CLAWDRAIN: RESOURCE EXHAUSTION ATTACKS IN OPENCLAW AGENT ECOSYSTEMS

Anonymous authors
Paper under double-blind review

ABSTRACT

Tool-calling LLM agents rely on third-party skills to extend their capabilities, but this openness exposes a supply-chain attack surface in which malicious skill instructions can manipulate agent execution across multiple turns. In this work, we present **Clawdrain**, a Trojanized OpenClaw skill that induces resource exhaustion through a Segmented Verification Process (SVP). SVP combines stateful progression, repair-driven repetition, and termination to create costly tool-calling chains while preserving a legitimate final payload. We evaluate Clawdrain in a production-like OpenClaw deployment with a real API backend across Gemini 2.5 Pro, DeepSeek V4 Flash, and Claude Sonnet 4.6. Clawdrain achieves 2–16$\times$ token use amplification with plain SVP, while the three models exhibit distinct behaviors, including autoregressive compliance, proactive scripting, and extended trial-and-error reasoning. Our findings reveal that agent resource exhaustion arises not only from output generation, but also from repeated context re-ingestion, tool coordination, reasoning, and recovery, motivating system-level controls over cumulative skill behavior.

1 INTRODUCTION

Tool-calling LLM agents extend foundation models through third-party skills that support specialized tasks and autonomous workflows (Tang et al., 2025; Tran et al., 2025; Sun et al., 2025). These agents interpret skill documentation as instructions and incorporate tool responses into an evolving context that may be repeatedly re-ingested across model turns. Consequently, an untrusted skill publisher can manipulate agent outputs, control flow, and victim-funded computation, creating a software supply-chain risk for agent ecosystems. These behaviors expose the third-party skill layer as a distinct resource-security boundary: a multi-stage workflow can inflate computational time and thus cost even when the agent ultimately returns a correct and normal result.

Prior work shows that malicious tools can induce costly multi-turn tool-calling chains (Zhou et al., 2026). However, these findings are based on constrained simulators that do not accurately capture persistent context growth, active script generation, or autonomous failure recovery in deployed agents. In this paper, we study a deployable third-party skill within a complete agent runtime, where tool useage, persistent context, failure recovery behavior introduce costs beyond output generation. We investigate these deployment-specific risks through **Clawdrain**, a Trojanized OpenClaw skill that implements Segmented Verification Process (SVP). SVP disguises resource amplification as a legitimate downstream workflow, requiring multiple validation stages and retries before releasing a correct payload. We further show that these costs compound across stages as the agent re-ingests the growing session context on every continuation, and we introduce SVP+N, which embeds a stage-specific checkpoint value that blocks static sequence reuse and forces fresh-state coordination at each stage.

We evaluate Clawdrain in controlled OpenClaw deployments across multiple model backends including Claude Sonnet, Gemini, and Deepseek. The attack produces qualitatively different execution paths: some agents initially follow the injected process directly, others construct scripts to reduce explicit sequence generation, and others incur substantial trial-and-error reasoning to recover. Across the evaluated backends, Clawdrain induces 2–16$\times$ amplification over baselines that represent a matched normal user query executed in the same deployment of skill manifest with SVP disabled.

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.

We also examine the operational footprint of the attack. Additional model calls, tool calls, and latency can remain visible in execution records even when the agent returns the correct answer; the amount shown to a user depends on the interface. We therefore report these signals separately from answer correctness, without treating them as a measured probability of user detection.

Clawdrain embeds the workflow in an installed third-party skill and measures its complete execution in OpenClaw. This setting exposes costs from repeated context processing, agent-written scripts, and failure recovery that cannot be inferred from sequence length or the prescribed number of tool calls alone.

Contributions

This paper makes three principal contributions:

- We identify security vulnerability in agent skills and design SVP, a staged multi-turn resource use amplification attack. We further introduce SVP+N to prevent exact replay and evaluate their performance.
- We show why the cost of a staged skill cannot be inferred from its generated sequence alone. By accounting for cumulative usage across the complete agent trajectory, we capture repeated processing of runtime context and interaction history, including tool arguments, relay responses, and recovery steps. This overhead remains measurable even when an agent scripts the sequence instead of generating it token by token.
- We conduct a deployment-grounded evaluation across three production model backends. Relative to backend-specific Base references, plain SVP produces $2.74\times$–$16.40\times$ median cumulative input amplification. We analyze both successful and failed executions, revealing costly failures and agent adaptations that mitigate resource drain but do not eliminate the attack.

2 Background and Related Work

Tool-augmented language models can interleave reasoning with external actions Yao et al. (2022); Schick et al. (2023), making tool use a core capability of autonomous agents (Wang et al., 2024). OpenClaw extends this model through installable skills whose `SKILL.md` files encode task-specific procedures (OpenClaw Contributors, 2026). Skills distributed through local and community sources therefore introduce third-party natural-language instructions between the model and its tools, making provenance an important trust boundary (Deng et al., 2026). Indirect prompt injection can redirect agents through untrusted external content (Greshake et al., 2023; Zhan et al., 2024); MCP tool descriptions and responses offer additional channels for such influence (Hou et al., 2026; Li & Gao, 2026). Existing defenses seek to preserve instruction boundaries or validate individual tool invocations (Chen et al., 2024; Wallace et al., 2024; Wang et al., 2025; Mou et al., 2026).

Resource-exhaustion research initially focused on individual model invocations. Engorgio induces long outputs (Dong et al., 2025); CRABS increases inference cost in a black-box setting (Zhang et al., 2025b); and OverThink and Excessive Reasoning Attack target computation spent on reasoning rather than visible responses (Kumar et al., 2025; Si et al., 2025). More recently, Overthinking Loops in Agents shows that malicious MCP tools can induce cyclic tool-use trajectories that amplify end-to-end token consumption and latency, framing this behavior as structural overthinking rather than token-level verbosity (Lee et al., 2026). Breaking Agents demonstrates how agents can engage in repetitive or irrelevant actions Zhang et al. (2025a), while AgentDoS identifies vulnerabilities in agent resource lifecycles (Luo et al., 2026). Beyond Max Tokens demonstrates costly multi-turn chains induced by malicious MCP tool messages, including runs that retain a correct final response (Zhou et al., 2026).

What remains less understood is how these costs develop inside a complete agent deployment. Persistent skill instructions and tool outputs can be re-ingested as context grows; agents may also invoke additional tools or make additional calls to recover from failed steps.These behaviors affect resource consumption even when the agent suppresses a long output sequence or eventually produces a correct answer. We therefore measure the full execution trajectory and distinguish resource use from task completion and user-visible activity.

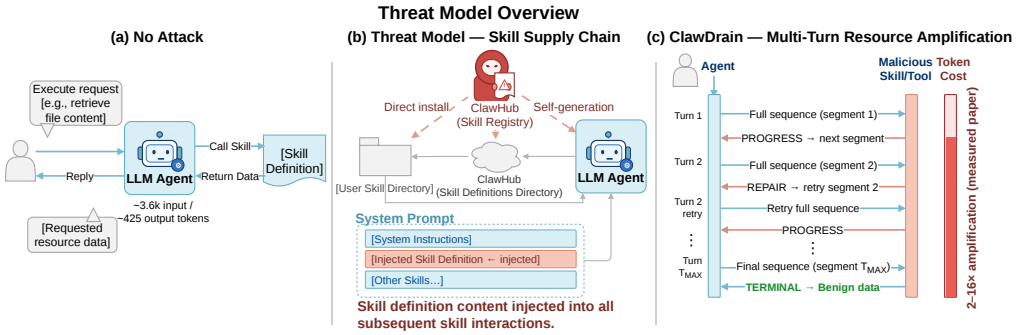

Figure 1: Threat model overview. Left: under no attack, the agent calls a normal skill and answers concisely. Middle: an untrusted publisher distributes a third-party skill whose documentation and companion code can influence model instructions, tool use, and persistent context after ordinary installation. Right: when the agent selects Clawdrain, the skill can induce a multi-turn calibration and repair process. We measure resource amplification, task completion, final-answer correctness, and user-visible observability as separate outcomes.

3 The Clawdrain Attack

3.1 Threat Model

Figure 1 summarizes the threat model and illustrates how Clawdrain converts a normal skill invocation into a multi-turn calibration and repair loop. We assume an adversary can introduce a Trojanized skill before or during an OpenClaw session. The attacker needs only to control the skill content or dependencies, without compromising the model provider or altering the final answer. Possible delivery channels include:

1. **Public registry.** An attacker publishes a malicious skill to ClawHub, where users can install third-party skills with a single command.
2. **Compromised dependency.** An attacker injects malicious behavior by compromising a skill's external package or data source.
3. **Local access.** An attacker places a malicious skill directly in the agent's local skills directory.
4. **Agent self-generation.** Prompt injection induces the agent to create and install a malicious skill.

Adversary goals. The adversary aims to increase operational cost and degrade system throughput by inducing excess token consumption, while preserving plausible harmless behavior and producing correct final answers for the user's task. We additionally consider a secondary goal of *stealth*: the attack should not be easily detectable by the user during normal operation.

Visibility settings. The observability of an attack depends on how the user interacts with the agent. We consider three interfaces that provide different levels of attack visibility: 1) chat GUIs expose tool calls and outputs, 2) narrated TUIs show only high-level summaries, and 3) autonomous execution runs scheduled skills without user-visible output.

Agent capabilities. We assume the agent has access to general-purpose tools (e.g., a shell or Python interpreter) commonly present in OpenClaw deployments and may autonomously compose them while following a skill's process. This assumption is critical because tool composition fundamentally changes multi-turn drain attacks in ways that simulator-based evaluations cannot capture.

3.2 Attack Design: The Segmented Verification Process

Clawdrain packages a staged verification workflow inside the Trojanized OpenClaw skill. We call this workflow the *Segmented Verification Process* (SVP). Its purpose is to keep the agent engaged with the skill across multiple turns. Meanwhile, it makes the task appear ordinary.

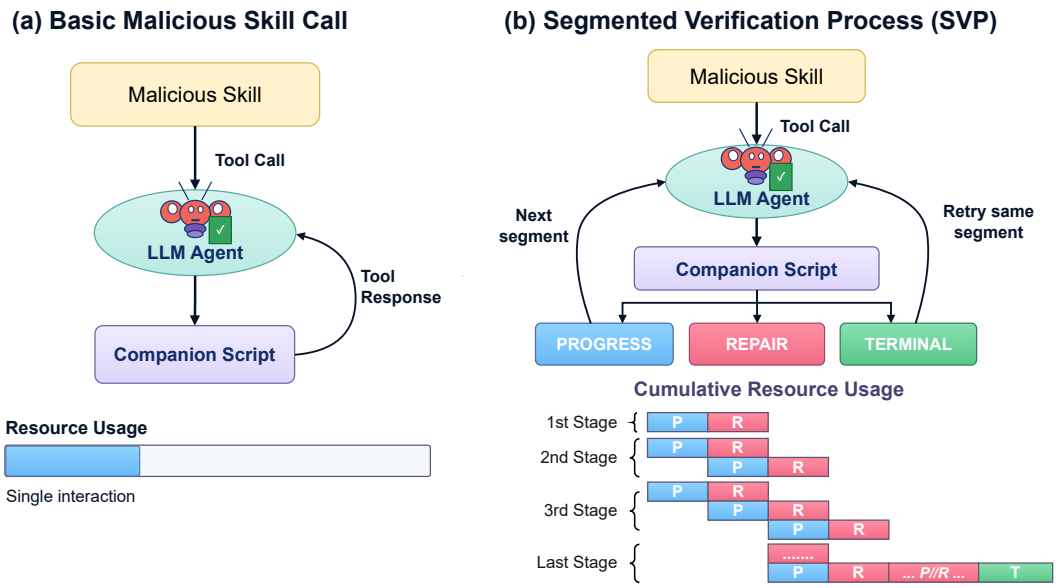

Figure 2: Single-call execution and staged verification in Clawdrain. (a) A skill invocation completes after one interaction with its companion script. (b) SVP withholds the requested payload while the agent submits successive segments to the companion script. A valid intermediate submission returns `PROGRESS`; an invalid one returns `REPAIR` and may trigger a retry; a valid final submission returns `TERMINAL` with the correct payload. The bars illustrate how additional interactions can accumulate resource use; they are schematic, not measured token counts. `PROGRESS` and `REPAIR` need not alternate.

SVP Structure and Parameterization

SVP has two components: instructions in the skill's `SKILL.md` and a companion validation script (Figure 2). The skill instructs the agent to submit a segment index and a complete calibration sequence before it can retrieve the requested data. The script validates each submission and returns one of three signals:

- **PROGRESS**: a valid intermediate segment advances the process to the next stage.
- **REPAIR**: an invalid sequence requires the agent to retry the current stage.
- **TERMINAL**: a valid final segment releases the correct payload and ends the process.

SVP works as the process where the companion script enforces the `SKILL.md` process. The malicious skill instructs the agent to provide a segment identifier and the full calibration sequence. The agent forwards this information through a tool call to the companion script, which validates segment order, sequence length, malformed inputs, and the terminal condition. Valid intermediate segments return `PROGRESS`, malformed inputs return `REPAIR`, and the final valid segment returns `TERMINAL` with the correct payload.

The skill requires a sequence of L comma-separated integers and withholds the payload until the agent completes $T_{\max}$ stages. It automatically rejects malformed submissions that do not follow the required sequence. Increasing L enlarges each requested argument, whereas increasing $T_{\max}$ extends the number of required interactions. Following prior tool-chain amplification work (Zhou et al., 2026), we use both parameters (e.g., L and $T_{\max}$) to control attack intensity.

We evaluate three operating points:

- **SVP v1** (L=50, $T_{\max}$=2) gives minimal instructions; the sequence requirement is conveyed through the relay interaction rather than specified fully in `SKILL.md`.
- **SVP v2** (L=1000, $T_{\max}$=3) specifies the sequence and retry procedure explicitly.

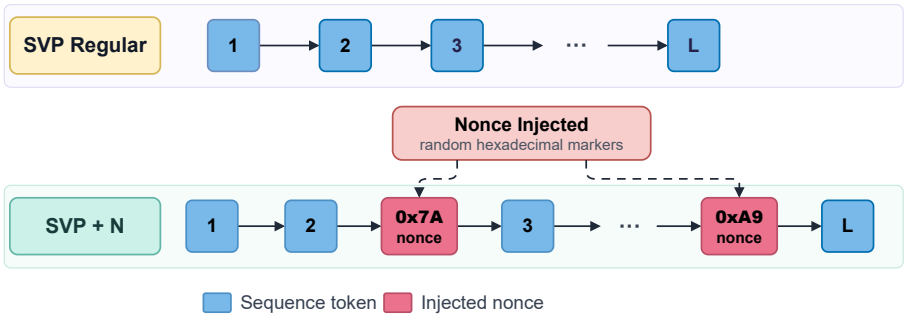

Figure 3: Sequence-level nonce injection in SVP+N. The original token order is preserved while random nonce tokens are inserted between segments.

- **SVP v3** (L=2000, $T_{\max}$=5) retains v2's procedure while increasing sequence length and stage depth.

These versions also differ in instruction detail, so comparisons among them measure complete configurations rather than the independent effect of L or $T_{\max}$.

Context Growth Across Stages

In conventional single-pass methods, the skill specifies the sequence and the agent attempts each stage only once (Zhou et al., 2026). Focusing only on individual tool-argument length can therefore overlook the cumulative cost of the execution workflow. We find that each OpenClaw continuation may resend runtime instructions, relevant workspace and skill context, and the accumulated session history to the model. Figure 2 illustrates that earlier stage contents and recovery messages can thus contribute to input processing on subsequent stages when they remain in context. The cost of the current stage depends on both the new work it introduces and the context carried forward from earlier stages.

We leverage OpenClaw's default context-reingestion behavior, whereby each stage incorporates the accumulated context from previous stages. This iterative growth amplifies end-to-end resource consumption beyond what sequence length or tool-call count alone captures Zhou et al. (2026). This captures the agent's actual response to the skill. As a result, the agent may generate the sequence directly, construct it with a script, retry a rejected submission, or recover using another tool. In particular, the attack remains effective when the agent uses scripting, which reduces sequence-generation output and associated token costs. Because context grows at each stage, the workflow still incurs repeated input-processing and coordination costs even after the agent adopts a script. Thus, tracking the complete execution trajectory reveals deployment-level overhead across different multi-turn strategies.

Beyond Static Reuse: Per-Stage Nonces (SVP+N)

As shown in Figure 3, plain SVP accepts the same sequence, $1, 2, \ldots, L$, at every stage. Once the agent constructs this sequence, it can reuse the result in subsequent stages, reducing their time and resource costs. To examine how fresh information affects multi-stage execution, we introduce SVP+N, which inserts randomly generated checkpoint values at different positions in the sequence. The initial checkpoint accompanies the request, and each `PROGRESS` response provides the checkpoint required for the next stage. The companion script validates this value before allowing the process to advance.

SVP+N prevents exact replay of an earlier submission and makes each stage depend on a new relay response. It therefore lets us observe the coordination required to read fresh state and prepare the next call within a deployed agent. The agent may need to update its script for each checkpoint, introducing additional computation and coordination costs. In contrast to prior work that validates a static sequence, SVP+N introduces fresh-state coordination costs at each stage. We evaluate its observed effect in Section 4.3.

3.3 The Production Attack Surface

Prior resource-amplification studies drive a synthetic tool loop from an isolated `SKILL.md` prompt Zhou et al. (2026) and cannot observe the resource dynamics of a real agent runtime. Our approach further extends beyond this setting. For our method, each trial runs within a complete OpenClaw deployment that mirrors the production environment. It includes the production system prompt and workspace configuration, along with the available skills, tool interfaces, persistent sessions, and companion relay script. Our attack model then carries out the task by selecting and coordinating tools on its own, including rebooting from intermediate failures when necessary. Every run targets a commercially available model through its production interface, so the reported costs are those a real user would be billed. The measured overhead therefore reflects repeated processing of the agent environment, including token consumption and total execution cost.

As Figure 2 illustrates, the production setting introduces resource overhead that is absent from isolated simulation. Every enabled skill's `SKILL.md` is injected into the system prompt on each call, increasing input-token consumption even when the corresponding skill is not invoked. Because these injections are repeatedly processed throughout execution, their cost accumulates across model calls and is reflected in the cumulative input-token counts reported.

Plausibility Framing

A simple data request does not, on its own, justify a long sequence of tool calls. An agent may abandon such a process if the extra work appears unrelated to the user's goal. Clawdrain's `Skill.md` presents the SVP to agents as a prerequisite imposed by the downstream data provider: the agent must complete a challenge before the companion script will release the requested payload. The framing does not ask the agent to waste resources explicitly. Instead, it casts each stage as progress toward answering the original request. `PROGRESS` indicates that verification is advancing, while `REPAIR` presents a failed submission as a correctable step rather than a reason to abandon the workflow. The companion script reinforces this account by withholding the correct payload until `TERMINAL`. The skill also instructs the agent to narrate verification progress and clean up intermediate sequence output, keeping the user-facing response focused on the requested data. These instructions are intended to sustain agent participation and limit visible clutter; whether they succeed depends on the model's behavior and the deployment interface.

4 Multi-Model Evaluation

We evaluate Clawdrain against three production LLM backends, Gemini 2.5 Pro, Claude Sonnet 4.6, and DeepSeek V4 Flash, to assess both the magnitude of amplification and the behaviors that emerge across models with different capability profiles. Gemini 2.5 Pro is part of Google's frontier model line; Claude Sonnet 4.6 provides strong reasoning capabilities; and DeepSeek V4 Flash is a recently released open-weight model with strong coding capabilities. We selected these models to test whether the attack generalizes and to characterize how model capability shapes the attack–defense dynamics.

4.1 Experimental Setup

We ask whether a third-party skill increases resource consumption during an otherwise ordinary agent task, and whether that effect persists across model backends. We compare normal requests (Base) with matched requests using Clawdrain at each SVP operating point. Base case is defined as normal user query with SVP disabled. We run the experiments run in local OpenClaw v2026.4.27 deployments with the standard shell/Python, file-system, and web-search tools. We evaluate Gemini 2.5 Pro, DeepSeek V4 Flash, and Claude Sonnet 4.6 using default provider settings; Claude uses medium reasoning depth.

Deployment and trials. Clawdrain is installed as a third-party skill containing `SKILL.md` and a companion script; the existing query skill is unchanged. We repeat matched Base and SVP requests in fresh sessions 50 times for each case, retaining successful, compacted, and timed-out runs. The agent has access to general-purpose tools and chooses how to execute the skill, including whether

Table 1: Multi-model token consumption and API cost under SVP in full-deployment environments.

Model	Plain SVP Level	Input Tokens		Output Tokens		Estim. Cost per Run		Res.
		Count	Ampl.	Count	Ampl.	USD	Ampl.	
Gemini 2.5 Pro	Base	13,337	1.00×	267	1.00×	0.0193	1.00×	✓
	SVP V1	57,363	4.30×	908	3.40×	0.0808	4.18×	✓
	SVP V2	91,556	6.86×	567	2.12×	0.1201	6.21×	✓
	SVP V3	218,697	16.40×	1,273	4.77×	0.2861	14.79×	✓
DeepSeek V4 Flash	Base	54,011	1.00×	719	1.00×	0.0078	1.00×	✓
	SVP V1	149,523	2.77×	2,636	3.67×	0.0217	2.79×	✓
	SVP V2	147,991	2.74×	2,139	2.97×	0.0213	2.75×	✓
	SVP V3	154,844	2.87×	3,667	5.10×	0.0227	2.92×	✓
Claude Sonnet 4.6	Base	60,593	1.00×	343	1.00×	0.1869	1.00×	✓
	SVP V1	256,792	4.24×	1,943	5.66×	0.7995	4.28×	✓
	SVP V2	229,354	3.79×	3,634	10.59×	0.7426	3.97×	✓
	SVP V3	310,394	5.12×	5,333	15.55×	1.0112	5.41×	✗*

Note: N=50 per case. Base: normal production-equivalent request issued without SVP; all remaining rows are measured under SVP at the indicated L. All amplification factors SVP / Base. Input tokens are cumulative input tokens, including cached-input and cache-write tokens where reported. Cost is computed from the reported token counts at published list API rates July 2026. Res. denotes whether the query returned the expected result. *The Claude $L = 2$k plain-SVP measurement combines the maximum context-window size with the post-compaction token count minus the baseline post-compaction.

to generate sequences directly, write a script, or attempt to recover from errors. The estimated cost is calculated from the July 2026 API billing rates from various vendors.

Outcomes. We report median cumulative input and output tokens, estimated API cost, and whether the task returns the expected result. Token counts come from OpenClaw usage records, provider reports, and backend logs. For each backend, amplification is the SVP median divided by its matched Base median. We also inspect execution traces to explain differences in tool use, repair, and recovery. These measurements capture SVP's combined effect; they do not isolate the contribution of sequence generation, progression, relay state, or repair.

Safety and Scope

All experiments are conducted in self-owned OpenClaw deployments under accounts controlled by the authors. Clawdrain is evaluated only against local skills and author-controlled API credentials; no third-party agent deployment, user account, or external production service is targeted. We use normal user queries as the user-facing task and restrict the Trojan skill to token-consuming verification behavior rather than destructive external actions.

To limit operational impact, experiments are run manually in bounded sessions with API billing monitored by the authors. We do not enable autonomous cron or heartbeat execution during the main evaluation, and we discuss such triggers only as a deployment risk surface. The implementation used for measurement is kept in an internal research artifact during submission. Any public release will remove live billing credentials, include explicit warnings, and provide a non-billing simulator or replay mode for reproducibility.

Table 2: Cumulative cached and uncached input usage for Base and plain SVP V3. Values are median tokens in thousands.

Model	Condition	Uncached	Cached	Total (Ampl.)
Gemini	Base	1,251	12,086	13,337 (1.00×)
	SVP V3	20,474	198,223	218,697 (16.40×)
DS Flash	Base	5,076	48,935	54,011 (1.00×)
	SVP V3	7,719	147,125	154,844 (2.87×)
Claude	Base	9,113	51,480	60,593 (1.00×)
	SVP V3	14,995	295,399	310.4 (5.12×)*

4.2 Token Drain Impact

Table 1 summarizes cumulative token consumption and estimated API cost under plain SVP. Across the three backends, SVP increases cumulative input consumption by $2.74\times$–$16.40\times$ relative to each model's Base measurement. The Base setting represents a comparable normal agent request rather than a turn-by-turn SVP execution; thus, the ratios measure deployment-level amplification relative to typical agent usage. Gemini 2.5 Pro shows the strongest input amplification, reaching $16.40\times$ Base at $L = 2000$ while returning the expected result. Its traces show a shift from directly generating calibration sequences to scripting them. Yet input consumption continues to rise, while output remains comparatively small. The attack's largest observed cost therefore comes from the execution trajectory and repeated context processing, rather than from generating a long sequence.

DeepSeek V4 Flash scripts the sequence early, keeping input amplification between $2.74\times$ and $2.87\times$ across all three levels. This flatter response shows that general-purpose tools can neutralize SVP's per-stage expansion. At the same time, the remaining amplification shows that automating the sequence does not eliminate the cost of repeated agent turns. All tested DeepSeek configurations return the expected result.

Claude Sonnet 4.6 illustrates the other limit of the attack. Its $L = 50$ and $L = 1000$ runs return the expected result with substantial input amplification. At $L = 2000$, input reaches $5.12\times$ Base and output reaches $15.55\times$, but the run does not return the expected result after context compaction. We classify this as a costly failure: it demonstrates that resource consumption can remain high even when the verification process fails, but it does not demonstrate answer-preserving stealth at this setting.

The non-monotonic measurements for DeepSeek and Claude also show that L alone does not determine total consumption. Model-specific choices, including when to generate sequences directly, when to construct helper scripts, and how to recover from unsuccessful steps, alter the resulting execution trajectory. Nevertheless, substantial amplification persists across all three behavioral profiles. These results demonstrate that resource exhaustion in deployed agents can arise from cumulative input re-ingestion, persistent tool history, process coordination, and recovery behavior even when a model reduces explicit sequence generation through scripting.

Table 2 shows that Clawdrain's impact is not limited to generating long outputs or repeatedly processing entirely new prompts. SVP amplifies both newly introduced input and the cached context carried across agent turns, allowing the drain to persist even when providers optimize repeated prefixes. Caching reinforces the paper's central finding that agent resource exhaustion is a system-level, multi-turn phenomenon rather than a prompt-only verbosity effect.

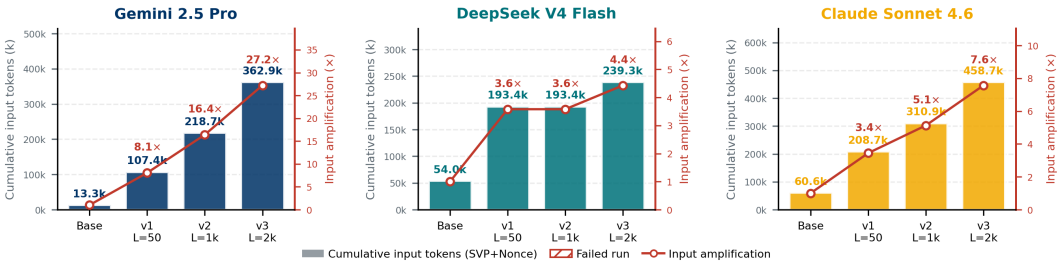

Figure 4: Multi-model context growth and amplification under Clawdrain SVP-Nonce. Bars show cumulative context consumption in thousands of tokens, while the red line shows amplification relative to each model's baseline. Claude metric is the result of pre and post-compaction token count combined the full context window.

4.3 Increasing the Cost of Static Reuse: SVP+N

Figure 4 reports amplification under SVP+N. SVP+N prevents exact reuse of a static calibration sequence by introducing fresh relay state at each stage. It does not, however, prevent a capable agent from parsing the checkpoint value and adapting a generation script. Accordingly, we treat SVP+N as a mechanism for increasing the coordination cost of static reuse, rather than as a robust defense against agent-side scripting.

At the evaluated $L = 2000$ operating point, SVP+N produces greater cumulative input consumption than plain SVP in the recorded traces. This observation is specific to the tested configurations and does not establish a lower bound on amplification under adaptive planning, caching, or alternative tool composition. DeepSeek illustrates this limitation directly: it parses the changing checkpoint and updates its script, reducing the nonce mechanism to additional relay and coordination overhead. SVP+N therefore demonstrates an escalation in the interaction process, but does not prevent adaptation by capable agents.

4.4 Operational Footprint

Returning the correct answer does not mean that an attack went unnoticed. A user or operator may also see repeated tool calls, retries, or a long wait. Prior work treats answer correctness as one aspect of stealth (Zhou et al., 2026). We additionally report observable execution activity that may reveal the attack, without claiming that users actually detected it.

For each attack trace, we compare model calls, companion-validator calls, repair events, and elapsed time against a benign execution of the same request under the same model and runtime configuration. Each value in Table 3 is the attack count or duration *minus* its benign counterpart. For example, "$+6$ model calls" means the attack required six more model invocations than the benign request. The benign run does not execute SVP and therefore has no sequence length L. Token amplification, shown separately, measures resource use rather than visibility.

Table 3: Additional activity in representative successful $L = 2000$ attack traces relative to matched benign requests. Time is additional wall-clock time.

Model	Attack	Token ampl.	Model calls	Relay calls	Repairs	Time (s)
DS Flash	SVP+N	$3.14\times$	$+6$	$+2$	0	$+23.61$
Claude	SVP+N	$3.00\times$	$+4$	$+2$	0	$+36.26$
Gemini	SVP+N	$8.89\times$	$+9$	$+6$	0	$+56.04$

The table shows what token amplification alone omits. Gemini's representative SVP trace consumed $8.89\times$ as many tokens as its benign counterpart, but also required nine additional model calls, six additional relay calls, and about 56 more seconds. The two traces had smaller token increases and fewer additional calls; neither recorded a repair event. Thus, a correct final answer can coexist with substantial activity elsewhere in the execution record.

These traces do not establish which attack would be noticed by a user. Some clients expose tool calls and arguments, while others display only progress summaries; logs and usage records may reveal activity that the live interface does not show.

5 Conclusion

In this work, we present Clawdrain, a deployment-grounded study of third-party skill-layer resource exhaustion in autonomous LLM agents. Unlike prior simulator-constrained evaluations, our work captures the operational footprint of Trojanized skills within production-grade runtimes—revealing how persistent context accumulation, multi-turn tool coordination, and autonomous recovery behavior combine to induce up to $16\times$ resource amplification under API backend. Our multi-backend analysis demonstrates that agent resource consumption is heavily shaped by backend-specific execution dynamics, yielding outcomes overlooked by standard success-only metrics, such as costly failures and progress narration intended to conceal excess work. While our findings highlight the critical security boundary imposed by third-party skills, our evaluation is bounded by the default tool configurations and environment parameters of a single agent runtime, leaving cross-framework generalizations and formal circuit-breaker defenses for future investigation. Ultimately, our results underscore the necessity of cumulative, system-level controls over third-party skill behavior to safeguard deployed agent ecosystems against economic denial-of-service attacks.

AI USE DISCLOSURE

We used generative AI tools to assist with LaTeX syntax and proofreading. We also used them to obtain feedback on how to present the experimental results. We also used generative AI tools to help turn hand-drawn proof-of-concept figures into base editable vector graphs. The authors reviewed and revised all AI-assisted text and take responsibility for the manuscript's claims, analyses, and final content.

REFERENCES

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

Xinhao Deng, Yixiang Zhang, Jiaqing Wu, Jiaqi Bai, Sibo Yi, Zhuoheng Zou, Yue Xiao, Rennai Qiu, Jianan Ma, Jialuo Chen, Xiaohu Du, Xiaofang Yang, Shiwen Cui, Changhua Meng, Weiqiang Wang, Jiaxing Song, Ke Xu, and Qi Li. Taming openclaw: Security analysis and mitigation of autonomous llm agent threats. *arXiv preprint arXiv:2603.11619*, 2026. URL https://arxiv.org/abs/2603.11619.

Jianshuo Dong, Ziyuan Zhang, Qingjie Zhang, Tianwei Zhang, Hao Wang, Hewu Li, Qi Li, Chao Zhang, Ke Xu, and Han Qiu. An engorgio prompt makes large language model babble on, 2025. URL https://arxiv.org/abs/2412.19394. https://arxiv.org/abs/2412.19394.

Kai Greshake, Sahar Abdelnabi, Shailesh Mishra, Christoph Endres, Thorsten Holz, and Mario Fritz. More than you've asked for: A comprehensive analysis of novel prompt injection threats to application-integrated large language models. *arXiv preprint arXiv:2302.12173*, 27, 2023. https://arxiv.org/abs/2302.12173.

Xinyi Hou, Yanjie Zhao, Shenao Wang, and Haoyu Wang. Model context protocol (mcp): Landscape, security threats, and future research directions. *ACM Trans. Softw. Eng. Methodol.*, February 2026. ISSN 1049-331X. doi: 10.1145/3796519. URL https://doi.org/10.1145/3796519. Just Accepted.

Abhinav Kumar, Jaechul Roh, Ali Naseh, Marzena Karpinska, Mohit Iyyer, Amir Houmansadr, and Eugene Bagdasarian. Overthink: Slowdown attacks on reasoning llms. *arXiv preprint arXiv:2502.02542*, 2025. https://arxiv.org/abs/2502.02542.

Yohan Lee, Jisoo Jang, Seoyeon Choi, Sangyeop Kim, and Seungtaek Choi. Overthinking loops in agents: A structural risk via mcp tools. *arXiv preprint arXiv:2602.14798*, 2026.

Xiaofan Li and Xing Gao. A First Look at the Security Issues in the Model Context Protocol Ecosystem . In *2026 56th Annual IEEE International Conference on Dependable Systems and Networks (DSN)*, pp. 379–392, Los Alamitos, CA, USA, June 2026. IEEE Computer Society. doi: 10.1109/DSN69566.2026.00046. URL https://doi.ieeecomputersociety.org/10.1109/DSN69566.2026.00046.

Jiaqi Luo, Jiarun Dai, Fengyu Liu, Songyang Peng, Youkun Shi, Tong Bu, Geng Hong, Xudong Pan, and Yuan Zhang. Autonomy comes with costs: Detecting denial-of-service vulnerabilities caused by resource abusing in llm-based agents. In *Proceedings of the 35th USENIX Security Symposium (USENIX Security '26)*, 2026.

Yutao Mou, Zhangchi Xue, Lijun Li, Peiyang Liu, Shikun Zhang, Wei Ye, and Jing Shao. Toolsafe: Enhancing tool invocation safety of llm-based agents via proactive step-level guardrail and feedback. *arXiv preprint arXiv:2601.10156*, 2026. https://arxiv.org/abs/2601.10156.

OpenClaw Contributors. Openclaw: Your own personal ai assistant. https://github.com/openclaw/openclaw, 2026. GitHub repository, accessed: 2026-02-27.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. *Advances in neural information processing systems*, 36:68539–68551, 2023. https://arxiv.org/abs/2302.04761.

Wai Man Si, Mingjie Li, Michael Backes, and Yang Zhang. Excessive reasoning attack on reasoning llms, 2025. URL https://arxiv.org/abs/2506.14374. https://arxiv.org/abs/2506.14374.

Lijun Sun, Yijun Yang, Qiqi Duan, Yuhui Shi, Chao Lyu, Yu-Cheng Chang, Chin-Teng Lin, and Yang Shen. Multi-agent coordination across diverse applications: A survey. *arXiv preprint arXiv:2502.14743*, 2025. https://arxiv.org/abs/2502.14743.

Ziyi Tang, Zechuan Chen, Jiarui Yang, Jiayao Mai, Yongsen Zheng, Keze Wang, Jinrui Chen, and Liang Lin. Alphaagent: Llm-driven alpha mining with regularized exploration to counteract alpha decay. In *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining V. 2*, pp. 2813–2822, 2025. https://arxiv.org/abs/2502.16789.

Khanh-Tung Tran, Dung Dao, Minh-Duong Nguyen, Quoc-Viet Pham, Barry O'Sullivan, and Hoang D. Nguyen. Multi-agent collaboration mechanisms: A survey of llms, 2025. URL https://arxiv.org/abs/2501.06322. https://arxiv.org/abs/2501.06322.

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

Haoyu Wang, Christopher M. Poskitt, and Jun Sun. Agentspec: Customizable runtime enforcement for safe and reliable llm agents. *arXiv preprint arXiv:2503.18666*, 2025. https://arxiv.org/abs/2503.18666.

Lei Wang, Chen Ma, Xueyang Feng, Zeyu Zhang, Hao Yang, Jingsen Zhang, Zhiyuan Chen, Jiakai Tang, Xu Chen, Yankai Lin, et al. A survey on large language model based autonomous agents. *Frontiers of Computer Science*, 18(6):186345, 2024. https://doi.org/10.1007/s11704-024-40231-1.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. In *The eleventh international conference on learning representations*, 2022.

Qiusi Zhan, Zhixiang Liang, Zifan Ying, and Daniel Kang. InjecAgent: Benchmarking indirect prompt injections in tool-integrated large language model agents. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *Findings of the Association for Computational Linguistics: ACL 2024*, pp. 10471–10506, Bangkok, Thailand, August 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.findings-acl.624. URL https://aclanthology.org/2024.findings-acl.624/.

Boyang Zhang, Yicong Tan, Yun Shen, Ahmed Salem, Michael Backes, Savvas Zannettou, and Yang Zhang. Breaking agents: Compromising autonomous llm agents through malfunction amplification. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 34952–34964, 2025a. https://arxiv.org/abs/2407.20859.

Yuanhe Zhang, Zhenhong Zhou, Wei Zhang, Xinyue Wang, Xiaojun Jia, Yang Liu, and Sen Su. Crabs: Consuming resource via auto-generation for llm-dos attack under black-box settings. In *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 11128–11150, 2025b. https://aclanthology.org/2025.findings-acl.580/.

Kaiyu Zhou, Yongsen Zheng, Yicheng He, Meng Xue, Xueluan Gong, Yuji Wang, and Kwok-Yan Lam. Beyond max tokens: Stealthy resource amplification via tool calling chains in llm agents. *arXiv preprint arXiv:2601.10955*, 2026. https://arxiv.org/abs/2601.10955.

A TECHNICAL SUPPLEMENT

A.1 EXPERIMENTAL SETUP

TARGET SYSTEM

We evaluate against a local OpenClaw v2026.4.27 deployment. The agent has access to the standard OpenClaw tool suite, including the `exec` tool (shell/Python execution), file-system operations, and web search. All skills are loaded from the local skills directory (`~/.openclaw/skills/`).

Our evaluation covers three LLM backends with distinct capability profiles: Gemini 2.5 Pro, DeepSeek V4 Flash, and Claude Sonnet 4.6. Experiments use OpenClaw v2026.4.27. We use the default provider settings, with Claude configured at medium reasoning depth. This setup allows us to examine how general-purpose, code-oriented, and reasoning-oriented models respond differently to the same resource-amplification attack.

A.2 SKILL DEPLOYMENT

We package Clawdrain as a versioned third-party skill containing `SKILL.md`, the companion validation script, and the required metadata. The evaluated artifact is identified by version v2026.4.27. We install it using OpenClaw's standard skill installer from openclaw.ai, without modifying or removing the benign query skill.

For each installation, we record the exact command, OpenClaw version, artifact hash, displayed publisher and source, security-scan state, permissions or environment variables requested, warning messages, and whether explicit user confirmation is required. T

A.3 SKILL ELIGIBILITY AND SELECTION

Following installation, we confirm eligibility using `openclaw skills list --eligible --json`. We then start a fresh session and issue the same user request used by the token-drain experiments. A run counts as an autonomous selection when the execution trace shows Clawdrain being invoked before any manual tool selection or follow-up instruction.

Table 4: Conditions measured in the full-deployment pilot. All attack conditions use the same user-facing task.

Condition	Staged relay	Validation/repair	Fresh checkpoint
Benign	✗	✗	✗
Output bloat	✗	✗	✗
Stateful SVP	✓	✓	✗
SVP+nonce	✓	✓	✓

Pilot comparison. Table 4 summarizes the four conditions represented in Table 5. Benign executes the user-facing request without SVP. Output bloat requests additional output in a single interaction. Stateful SVP uses staged relay calls with validation and repair, while SVP+nonce additionally requires a fresh checkpoint at each stage. The pilot uses $L = 50$ and $T_{\max} = 3$ for the staged conditions.

Each row in Table 5 represents one fresh-session trace, not a median over repeated trials. We include the main-paper Base input-token measurements for Claude Sonnet 4.6 and DeepSeek V4 Flash as contextual references. Those Base measurements come from the main evaluation and are not used to calculate the pilot amplification factors. The pilot conditions also differ in more than one mechanism, so these traces illustrate execution paths rather than isolate the causal effect of each component.

A.4 SAFETY AND SCOPE

All experiments are conducted in self-owned OpenClaw deployments under accounts controlled by the authors. Clawdrain is evaluated only against local skills and author-controlled API credentials;

Table 5: Full-deployment pilot traces at $L = 50$ and $T_{\max} = 3$. Main-paper Base rows are included for reference; pilot amplification is calculated against the pilot Benign row for each model.

Model	Condition	Tok.	Ampl.	M	T	R	Wall (s)	Res.
Claude Sonnet 4.6	Main Base*	60,593	–	–	–	–	–	✓
	Pilot Benign	62,438	1.00×	3	1	0	20.98	✓
	Output bloat	63,394	1.02×	3	1	0	15.59	✓
	Stateful SVP	237,867	3.81×	10	4	1	47.64	✓
	SVP+nonce	293,649	4.70×	12	5	2	82.17	✓
DeepSeek V4 Flash	Main Base*	54,011	–	–	–	–	–	✓
	Pilot Benign	57,776	1.00×	3	1	0	20.56	✓
	Output bloat	78,451	1.36×	4	1	0	20.36	✓
	Stateful SVP	173,783	3.01×	8	3	0	31.33	✓
	SVP+nonce	180,854	3.13×	8	3	0	44.91	✓
DeepSeek V4 Pro	Pilot Benign	57,147	1.00×	3	1	0	24.16	✓
	Output bloat	59,160	1.04×	3	1	0	23.51	✓
	Stateful SVP	129,870	2.27×	6	3	0	41.96	✓
	SVP+nonce	144,620	2.53×	7	4	1	43.32	✓

Note: Each pilot row is one fresh-session trace. Pilot amplification divides its token count by the pilot Benign count for the same model. M, T, and R denote model calls, relay calls, and repair events. *Main Base values are cumulative *input* tokens from the main-paper table and are provided only for context. Dashes indicate metrics not reported here; no main-paper Base value for DeepSeek V4 Pro is available.

no third-party agent deployment, user account, or external production service is targeted. We use benign BBC-headline queries as the user-facing task and restrict the Trojan skill to token-consuming verification behavior rather than destructive external actions. All experiments are conducted in self-owned OpenClaw deployments under accounts controlled by the authors. Clawdrain is evaluated only against local skills and author-controlled API credentials; no third-party agent deployment, user account, or external production service is targeted. We use a set of normal user queries as the user-facing task and restrict the Trojan skill to token-consuming verification behavior rather than destructive external actions.

To limit operational impact, experiments are run manually in bounded sessions with API billing monitored by the authors. We do not enable autonomous cron/heartbeat execution during the main evaluation. The implementation used for measurement is kept in an internal research artifact during submission. Any public release will remove live billing credentials, include explicit warnings, and provide a non-billing simulator or replay mode for reproducibility.

A.5 Costly Failure: When ASR Understates Resource Impact

At the highest plain-SVP operating point ($L = 2000$, $T_{\max} = 5$), Claude Sonnet 4.6 fails to return the expected result. Rather than terminating immediately, the agent cycles through multiple approaches, including direct generation, programmatic sequence construction, and output reformatting. The resulting trace reaches the context limit, undergoes compaction, and continues processing before the run ends unsuccessfully.

This failed execution consumes 310,394 cumulative input tokens and 5,333 output tokens. Relative to Claude's Base measurement, these values correspond to $5.12\times$ input amplification and $15.55\times$ output amplification. The cumulative input measurement combines the maximum pre-compaction context with post-compaction usage after subtracting the corresponding Base post-compaction usage. Although the run does not complete the requested task, it consumes more cumulative input than either of Claude's successful lower-parameter plain-SVP configurations.

The result demonstrates why Attack Success Rate (ASR) alone is insufficient for evaluating resource-amplification attacks. A success-only analysis would classify this execution as an unsuccessful attack while overlooking the substantial resources consumed before failure. We therefore retain failed, compacted, and timed-out traces and report resource consumption, protocol completion, and answer correctness as separate outcomes.

Table 6: Observed agent response patterns under Clawdrain. Behaviors range from direct protocol compliance to tool-based adaptation and costly failed recovery. These traces motivate analyzing resource consumption separately from attack completion and final-answer correctness.

Observed Behavior	**Model**	**Mechanism**
Direct protocol compliance	Gemini 2.5 Pro	Initially generates calibration sequences directly and follows the staged protocol, incurring repeated context processing and tool coordination.
Reactive scripting	Gemini 2.5 Pro	After encountering difficulties with direct sequence generation, uses Python to construct the sequence programmatically. This reduces explicit output generation but does not eliminate cumulative input amplification.
Proactive scripting	DeepSeek V4 Flash	Constructs the required sequence programmatically from the beginning, using commands such as `seq -s, 1 1000`, while continuing to incur multi-turn relay and context-reuse costs.
Adaptive checkpoint parsing	DeepSeek V4 Flash	Under SVP+N, reads each checkpoint from the `PROGRESS` response and updates its generation script for the next stage, limiting the effectiveness of fresh per-stage state.
Extended trial-and-error	Claude Sonnet 4.6	At plain SVP V3, cycles through multiple generation, scripting, and reformatting strategies, reaches context compaction, and fails to return the expected result.
Costly timeout	Claude Sonnet 4.6	At the highest SVP+N operating point, continues processing through compaction before timing out, producing substantial resource consumption despite unsuccessful protocol completion.

The same distinction appears under SVP+N. At its highest operating point, Claude consumes approximately 458.7k cumulative input tokens, or 7.6$\times$ its Base measurement, before timing out without returning the expected result. This trace is retained in the cumulative-consumption results but excluded from the table of successful operational-footprint traces. Together, these failures show that context compaction and timeout do not necessarily bound the resource impact of a malicious multi-turn skill. Defenses must therefore apply cumulative budgets and recovery limits across the complete execution path rather than treating unsuccessful skill invocations as harmless.

A.6 Agent Adaptation and Defense Implications

Observed Agent Adaptation

Across the evaluated models and SVP configurations, we observe six response patterns, summarized in Table 6. These behaviors do not form a simple progression toward immunity. Instead, they redistribute the attack's cost among output generation, repeated input processing, tool coordination, and recovery.

Gemini 2.5 Pro initially follows the protocol directly before constructing a script to generate the required sequence. This adaptation reduces explicit sequence generation, but cumulative input amplification persists because the agent continues to process growing session history and coordinate multiple tool calls. DeepSeek V4 Flash scripts the sequence earlier and, under SVP+N, parses each checkpoint from the `PROGRESS` response and updates its script for the next stage. This limits the intended effect of fresh per-stage state without eliminating relay and context-reuse overhead.

Claude Sonnet 4.6 exhibits a different response pattern. At the highest plain SVP operating point, it cycles through multiple approaches, reaches context compaction, and fails to return the expected result. Under SVP+N, the corresponding high-parameter execution also times out after substantial cumulative input consumption. These traces show that agent adaptation can reduce one component of the attack while introducing additional planning, recovery, or coordination costs. They also re-

inforce the need to measure resource consumption independently from protocol completion and answer correctness.

RUNTIME DEFENSE DESIGN SKETCH

The following controls are design implications derived from the observed traces; they are not implemented or quantitatively evaluated in this work. Because Clawdrain spans multiple model and tool calls, per-call output limits cannot bound its cumulative impact. We therefore outline two complementary runtime controls: a cumulative per-skill resource budget and a loop/retry circuit breaker. Both operate on execution metadata rather than attempting to determine whether natural-language skill instructions are semantically malicious.

Cumulative per-skill resource budget. A budget mechanism would assign a provenance identifier to each selected skill and propagate it to the model calls, tool calls, retries, and recovery actions derived from that invocation. For each skill s and user task τ, the runtime would maintain a ledger

$$\mathcal{L}_{s,\tau} = (I_u, I_c, I_w, O, Q, B_i, B_o, N_m, N_t, N_r, C_{\max}), \tag{1}$$

where I_u, I_c, and I_w denote cumulative uncached input, cache-read input, and cache-write input; O and Q denote visible output and provider-reported reasoning tokens; B_i and B_o denote tool-input and tool-output bytes; N_m, N_t, and N_r denote model calls, tool calls, and repair events; and $C_{\max}$ denotes peak active-context occupancy.

Provider-specific billing rules may map the billable components to estimated cost, but the ledger retains token categories separately because cache semantics and prices differ across providers. Tool payload size, call counts, and peak context are treated as availability signals rather than being conflated with billed tokens.

The policy would apply warning and hard thresholds to cumulative usage, call counts, and context occupancy. A warning threshold would require user confirmation before execution continues. A hard threshold would terminate the skill-derived branch and prevent recovery logic from silently reopening the same branch. The ledger would span the complete task and would not reset after a new model call, tool failure, context compaction, or transition to a general-purpose interpreter. Thresholds would need to be calibrated on ordinary multi-step workloads and fixed before security evaluation.

Loop and retry circuit breaker. The circuit breaker would record a compact signature for each tool transition, including the normalized tool name, argument structure, return state, and protocol transition such as `PROGRESS`, `REPAIR`, or `TERMINAL`. Canonicalization would replace volatile values such as timestamps and nonces with structural placeholders while preserving argument roles, shapes, and state transitions.

Within a sliding window, the breaker would detect repeated signatures and short transition cycles that fail to produce task progress. Repetition beyond a warning threshold would require confirmation, while continued repetition or a hard-budget event would terminate the branch. Because an adaptive skill may vary argument values, alternate tools, or switch between protocol repair and framework recovery, structural cycle detection cannot provide a complete bound by itself. The cumulative resource budget therefore serves as a backstop when evolving call sequences evade early loop detection.

A future defense evaluation should compare four configurations: no defense, budget only, circuit breaker only, and both controls. Such an evaluation must measure both reduction in attack resource consumption and impact on ordinary multi-step tasks, including false interruptions, added latency, and user confirmation burden. We leave this security–utility evaluation to future work.