InjecAgent and AgentDojo aggregate over their benchmark-defined evaluation cases, so that each test case (or sampled attack, where applicable) receives equal weight. AgentDyn instead first computes each metric separately for its Shopping, GitHub, and Daily Life suites and reports the unweighted mean of the three suite-level scores.

InjecAgent. InjecAgent Zhan et al. (2024) evaluates the vulnerability of tool-integrated agents to indirect prompt injections embedded in external content such as emails or webpages. Its 1,054 test cases span 17 user tools and 62 attacker tools, with malicious instructions designed either to cause direct harm to the user or to exfiltrate private information observed by the agent during task execution. The agent must continue carrying out the benign user request while ignoring the injected objective. We report total attack success rate separately for the Base and Enhanced settings. In direct-harm cases, an attack succeeds when the agent calls the attacker's target tool; in data-stealing cases, it succeeds when the agent completes both stages and supplies the targeted private information to that tool.

AgentDojo. AgentDojo Debenedetti et al. (2024) evaluates agents on realistic tool-use tasks in domains such as workspace management, online banking, travel booking, and messaging. Prompt injections are placed in untrusted data returned by tools, for example in emails, documents, or other environment content, and attempt to redirect the agent toward a separate attacker-specified goal, such as leaking information or performing an unauthorized action. The benchmark jointly measures whether the agent completes the original user task and whether the injected attacker objective succeeds. We report clean utility (U), the fraction of benign user tasks completed without an attack; utility under attack (U_{atk}), the fraction of security cases in which the original user task is completed; and targeted attack success rate (ASR), the fraction of security cases in which the attacker's objective is achieved.

AgentDyn. AgentDyn Li et al. (2026) extends agentic prompt-injection evaluation to more open-ended and dynamically planned tasks across Shopping, GitHub, and Daily Life scenarios. Its tasks require longer-horizon interaction and may contain legitimate third-party instructions that the agent should follow, making it necessary to distinguish helpful external instructions from malicious injections. Adversarial instructions are embedded in third-party content encountered during the trajectory and attempt to divert the agent from the user's intended objective while it continues interacting with the environment. We report clean utility (U), the fraction of user tasks completed without an attack; utility under attack (U_{atk}), the fraction of attacked cases in which the original user task is completed; and attack success rate (ASR), the fraction of attacked cases in which the malicious objective is achieved.

D.6 Qwen: Additional Low-ASR Stress Test

Qwen provides an additional stress test in a different robustness regime from the primary experiments. Its Vanilla checkpoint already exhibits low attack success rates on AgentDojo and AgentDyn, allowing us to test whether RAISED remains effective when the starting model is already comparatively resistant to prompt injection. We evaluate the same three dimensions as in the main paper: prompt-injection robustness, benign agentic capabilities, and general capabilities.

Table 8: **Prompt-injection robustness and utility on Qwen.** Higher is better for utility (U) and utility under attack (U_{atk}); lower is better for attack success rate (ASR). Trained defenses report mean $\pm$ standard deviation over three seeds.

		InjecAgent ASR ↓		AgentDojo			AgentDyn		
Model	Method	Base	Enhanced	$U\uparrow$	$U_{\mathrm{atk}}\uparrow$	ASR↓	$U\uparrow$	$U_{\mathrm{atk}}\uparrow$	ASR↓
Qwen	Vanilla	0.0	0.0	88.7	80.0	5.8	66.7	64.8	4.2
	Prompt Sandwiching	0.0	0.0	89.7	82.6	3.8	73.3	68.2	1.8
	Spotlighting	0.0	0.0	89.7	85.4	0.9	73.3	67.9	0.5
	PromptGuard2	0.0	0.0	88.7	58.6	2.7	66.7	47.7	3.3
	SecAlign++	0.0±0.0	0.0±0.0	88.7±1.7	84.2±1.6	1.9±0.1	70.0±1.4	67.0±0.1	1.5±0.2
	RAISED (Ours)	0.1±0.0	0.0±0.0	87.3±0.5	84.1±0.1	0.9±0.4	68.3±1.4	65.1±0.3	0.1±0.2

Despite the already low ASR of the Vanilla checkpoint, RAISED further reduces attack success on both AgentDojo and AgentDyn. The improvement is therefore not restricted to models that are initially highly vulnerable to prompt injection, and persists in a low-ASR regime.

Table 9: **Benign agentic capabilities on Qwen.** Results on five tool-use benchmarks without prompt injection. Trained defenses report mean $\pm$ standard deviation over three seeds.

Model	Method	ToolTalk	BFCL	WorkBench	MMT-Bench	AsyncTool
Qwen	Vanilla	66.7	50.0	83.0	50.5	68.1
	Prompt Sandwiching	67.9	46.0	84.5	48.8	68.4
	Spotlighting	69.2	47.0	82.3	49.3	68.9
	PromptGuard2	66.7	50.0	83.0	50.5	68.1
	SecAlign++	65.4±0.0	49.3±0.6	84.0±0.5	47.9±0.5	68.7±0.1
	RAISED (Ours)	66.2±0.6	50.0±0.4	83.1±0.6	51.2±0.2	67.8±0.4

The additional robustness is not obtained by reducing benign tool-use capability. RAISED remains close to the Vanilla checkpoint across all five agentic benchmarks, confirming the preservation trend observed on Gemma and Nemotron.

Table 10: **General capabilities on Qwen.** Results on ten non-agentic benchmarks. Trained defenses report mean $\pm$ standard deviation over three seeds.

Model	Method	MMLU	MMLU-Pro	MMMLU	GPQA	AGIEval	IFEval	GSM8K	Minerva	HumanEval	MBPP
Qwen	Vanilla	76.5	85.0	58.6	77.6	36.5	94.7	96.1	93.6	96.3	81.6
	SecAlign++	74.4±0.3	85.0±0.2	54.7±0.7	79.5±2.3	35.1±0.3	94.3±0.5	96.1±0.1	90.0±1.4	96.5±0.3	81.6±0.6
	RAISED (Ours)	76.7±0.1	85.1±0.1	58.8±0.1	74.2±1.1	37.1±0.5	94.1±0.6	96.4±0.2	93.6±0.1	97.0±0.0	81.3±0.2

Finally, the general-capability evaluation shows the same preservation pattern. RAISED remains close to the Vanilla checkpoint across the benchmark suite while achieving the additional robustness reported above. Taken together, the Qwen results show that RAISED remains effective even when applied to a model that already starts from a low-ASR regime.

D.7 Looking Inside the Model

RAISED trains the model to match its clean-context behavior when an injection is added. We therefore ask whether the trained model shows less internal sensitivity to the injection while remaining close to the base model on clean inputs.

We evaluate this on 88 clean/injected context pairs from AgentDojo. For each pair, we run the base model on the clean context to obtain its continuation $y^\star$, then compare model states while reading the clean and injected contexts followed by the same continuation. At each of the 48 decoder blocks, we measure the injection-induced change in the hidden state, as well as the distance between each trained model and the base model on the clean context.

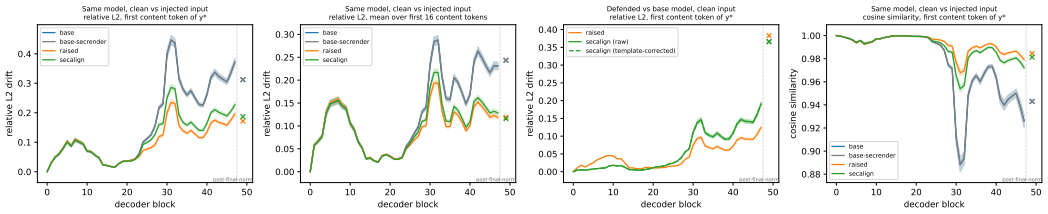


Figure 7: Per-block hidden-state changes over 88 AgentDojo pairs (Gemma-4-12B). Panels 1, 2, and 4 measure the effect of the injection within each model; panel 3 measures the distance from the base model on clean contexts. Bands show s.e.m. across pairs. $\times$ at block 48 denotes the state after the final norm.

Table 11: Summary of internal-state changes (88 AgentDojo pairs) and of the behavioral check (72 pairs, see text).

	Base	SecAlign++	RAISED
Starts the injected tool call (%, 72 pairs) ↓	62.5	13.9	12.5
KL to the base model's clean prediction (nats, 88 pairs) ↓	0.338	0.062	0.011
Injection effect at block 31, peak (rel. L_2) ↓	0.45	0.29	0.23
Distance to base on clean context, block 47 ↓	–	0.19	0.12

The injection produces a similar pattern across models in the early blocks, followed by a peak around block 31. At that block, the relative L_2 change is 0.45 for the base model, compared with 0.23 for RAISED. This reduction goes with the behavioral check: RAISED starts the injected tool call in 12.5% of the checked pairs, versus 62.5% for the base model.

RAISED also remains closer to the base model on clean contexts than SecAlign++ (0.12 vs. 0.19 at block 47). Thus, the internal-state measurements are consistent with the intended behavior of the objective: reduce sensitivity to injected content while limiting changes to clean-context behavior.

D.8 SecAlign++ training-step trade-off

The SecAlign++ results reported in the main paper use checkpoints obtained after 313 training steps. The effect of extending training differs across model families. On Gemma and Qwen, further training primarily reduces utility without materially improving robustness. On Nemotron, it further reduces attack success but comes with substantial utility degradation. We use Nemotron below as a representative illustration because the trade-off is particularly clear. We compare the 313-step checkpoint used in the main paper with the same model trained for 625 steps, and include both the Vanilla checkpoint and RAISED as reference points.

Table 12 shows that additional SecAlign++ training continues to improve robustness. On AgentDojo, ASR decreases from $4.0 \pm 0.7\%$ at 313 steps to $0.7 \pm 0.0\%$ at 625 steps, while on AgentDyn it decreases from $7.8 \pm 0.3\%$ to $3.0 \pm 0.2\%$. Thus, considering the security benchmarks alone would suggest that further training is beneficial.

This security improvement, however, does not correspond to a uniformly better checkpoint. Table 13 shows the same models on the general-capability evaluation. While the 313-step checkpoint already departs from Vanilla on some tasks, extending training to 625 steps amplifies this drift on several benchmarks. In particular, MMLU decreases from 48.3 ± 0.7 to 42.8 ± 0.3, GPQA from 43.8 ± 1.7 to 39.4 ± 1.1, Minerva Math from 75.0 ± 0.1 to 66.1 ± 0.9, and MBPP from 64.6 ± 1.0 to 13.4 ± 2.7. By contrast, RAISED remains close to the Vanilla checkpoint across the same evaluation suite.

The benign agentic benchmarks show a milder version of the same effect. For Nemotron, extending SecAlign++ training from 313 to 625 steps reduces ToolTalk from 43.2 ± 0.6 to 42.7 ± 0.6, BFCL from 33.0 ± 0.4 to 31.8 ± 0.8, MMT-Bench from 43.4 ± 0.3 to 41.9 ± 0.3, and AsyncTool from 34.5 ± 0.3 to 32.3 ± 0.8, while WorkBench increases. The main effect of prolonged training is

Table 12: **Effect of extended SecAlign++ training on prompt-injection robustness for Nemotron.** SecAlign++ is evaluated at the 313-step checkpoint used in the main paper and after 625 training steps. Vanilla and RAISED are included as reference points. Higher is better for utility (U and U_{atk}); lower is better for ASR.

		InjecAgent ASR ↓		AgentDojo			AgentDyn		
Method	Steps	Base	Enhanced	U ↑	U_{atk} ↑	ASR↓	U ↑	U_{atk} ↑	ASR↓
Vanilla	–	26.7	33.7	48.5	39.9	11.7	25.0	19.6	17.6
SecAlign++	313	6.4±0.9	8.8±1.4	50.2±1.0	47.7±0.7	4.0±0.7	22.2±0.8	20.7±0.5	7.8±0.3
SecAlign++	625	1.9±0.2	1.4±0.5	50.2±1.0	49.5±0.9	0.7±0.0	22.2±0.8	21.2±1.2	3.0±0.2
RAISED (Ours)	–	3.5±0.5	1.0±0.6	50.2±1.3	46.4±0.6	0.5±0.1	25.0±0.0	25.4±1.2	1.9±0.6

Table 13: **General capabilities under extended SecAlign++ training on Nemotron.** The 313-step SecAlign++ checkpoint is the one used in the main paper. Bold values highlight the largest capability degradations observed after extending SecAlign++ training to 625 steps. Vanilla and RAISED are included as reference points.

Method	Steps	MMLU	MMLU-Pro	MMMLU	GPQA	AGIEval	IFEval	GSM8K	Minerva	HumanEval	MBPP
Vanilla	–	49.5	60.1	34.6	42.4	23.3	81.3	87.2	76.5	87.2	67.8
SecAlign++	313	48.3±0.7	60.3±0.2	34.9±0.0	43.8±1.7	23.1±0.2	79.4±1.5	88.2±0.5	75.0±0.1	86.0±0.0	64.6±1.0
SecAlign++	625	**42.8±0.3**	60.8±0.1	34.8±0.1	**39.4±1.1**	21.3±0.3	79.1±0.8	87.2±0.3	**66.1±0.9**	86.8±0.6	**13.4±2.7**
RAISED (Ours)	–	49.6±0.1	59.3±0.2	35.1±0.0	41.2±2.1	23.3±0.3	80.0±0.3	87.2±0.3	76.7±0.1	86.6±0.5	66.2±0.6

therefore not a uniform collapse of agentic performance, but a broader and increasingly uneven drift away from the capabilities of the original checkpoint.

These results explain our choice of the 313-step SecAlign++ checkpoint for the main comparison. SecAlign++ can be trained longer to obtain lower ASR, but selecting a checkpoint solely according to attack success can conceal substantial degradation elsewhere. The same qualitative training-duration trade-off is observed for Gemma and Qwen, although the magnitude and affected capabilities differ across models. Nemotron provides a particularly clear illustration of this phenomenon. In contrast, RAISED reaches low attack success while remaining substantially closer to the capability profile of the original model.

E Ablation Study

The main comparison preserves SecAlign++'s original fine-tuning data source, preference-data construction, and input representation, providing the appropriate method-level comparison. However, RAISED differs from SecAlign++ in both its training objective and training data.

Controlled comparison. As an alternative to our self-generated data, we use trajectories from Glaive Function Calling v2, a synthetic function-calling dataset containing tool specifications and multi-turn interactions. For each clean Glaive trajectory, we construct an injected counterpart by taking an instruction from a different Glaive example and inserting it into a randomly selected tool output. The injected instruction is therefore unrelated to the original user task, while the underlying trajectory remains unchanged.

To isolate the training objective, we hold the trajectories, injection placements, and rendered inputs fixed and compare SFT, DPO, and self-distillation. For DPO, we follow the preference construction of SecAlign++. The frozen base model generates the preferred response from the clean Glaive context and the rejected response separately from the injection itself, mirroring SecAlign++'s construction of preferred and rejected responses from the clean input and injected instruction, respectively. Both responses are then evaluated under the resulting injected context by the DPO objective.

To isolate the effect of the training data, we then hold the objective fixed and compare Glaive trajectories with the self-generated trajectories used by RAISED, for both DPO and self-distillation.

Effect of the training objective. Holding the Glaive data fixed, self-distillation preserves substantially more benign agentic utility than SFT or DPO while maintaining low ASR. On AgentDojo,

Table 14: **Ablation of training data and objective on Gemma-4-12B-it.** Higher is better for benign utility (U); lower is better for attack success rate (ASR). Trained variants report mean±standard deviation over three seeds.

		AgentDojo		AgentDyn	
Data	Objective	$U \uparrow$	ASR↓	$U \uparrow$	ASR↓
–	Vanilla	84.5	39.5	61.7	53.4
Glaive	SFT	73.2±3.6	0.5±0.3	17.5±3.5	1.5±1.2
Glaive	DPO	79.1±0.6	1.8±0.1	48.6±0.3	2.3±1.5
Glaive	Self-distillation	80.1±0.6	1.1±0.2	55.5±0.8	2.1±1.3
Self-generated	DPO	82.2±0.5	1.0±0.1	55.6±3.4	1.6±1.4
Self-generated	Self-distillation (RAISED)	83.5±0.8	1.0±0.3	60.0±2.4	1.6±1.8

self-distillation reaches 80.1 utility, compared with 79.1 for DPO and 73.2 for SFT. The difference is larger on AgentDyn, where self-distillation reaches 55.5, compared with 48.6 for DPO and 17.5 for SFT. SFT attains slightly lower ASR than self-distillation, but at a much larger utility cost, while self-distillation also substantially improves robustness over DPO on AgentDyn.

The self-generated data provide a second controlled comparison between DPO and self-distillation. Both reach an AgentDojo ASR of 1.0 and an AgentDyn ASR of 1.6 within the reported variation, while self-distillation increases benign utility from 82.2 to 83.5 on AgentDojo and from 55.6 to 60.0 on AgentDyn. Thus, with the data and inputs held fixed, clean-context distribution matching preserves more of the base model's agentic behavior than the DPO objective used by SecAlign++.

Effect of self-generation. Holding the objective fixed shows an additional utility benefit from the self-generated trajectories. With DPO, replacing Glaive with self-generated data increases benign utility from 79.1 to 82.2 on AgentDojo and from 48.6 to 55.6 on AgentDyn, while reducing ASR from 1.8 to 1.0 and from 2.3 to 1.6, respectively.

Self-distillation shows the same utility trend. Replacing Glaive with self-generated trajectories increases utility from 80.1 to 83.5 on AgentDojo and from 55.5 to 60.0 on AgentDyn, while maintaining low ASR. The benefit of self-generation therefore remains when the training objective is held fixed.

Role of the structural patterns. To test whether the gain from self-generated data comes from self-generation alone or from the way generation is steered, we repeat the experiment with the high-level structural patterns disabled while keeping the rest of the generation pipeline unchanged. Benign utility falls from 83.5 to 80.8±0.4 on AgentDojo and from 60.0 to 56.7±2.1 on AgentDyn, close to the 80.1 and 55.5 obtained with Glaive trajectories under self-distillation. This indicates that the benefit of self-generation comes largely from targeting scenarios that exercise the behaviors we aim to preserve, rather than from synthetic data generation alone.

Summary. The ablation supports both components of RAISED. With identical data and inputs, self-distillation preserves more agentic utility than SFT or the DPO objective used by SecAlign++ at similarly low attack success. Holding the objective fixed, the self-generated trajectories provide a further utility gain. RAISED combines these two effects.

F Adaptive attacks

Our threat model considers static indirect prompt injections, in which the adversarial content is chosen independently of the defended model (Section 3.2). This setting matches the evaluation protocol of InjecAgent, AgentDojo, and AgentDyn, and is the setting for which the robustness claims in this work are made. In contrast, an adaptive attacker repeatedly queries a particular defended model and optimizes the injected content against its observed behavior. Such attacks grant the adversary an additional capability and evaluate a distinct question: whether a defense remains robust when the

adversary can specialize its attack to that defense. We nevertheless investigated this stronger setting using PISmith (Yin et al., 2026), a recent black-box framework that optimizes prompt injections against the target model. Using the authors' released code and protocol, however, the optimization reward remained identically zero for most model–defense configurations. Consequently, the optimizer received no signal with which to distinguish or improve candidate attacks. This is consistent with the reward-sparsity regime discussed by PISmith, but, importantly, a failed optimization run is not evidence of adaptive robustness. We therefore do not report these runs as a robustness result. Obtaining a more informative signal would require modifying PISmith's protocol, which falls outside the scope of this work. We thus restrict our empirical robustness claims to the static threat model defined in Section 3.2. Robustness to model-adaptive prompt injection remains an open question.

G Limitations

Model scale and diversity. We evaluate clean-context self-distillation on three model families with different tokenizers, post-training recipes, and baseline robustness. The consistency of the effect across Gemma, Nemotron and Qwen suggests that it is not specific to one family, and the objective itself makes no architecture-specific assumption. We have not yet evaluated additional model families or frontier models in the trillion-parameter regime, so whether the same security–utility behavior persists at that scale remains an empirical question.

Dependence on clean-model behavior. Clean-context self-distillation preserves the model's own behavior; it does not improve its correctness. If the clean model selects the wrong tool, follows a poor plan, hallucinates, or otherwise behaves undesirably, the distillation objective will preserve that behavior under injection rather than correct it. The quality of the defended model is therefore bounded by the quality of the clean-context teacher. The method should be understood as a defence against adversarial context influence, not as a general alignment or safety intervention.

Cost of soft self-distillation. Soft self-distillation requires an offline teacher pass and storage of truncated predictive distributions. For our 10,000-trace corpus, storing the top-256 teacher distributions requires approximately 15 GB per model, and teacher generation completes in well under one hour using eight parallel inference workers on our GB200 cluster. At frontier scale, the teacher pass becomes substantially more expensive: a model such as Kimi K3 activates roughly 104B parameters per token, approximately $4\times$ to $9\times$ the active parameter count of the dense models evaluated here.

H Broader Impact

Indirect prompt injection is a practical obstacle to deploying language-model agents over untrusted external content. Clean-context self-distillation may improve the reliability of agents operating over web pages, documents, emails, APIs, and other external data sources while retaining their useful tool-use behavior. The method should be viewed as a defence-in-depth mechanism that complements access control, sandboxing, capability restrictions, and other system-level safeguards rather than replacing them.