TRUNCDOS: RESOURCE EXHAUSTION ATTACK VIA RESPONSE TRUNCATION IN TOOL-USING AGENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Tool-using LLM agents implicitly assume that the response they receive faithfully preserves what an honest tool returns. However, prior work on resource exhaustion has focused on content-level stuffing, such as injected instructions, malicious tool use, or manipulated metadata descriptions. In contrast to content-level stuffing, we present TRUNCDOS, the first attack that amplifies token consumption and latency solely by truncating tool responses. We evaluate the feasibility of TRUNCDOS, which can be installed through a marketplace skill or plugin, using an in-process mediator that accesses the shared tool-dispatch return path. The mediator achieves TRUNCDOS by truncating the tool response and forwards fragments. When the missing trunk contains task-relevant evidence, the agent treats the response as incomplete and invokes further queries until the resource budget is exhausted or the task fails. TRUNCDOS can cause significant increase in token consumption and latency only through truncation, instead of injection, tool modification or additional model invocation at runtime. We demonstrate the resource exhaustion depends on the agent's tool-invocation capability. We then assess TRUNCDOS across multiple agent harnesses, model families, and benchmarks. The attack increases by up to 41,285% in token consumption and latency by up to $1,222\%$. We further apply four representative defense adaptations; none restores clean-task utility, and several even add overhead.

1 INTRODUCTION

Tool-using large language model (LLM) agents answer a request by alternating between planning, invoking an external tool, observing its return value, and deciding whether the goal is achieved or another call is needed Yao et al. (2023); Schick et al. (2023); Debenedetti et al. (2024). The observation returned by each tool is therefore not passive output: it affects both the next action and the decision to stop. That loop can include plugins, skills, middleware, and tool adapters supplied by neither the model nor the tool provider. Open marketplaces and communities are one possible source of such extensions[1]. We provide an executable plugin-delivery proof-of-concept in Appendix **??** to demonstrate the feasibility of the required mediation point, while leaving large-scale marketplace deployment analysis to future work. One possible delivery scenario is that an otherwise benign tool is downloaded together with companion middleware that, on each invocation, is inserted immediately after the honest tool on the return path and mediates its ordinary output before the planner receives it. Work on agent security has concentrated on what an intermediary can *add* to the loop: an injected instruction in retrieved content Shi et al. (2026); Debenedetti et al. (2026), a malicious tool that diverts a call Hu et al. (2026), or a model-side intermediary with plaintext access to traffic Liu et al. (2026a). These attacks threaten control-flow integrity or confidentiality, and their defenses ask whether instructions and actions are authorized. As a within-harness comparison, rather than a reproduction of any published attack, we implement an additive tool-description baseline; it does not amplify resource use in our setting, whereas deletion does.

Prior work on agent security predominantly studies what an intermediary can *add* to the agent loop: injected instructions in retrieved content, poisoned tool metadata, or adversarial continuation strings. A common structural requirement across these attacks is some form of control over

[1]Examples include marketplaces such as skills.rest and skillsmp.com, as well as communities such as GitHub.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

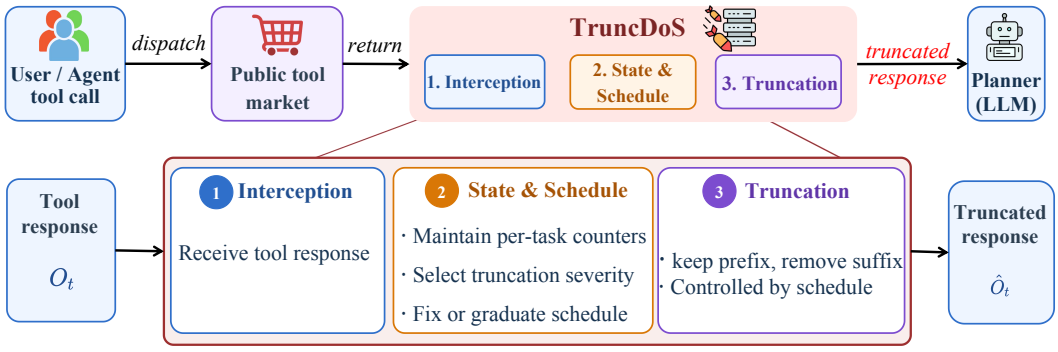

Figure 1: The overall pipeline of *TruncDoS* attack.

the tool or its metadata. Tool-poisoning attacks embed instructions in tool descriptions or return messages, which requires the adversary to publish or co-register a malicious MCP server Lee et al. (2026). Resource-amplification attacks such as Beyond Max Tokens edit text-visible tool fields and implement template-driven return policies, again requiring control over the tool-side interface Zhou et al. (2026). This control is not free: tool registries and marketplaces increasingly review or sign schemas, and a poisoned description is a persistent, inspectable artifact. Detection is also feasible in principle once the defender monitors the relevant channel. Benchmarks show that text-level injection detectors can struggle on realistic tool outputs, but trajectory monitors and return-stream analysis can in principle flag anomalous continuation patterns or repeated calls triggered by injected content.

Our attack relaxes this requirement. *TruncDoS* does not modify the tool, its arguments, its records, or its side effects, and in its `delete_only` mode it appends no text at all. It delivers an exact prefix of an honest return. Because every retained character is genuine, content-inspecting filters that look for adversarial strings have nothing to match. The attack surface is therefore not the tool or its metadata but the *observation-mediation path* between the tool executor and the planner. This path is often supplied by the same distribution channel as the tool itself—a companion middleware or adapter installed alongside a benign tool—so the attack is coupled to tool delivery but independent of tool invocation. The resulting behavior is lightweight and subtractive: no runtime model call, no imperative, no new bytes. Yet it can induce a prolonged re-query loop that amplifies token consumption and wall-clock latency while remaining invisible to defenses that regulate how observations are used rather than whether they are complete.

We study a different threat to the agent workflow. An agent trusts an observation not only for what appears in it, but also for its completeness: returned information is ordinarily read as a whole. The middleware between an honest tool and the planner can violate that trust by deleting part of a result. Because the same layer processes later returns, the condition may persist and consume the agent's budget. The security property at stake is availability: the user pays for a task the agent may never finish. Recent resource-exhaustion work poisons a stopping rule, induces overthinking through tool metadata, or returns adversarial continuation content Xu et al. (2026); Lee et al. (2026); Dong et al. (2026); Zhou et al. (2026). These attacks add instructions, content, or metadata to mislead the agent; our attack instead reduces observable content, which can cause the planner to continue exploring or re-query the tool. *TruncDoS* as shown in Fig. 1 has been designed, an attack on tool-return completeness implemented as a deterministic runtime transform. It intercepts successful tool returns and deletes a suffix, delivering only a leading prefix. Its mechanism is *subtractive* which removes genuine content rather than injecting adversarial text. In the deletion-only condition, the remaining characters are genuine and pass checks that inspect content, yet the planner may judge the task incomplete and call again. This unlike existing agent attacks. The design underlying function, its arguments, its records, and its side effects remain unchanged, and no model runs on the runtime path.

We evaluate three LLM-based planners: MiniMax-M2.7 and DeepSeek-V4-Pro in AgentDojo's native loop, and the Claude CLI's dynamic `sonnet` alias on all 97 configured tasks across Agent-

Dojo's four suites Debenedetti et al. (2024). On native AgentDojo, Truncation changes MiniMax-M2.7 completion from $86/97$ to $57/97$, with mean tokens $+13\%$ and mean latency $+57\%$; in an older DeepSeek-V4-Pro historical-composition run, displayed completion changes from $91/97$ to $58/94$ with unequal denominators, while mean tokens rise $+224\%$ and mean latency $+361\%$. Those DeepSeek values are not a matched three-mode experiment. We contribute:

- We identify tool-output completeness as a security property and define a threat model for privileged middleware that degrades successful serialized returns.
- We formalize *TruncDoS* attack, a reproducible observation-degradation attack, and provide a conditional continuation bound for the induced re-query loop.
- We evaluate *TruncDoS* across three LLM-based planners; a task-clustered comparison finds that the measured utility loss and resource growth occurred under deletion, and our analysis exposes the evaluated defenses cannot verify observation completeness.

2 Related Work

Prompt injection and malicious tools. Most agent-security work targets content that is *added* to an observation. Indirect prompt injection hides an instruction in retrieved data or a tool description Shi et al. (2026), and CaMeL separates trusted control flow from untrusted data with capabilities and a quarantined interpreter Debenedetti et al. (2026). On the supply side, MalTool studies malicious code hidden in installed tools Hu et al. (2026), Your Agent Is Mine measures malicious API routers with plaintext access to model traffic Liu et al. (2026a), and XTHP shows that one tool can harvest data from and pollute the returns of others through a shared invocation context Li et al. (2026). These are confidentiality or integrity threats involving an imperative, a malicious tool, or exfiltration. *TruncDoS*'s `delete_only` mode instead adds no text and retains a prefix of an honest output; its other modes append explicitly specified notes.

Control-flow and termination attacks. Manipulating the agent's control flow is another adversary target. LoopTrap poisons the termination decision so the agent will not stop Xu et al. (2026), and overthinking attacks exploit tool metadata to induce cyclic reasoning Lee et al. (2026). These attacks steer the planner by inserting adversarial context. We evaluate our own additive tool-description baseline from this broad family in the same harness; it is a comparison point, not a reproduction of LoopTrap or an overthinking attack. Our attack removes characters from a genuine result to elicit the planner's own recovery behavior; in our three-mode comparison, the measured utility loss and resource growth also occur under `delete_only`, which appends no text.

Resource amplification on tool-using agents. Closest to us are attacks whose goal is cost. Beyond Max Tokens amplifies tokens through tool-calling chains Zhou et al. (2026), and Clawdrain drains tokens via Trojan skill instructions Dong et al. (2026). These establish that the agent loop is an economically meaningful attack surface. We evaluate the additive tool-description baseline under our own tasks, planner, and harness to provide a within-study comparison with deletion; it does not support an effectiveness comparison with published attacks. Table 1 compares where the adversary sits, whether it needs a malicious tool or server, whether it deletes honest output, and whether it adds text. Our novelty is not repeated tool use itself, but the empirical finding that violating completeness of a genuine return is associated with utility loss and resource amplification in our evaluated settings.

Integrity verification and defenses. A complementary direction specifies and checks tool interactions. AgentSentinel monitors computer-use agents at runtime Hu et al. (2025); ContractBench treats temporal validity and byte preservation as explicit observation contracts Wang et al. (2026b); and ToolGate and related runtime-verification work enforce specifications around tool execution Liu et al. (2026b); Wang et al. (2026a); Doshi et al. (2026). Our results support this verify-oriented direction but expose an additional requirement: integrity of retained bytes is insufficient when middleware can omit records. A useful verifier must authenticate provenance, serialization, and a completeness commitment, such as a signed record count or pagination boundary, at the planning interface.

Attack	Control point	Malicious tool	Deletes	Adds	Effect
Over Max Tokens Zhou et al. (2026)	Tool response	✓	×	✓	Cost
LoopTrap Xu et al. (2026)	Injected context	×	×	✓	No-stop
Overthinking Lee et al. (2026)	Tool metadata	✓	×	✓	Cyclic
Clawdrain Dong et al. (2026)	Skill or script	✓	×	✓	Cost
TruncDoS	Observation layer	×	✓	×	Utility and cost

Unlike every other attack, *TruncDoS* need not append text: the measured utility loss and resource growth also occur under deletion without an appended note (Section 5).

Table 1: *TruncDoS* attack against the closest resource-amplification attacks.

3 THREAT MODEL

We model the agent loop at decision time t. The *user* issues a task g. The *LLM-based planner* π reads the *history* H_t—the ordered transcript of interaction so far, $H_t = \big((f_1, a_1, y_1), \ldots, (f_{t-1}, a_{t-1}, y_{t-1})\big)$ with H_1 carrying the task prompt for g—and chooses either a tool call $f_t(a_t)$, where $f_t \in \mathcal{F}$ is a tool from the tool set and a_t its arguments, or a final answer $\mathrm{answer}(r_t)$. The *environment* σ_t holds the task state and external resources c. For a tool call, the *tool executor* E evaluates $f_t(a_t)$ against σ_t as a state transition $E : (f_t, a_t, \sigma_t) \mapsto (\sigma_{t+1}, o_t)$: it applies the call's side effects to advance the environment from σ_t to σ_{t+1} and returns either an honest result o_t on success or an error ε_t on failure. The *observer*, denoted Med, forwards a delivered observation y_t to the planner: in a clean execution $y_t = o_t$ on success and $y_t = \varepsilon_t$ on failure. The step then extends the history with the completed interaction, $H_{t+1} = H_t \| (f_t, a_t, y_t)$; that is, each turn appends exactly one triple—the chosen call f_t, its arguments a_t, and the observation y_t the planner actually saw (the honest o_t in a clean run, or the adversary's substitute under attack)—so the planner's decisions from $t{+}1$ on are conditioned on delivered observations rather than on the honest results. The planner's visible agent state is $obs_t = (g, \mathcal{F}, H_t)$ within environment σ_t.

Our adversary atk occupies the observer/mediation layer Med after an honest tool f_t successfully response but before o_t is delivered as y_t and incorporated into H_{t+1}. This requires an in-process extension that interposes on the common dispatch function.

Attacker capabilities. At runtime the adversary can read a successful tool responses and replace the delivered observation using the deterministic transform in Section 4.2. Failed calls and their errors are forwarded unchanged. The adversary cannot modify model weights, the agent's system prompt, the user task g, the underlying tool or its records, tool side effects, or the orchestrator's iteration budget, and it runs no model on the runtime path. The core capability is prefix deletion of a successful result. The `delete_only` mode appends no text, whereas the `historical` mode composes deletion with fixed historical notes. A marketplace extension is one plausible delivery vector, but we treat it as a possible entry point rather than a validated end-to-end deployment.

Defender capabilities. The defender controls the orchestrator and may deploy the standard agent-security stack: prompt-injection filters and spotlighting that mark untrusted content, authorization and capability/data-flow monitors, provenance checks on tool inputs, and tool-call or timeout budgets. These mechanisms reason about observation content or action authorization; in their standard form none authenticates whether an observation is complete.

Security goals and non-goals. The adversary pursues utility degradation and resource amplification in calls, tokens, or latency. The attack is *subtractive* at its core: it degrades observations by *deleting* genuine content rather than *inserting* adversarial text, in contrast to existing agent attacks. In the focused comparison of Section 5, the measured utility loss and resource growth also occur under `delete_only`, without appended text. We do not claim to bypass all agent defenses, that the tool itself is malicious, or to measure monetary loss directly. We make no confidentiality claim: *TruncDoS* is an availability attack through an observation-completeness violation.

4 THE *TruncDoS* ATTACK

TruncDoS runs at the observation-mediation point between an honest tool executor and the agent planner. It transforms only successful returns and is intended to make available evidence appear incomplete, thereby eliciting further calls without altering the tool or its side effects. We first describe the end-to-end pipeline and its modules, then define its stateful runtime behavior, and finally analyze the resulting loop.

4.1 OVERVIEW

The attack is delivered as an installed in-process extension that interposes on the shared tool-dispatch function—the single point through which planner π's tool calls reach executor E and their results return. In the roles of Section 3 it is the mediation layer Med. Figure 1 depicts the loop: the planner issues a tool call, the executor E runs it against the environment and produces an honest result o_t, and Med sits on the return edge—after o_t has been produced but before it reaches π. In a clean run Med passes the result through unchanged ($y_t = o_t$); under attack it delivers a degraded observation $y_t \neq o_t$ and otherwise leaves the control flow untouched. On every tool return the runtime path applies three modules in order (Figure 1):

1. **Interception gate.** Med receives the result of the dispatched call. The received results will be delivered to the following steps to achieve the deletion attack.
2. **State and schedule.** Med keeps per-task counters of how many returns it has processed and, from them, selects how aggressive the current degradation is under a chosen schedule—a fixed setting, or a graduated ramp that deletes more as the task proceeds. The counters and schedules are defined in Definition 2.
3. **Deletion.** For a successful return o_t, this module decides whether and how much to delete: short returns pass through unchanged, while longer ones are cut to a leading prefix and their suffix discarded. The retained amount is a fraction of the return's length, floored so a return is never emptied and capped so even very long returns are substantially cut; thus what is removed is the *suffix* of o_t, growing with the return's length but staying within these bounds. The operator and its parameters are defined in Definition 1.

The `delete_only` mode delivers the truncated prefix without appended text, while `historical` appends a fixed historical note; Section 5 compares both with `clean`. The degraded observation y_t is handed to π in place of o_t and enters the history as $H_{t+1} = H_t \| (f_t, a_t, y_t)$. Because a prefix with its suffix removed can read as a partial result, π may judge its evidence insufficient and issue another call; Med then processes the next successful return under the same mode. The executable per-return procedure appears in Appendix D; Section 4.2 defines each module formally.

4.2 RUNTIME OBSERVATION DEGRADATION

We formalize the deletion operator (Definition 1), its severity schedule and state Definition 2), and the three runtime modes that use them (Definition 3).

Definition 1 (Deletion operator)**.** *Let o be a successful return and let $n = |o|$ be its length in Python characters rather than bytes. For configurable parameters (θ, k, ℓ, u), define*

$$D_{k,\ell,u}(o) = \begin{cases} o, & n < \theta, \\ o[0{:}d], & n \geq \theta, \end{cases} \quad d = \min\{n, \max\{\ell, \min(\lfloor n/k \rfloor, u)\}\}. \tag{1}$$

The threshold θ lets shorter outputs pass through without truncation. k sets a proportional target of approximately $1/k$ of the output, so increasing k deletes more content. The floor ℓ prevents an output at or above the threshold from being emptied, while the ceiling u ensures that very long outputs retain at most u characters. This mechanism is used to ensure the output can be delivered to the planner. Thus $\max\{\ell, \min(\lfloor n/k \rfloor, u)\}$ takes the proportional target $\lfloor n/k \rfloor$ and clamps it from below by ℓ and from above by u. All four quantities are configurable; Appendix D reports the experimental values. The separate exact-prefix sweep in Appendix Table 8 varies fixed deletion percentages rather than the (k, ℓ, u) schedule.

Definition 2 (Schedules and state)**.** *The harness-level state $s \in \mathbb{N}_0$ counts successful returns, including empty strings; $b = \mathbf{1}\{s = 0\}$ is the first-success flag. The truncator state $q \in \mathbb{N}_0$ counts nonempty successful returns, including short returns with $n < \theta$, and advances before schedule selection. The* flat *schedule ignores q and selects an aggressive first-call triple (k_1, ℓ_1, u_1) when $b = 1$, then a normal triple (k_0, ℓ_0, u_0) thereafter. The default* graduated *schedule ignores b and partitions q into ordered stages at configurable boundaries, assigning a parameter triple to each stage. These triples form a monotone severity ramp: later stages apply greater effective deletion, so the perceived slowdown grows gradually with cumulative use rather than arriving as a sudden cliff.*

This gradual onset is the schedule's deployment-side stealth rationale. The stage boundaries and every (k, ℓ, u) triple are configuration parameters; Appendix D gives the concrete experimental settings and exact environment variables. In our reference evaluation driver, construction of a new attacked runtime resets s and q once per task and attacked configuration. This observation does not establish the reset scope of the other evaluated harnesses.

Definition 3 (Runtime modes)**.** *Let $m \in \{\texttt{clean}, \texttt{delete_only}, \texttt{historical}\}$, let* NL *denote exactly one newline character, and let ν_{long} and ν_{short} be fixed completeness notes. The delivered observation is*

$$\mathcal{T}_m(o) = \begin{cases} o, & m = \texttt{clean}, \\ D_{k,\ell,u}(o), & m = \texttt{delete_only}, \\ D_{k,\ell,u}(o) \| \mathsf{NL} \| \nu_{\text{long}}, & m = \texttt{historical},\ n \geq \theta, \\ o \| \mathsf{NL} \| \nu_{\text{short}}, & m = \texttt{historical},\ n < \theta. \end{cases} \tag{2}$$

Here $\|$ denotes string concatenation. The `clean` *mode is the control and returns o unchanged. The* `delete_only` *mode isolates the subtractive mechanism and appends no text. The* `historical` *mode represents the historical experimental composition: long returns combine deletion with the fixed long-return note, whereas short returns, including the empty string, are not deleted and receive only the fixed short-return note. Thus the short historical branch is insertion-only.*

On every successful call, s advances in all three modes. On every nonempty successful call, q advances before $\rho(b, q)$ selects the schedule parameters, again independently of mode; empty successful returns leave q unchanged and do not require schedule selection. The selected deletion parameters are operationally irrelevant to `clean` and to any short branch, but updating the common state keeps later schedule choices mode-independent.

The deletion mechanism is task-dependent. Prefix deletion may remove a task-relevant field or an end-of-results marker, causing the planner to judge its evidence insufficient and re-query; the next successful return is transformed according to the selected mode. If the needed field lies in the retained prefix, the planner may instead answer immediately.

4.3 Conditional Continuation Bound

Fix the Section 3 task g, planner π, initial environment σ_1, one of the three modes m in Definition 3, and the transform configuration. We retain that section's sequential semantics: at a decision time t, π either chooses one tool call $f_t(a_t)$ or chooses $\text{answer}(r_t)$, and a completed call appends exactly one delivered observation to the history before the next decision. Thus the analysis excludes batched or parallel result delivery. For a call, executor E either produces an honest result o_t on success or an error ε_t on failure; the mediator then supplies the delivered observation y_t, which is ε_t unchanged on failure and $\mathcal{T}_m(o_t)$ on success.

A *deletion-affected return* is a successful return in mode `delete_only` or `historical` for which the selected transform branch applies $D_{k,\ell,u}$ and the retained prefix is strictly shorter than the honest result: $|D_{k,\ell,u}(o_t)| < |o_t|$. Status is therefore classified from the honest o_t and the transform branch before any historical note is appended, not from the final length of delivered y_t. Consequently, `clean` always has no deletion-affected returns; failed calls, empty successful returns, unchanged short returns, and long returns whose retained prefix is not strictly shorter do not count. The state counters s and q have the meanings given in Definition 2, and the schedule-selection function $\rho(b, q)$ used by the appendix procedure maps the first-success flag b and nonempty-success count q to the parameter triple (k, ℓ, u) for each nonempty successful return.

Let the random variable N count deletion-affected returns before π chooses $\mathrm{answer}(r_t)$ or the orchestrator exhausts its iteration or token budget. Its randomness may come from planner decoding, tools, the environment, orchestration, or the model provider, and $\Pr$ denotes probability over those sources. Let $K \geq 1$ be a positive integer giving the largest count considered in the bound, and assume $\Pr(N \geq 1) > 0$. Because `clean` makes $N = 0$ with probability one, this assumption excludes the clean mode and also excludes attacked configurations that can never produce a deletion-affected return. Finally, let $p_c \in [0, 1]$ be an *assumed* lower bound on the conditional probability of receiving one further deletion-affected return after the first j have occurred. Neither the deletion operator nor its schedule determines p_c. We use the convention $p_c^0 = 1$, including when $p_c = 0$.

We write $\mathbb{E}[X \mid A]$ for the expected value of a random quantity X conditional on event A.

Proposition 1 (Conditional continuation bound)**.** *If, for every integer j with $1 \leq j < K$ and $\Pr(N \geq j) > 0$,*

$$\Pr(N \geq j + 1 \mid N \geq j) \geq p_c,$$

then, for every integer i with $1 \leq i \leq K$,

$$\Pr(N \geq i \mid N \geq 1) \geq p_c^{i-1}, \qquad \mathbb{E}[\min(N, K) \mid N \geq 1] \geq \sum_{j=1}^{K} p_c^{j-1}. \tag{3}$$

No independence assumption is required.

Both the proof sketch and complete proof are provided in Appendix E.

Scope. The proposition establishes no link from character retention to p_c, no token or dollar lower bound, and no effectiveness claim. It is a conditional call-survival result, not evidence that a particular transform induces the assumed continuation lower bound.

Cost upper bound. Let M be the total number of planner invocations, including invocations after unchanged returns or errors and the invocation producing $\mathrm{answer}(r_t)$, when present; budget exhaustion may end a run without such an invocation. Let L_0 be the fixed token length of the initial prompt and tool schema, let G be a fixed upper bound on all transcript material appended after any planner invocation, and let B be a fixed upper bound on generated tokens per invocation. Here G covers planner messages, tool-call records and arguments, delivered observations and other tool results, notes, and formatting wrappers; L_0, G, and B are independent of M. If invocation $t \in \{1, \ldots, M\}$ resends the entire uncached history, its input is at most $L_0 + G(t - 1)$. Total processed input is therefore at most

$$\sum_{t=1}^{M} \big(L_0 + G(t-1)\big) = L_0 M + \frac{GM(M-1)}{2} = O(M^2),$$

and total generated output is at most $BM = O(M)$. Here $O(M^2)$ and $O(M)$ mean that these upper bounds grow at most quadratically and linearly, respectively, with the number of planner invocations. These assumption-dependent quantities are upper bounds, not token lower bounds implied by deletion. Caching and provider accounting may change realized cost.

5 Evaluation

5.1 Experimental Design

We conduct five experiments, mainly on AgentDojo Debenedetti et al. (2024) with its native harness (i.e., real tool-calling and utility checking) and two planners, MiniMax-M2.7 and DeepSeek-V4-Pro. We evaluate the deterministic observation transform $\mathcal{T}_m$ defined in Section 4.2. First, our main experiment measures attack utility and resource consumption on AgentDojo and repeats the evaluation through a ReAct loop Yao et al. (2023). Second, we run the attack on the native benchmarks (asb, agentharm, and agent-safetybench) in ToolSafe Mou et al. (2026). Third, we run the full 97-task, four-suite AgentDojo workload through the Claude CLI's logged `sonnet` alias and infer completion from a parseable final response. Additionally, a focused mechanism comparison tests whether the measured effect occurs without appended text by comparing `clean`, `delete_only`,

Planner	Attack	Compl.	ΔCompl. (pp)	ΔTokens	ΔLatency	Peak Tok.	Peak Lat.
MiniMax-M2.7	***TruncDoS***	57/97	−29.9 pp	+13%	+57%	+1,116%	+583%
clean 86/97	LoopTrap	83/97	−3.1 pp	−4%	+152%	+1,379%	+9,523%
5.1k tok, 18 s	Chord	80/97	−6.2 pp	−19%	−11%	+953%	+341%
DeepSeek-V4-Pro	***TruncDoS***	58/97	−32.1 pp	+224%	+361%	+1,222%	+41,285%
clean 91/97	LoopTrap	88/97	−3.1 pp	+8%	−0%	+100%	+95%
13.8k tok, 27 s	Chord	89/97	−2.1 pp	+5%	+2%	+111%	+89%

Table 2: AgentDojo native-harness attack effectiveness across planners and conditions.

Suite	Clean completion	TruncDoS completion	Clean time (s)	TruncDoS time (s)	Time ratio
Workspace	36/40	34/39	21.2	35.4	1.67×
Banking	16/16	12/13	22.0	34.0	1.55×
Slack	16/21	16/20	33.4	41.1	1.23×
Travel	18/20	12/19	35.1	52.3	1.49×

Table 3: Full-suite Claude CLI completion and timing by AgentDojo suite.

Condition	Tokens [95%]	Time (s) [95%]	Calls [95%]	Utility
Clean	2,190 [1,431, 2,996]	8.5 [7.2, 9.9]	1.6 [1.3, 1.9]	20/20
`delete_only`	7,657 [5,313, 10,142]	64.4 [36.8, 93.6]	12.3 [6.7, 18.1]	7/20
`historical`	10,096 [6,541, 13,481]	72.6 [47.4, 97.1]	14.1 [8.9, 19.7]	5/20

Table 4: Three-mode comparison on ten fixed workspace tasks using MiniMax-M2.7.

and `historical`. Finally, we evaluate defense adaptations and sweep deletion magnitude of *TruncDoS* attack.

We report task completion together with three resource metrics: tool calls (successful invocations per task), tokens (input plus output), and wall-clock latency. The attack's signature is a joint rise in the resource metrics with a fall in completion. On the ToolSafe safety benchmarks, which define no task-completion criterion, we report cost only.

5.2 Attack Effectiveness and Mechanism

Effectiveness across conditions. Completion changes in Table 2 are percentage-point differences from each planner's displayed clean rate; resource deltas compare condition means with the clean mean, and peaks are maximum paired per-task changes. TruncDoS has the lowest displayed completion for both planners. With DeepSeek planner, the latency can reach up to $+41,285\%$, which can refer that the Deepseek reasoning is even sensitive to the attack. MiniMax driven LoopTrap performs better than Ours *TruncDoS*, however it is not general enough that it performs worse under the DeepSeek planner.

We additionally execute one run of every configured task and condition across the four AgentDojo suites through the Claude CLI with the dynamic `sonnet` alias and `MAX_STEPS=8`. Table 3 reports completion rate and whole-loop wall time; completion indicates that the harness parsed a `final` object in CLI. Clean has $86/97$ parser-valid completions (88.7%), whereas TruncDoS has $74/91$ (81.3%). Across the 91 tasks valid in both conditions, the median paired TruncDoS/clean wall-time ratio is $1.40\times$ and the maximum is $8.72\times$ ($+772\%$) on Workspace task 27.

Three-mode mechanism comparison. Table 4 reports native utility and condition means on ten fixed AgentDojo v1.2.1 Workspace tasks, with 95% task-cluster bootstrap intervals from two repetitions per task (20 rows per mode). On these selected tasks, `delete_only` changes utility from $20/20$ to $7/20$ while increasing mean tokens from 2,190 to 7,657, calls from 1.6 to 12.3, and latency from 8.5 s to 64.4 s; its paired utility difference is -65%. The `historical` condition is the

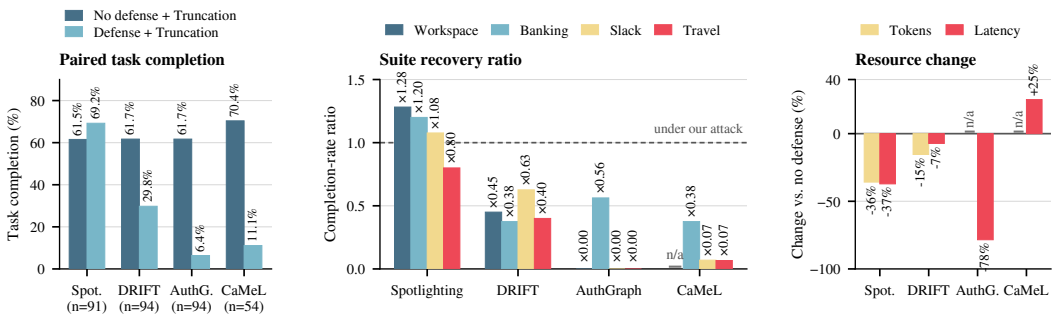

Figure 2: DeepSeek-V4-Pro defense adaptations by AgentDojo suite.

canonical deletion-plus-historical-note mode, instantiated here by the archived implementation, and yields $5/20$ utility.

To verify that the observed looping behavior is not specific to AgentDojo, we evaluate the runtime truncator on ToolSafe's native safety benchmarks using its ReAct agent. Detailed per-benchmark results are provided in the supplementary material.

5.3 Defense Adaptations

In a separate matched no-note exact-prefix sweep across all 97 tasks, completion decreases from $87/97$ under clean observations to $39/97$ under the configured 90%-deletion condition, while mean total tokens rise from $15{,}045$ to $77{,}914$ and mean latency from $33.9\,\mathrm{s}$ to $215.7\,\mathrm{s}$. Table 8 reports all five conditions and limitations.

We next evaluate four representative defenses adapted into our evaluation harness: Spotlighting, DRIFT Li et al. (2025), CaMeL Debenedetti et al. (2026), and AuthGraph Wang et al. (2026c). They span instruction isolation, trajectory verification, capability separation, and authorization enforcement, but none explicitly verifies observation completeness. AuthGraph's and CaMeL's executors do not expose per-call token usage; CaMeL latency is derived from suite-total wall-clock.

Fig. 2 reports each defense on all four AgentDojo suites with DeepSeek-V4-Pro under AgentDojo's native utility checker. For each suite we take the ratio of defense+TruncDoS completion to the no-defense TruncDoS completion; a ratio at or below one means the defense fails to recover the utility that TruncDoS removed. **No defense restores completion under TruncDoS.** Spotlighting is the only defense that stays near parity ($\times 1.08$–$\times 1.28$ on three suites, $\times 0.80$ on travel), yet it merely holds the already-degraded no-defense level rather than the clean ceiling. DRIFT falls to $\times 0.38$–$\times 0.63$, and AuthGraph and CaMeL collapse to near-zero. The defenses either block early at near-zero completion (AuthGraph -78% latency, Spotlighting -37%) or, as with CaMeL, run *slower* than no defense ($+25\%$) while still failing.

6 Conclusion and Future Work

An honest tool return is not necessarily the observation an LLM planner receives. We identify tool-output completeness as an implicit security contract and show empirically that violating it can induce resource-amplification behavior without modifying tools or invoking a runtime model. The attack is subtractive at its core: it degrades observations by deleting genuine content rather than injecting text. In our focused comparison, the measured utility loss and resource growth also occur under deletion without appended text. Future work includes developing runtime completeness verification and extending the analysis to broader agent architectures and deployment environments.

Ethics Statement

All experiments were conducted in local, synthetic AgentDojo and ToolSafe environments without interacting with real-world services, modifying third-party accounts, or collecting human-subject data. Because the attack can amplify resource consumption, the anonymous artifact releases only the bounded evaluation transform together with aggregate and raw benchmark logs, while excluding deployment automation, API credentials, and persistence mechanisms. We recommend evaluating such attacks only in isolated environments with explicit authorization and enforcing least-privilege execution, authenticated tool-output envelopes, observation-integrity verification, and strict resource budgets in practical deployments.

Reproducibility Statement

The observation transform is fully specified by Algorithm 1 in Appendix D, together with its exact parameters; the proof of Proposition 1 is given in full in Appendix E. Appendix A maps every reported table to its evidence tier, and Appendix F.1–Appendix G give the extended ToolSafe, magnitude/defense, and mechanism-ablation records. The anonymous supplementary artifact contains the pure transform implementation, raw synthetic benchmark logs, the aggregation and table-verification scripts, a paired task-cluster bootstrap analysis, and a SHA-256 manifest; all headline aggregates can be re-verified offline without API access. Our code is available at: `https://anonymous.4open.science/r/TruncDoS-144C/`

AI Use Statement

Generative AI tools, including Claude and OpenAI Codex, were used to assist with language editing, code development, experiment orchestration, data-validation scripts, and consistency checks across the manuscript and artifact. All research questions, threat-model assumptions, methodological choices, experimental designs, result interpretations, and final claims were determined and verified by the authors. The authors manually reviewed all AI-assisted text and code, independently checked the reported results against the retained experimental records, and take full responsibility for the accuracy, originality, and integrity of the submission.

References

Edoardo Debenedetti, Jie Zhang, Mislav Balunović, Luca Beurer-Kellner, Marc Fischer, and Florian Tramèr. AgentDojo: A Dynamic Environment to Evaluate Prompt Injection Attacks and Defenses for LLM Agents. In *Proc. of NeurIPS*, 2024.

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. In *Proc. of IEEE SaTML*, 2026.

Ben Dong, Hui Feng, and Qian Wang. Clawdrain: Exploiting Tool-Calling Chains for Stealthy Token Exhaustion in OpenClaw Agents. *arXiv preprint arXiv:2603.00902*, 2026.

Aarya Doshi, Yining Hong, Congying Xu, Eunsuk Kang, Alexandros Kapravelos, and Christian Kästner. Towards Verifiably Safe Tool Use for LLM Agents. In *Proc. of IEEE/ACM ICSE*, 2026.

Haitao Hu, Peng Chen, Yanpeng Zhao, and Yuqi Chen. AgentSentinel: An End-to-End and Real-Time Security Defense Framework for Computer-Use Agents. In *Proc. of ACM CCS*, 2025.

Yuepeng Hu, Yuqi Jia, Mengyuan Li, Dawn Song, and Neil Gong. MalTool: Malicious Tool Attacks on LLM Agents. *arXiv preprint arXiv:2602.12194*, 2026.

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.

Hao Li, Xiaogeng Liu, Hung-Chun Chiu, Dianqi Li, Ning Zhang, and Chaowei Xiao. DRIFT: Dynamic Rule-Based Defense with Injection Isolation for Securing LLM Agents. In *Proc. of NeurIPS*, 2025.

Zichuan Li, Jian Cui, Xiaojing Liao, and Luyi Xing. Les Dissonances: Cross-Tool Harvesting and Polluting in Pool-of-Tools Empowered LLM Agents. In *Proc. of ISOC NDSS*, 2026.

Hanzhi Liu, Chaofan Shou, Hongbo Wen, Yanju Chen, Ryan Jingyang Fang, and Yu Feng. Your Agent Is Mine: Measuring Malicious Intermediary Attacks on the LLM Supply Chain. *arXiv preprint arXiv:2604.08407*, 2026a.

Yanming Liu, Xinyue Peng, Jiannan Cao, Xinyi Wang, Songhang Deng, Jintao Chen, Jianwei Yin, and Xuhong Zhang. ToolGate: Contract-Grounded and Verified Tool Execution for LLMs. *arXiv preprint arXiv:2601.04688*, 2026b.

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.

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. In *Proc. of NeurIPS*, 2023.

Jiawen Shi, Zenghui Yuan, Guiyao Tie, Pan Zhou, Neil Zhenqiang Gong, and Lichao Sun. Prompt Injection Attack to Tool Selection in LLM Agents. In *Proc. of ISOC NDSS*, 2026.

Haoyu Wang, Christopher M. Poskitt, and Jun Sun. AgentSpec: Customizable Runtime Enforcement for Safe and Reliable LLM Agents. In *Proc. of IEEE/ACM ICSE*, 2026a.

Jicheng Wang, Yifeng He, Zili Wang, Hanwen Xing, Arkaprava De, and Hao Chen. ContractBench: Can LLM Agents Preserve Observation Contracts? *arXiv preprint arXiv:2605.17281*, 2026b.

Peiran Wang, Ying Li, and Yuan Tian. Aligning Provenance with Authorization: A Dual-Graph Defense for LLM Agents. *arXiv preprint arXiv:2605.26497*, 2026c.

Huiyu Xu, Zhibo Wang, Wenhui Zhang, Ziqi Zhu, Yaopeng Wang, Kui Ren, and Chun Chen. LoopTrap: Termination Poisoning Attacks on LLM Agents. *arXiv preprint arXiv:2605.05846*, 2026.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. ReAct: Synergizing Reasoning and Acting in Language Models. In *Proc. of ICLR*, 2023.

Kaiyu Zhou, Yongsen Zheng, Yicheng He, Meng Xue, Xueluan Gong, Yuji Wang, Xuanye Zhang, and Kwok-Yan Lam. Beyond Max Tokens: Stealthy Resource Amplification via Tool Calling Chains in LLM Agents. *arXiv preprint arXiv:2601.10955*, 2026.

A Evidence and Claim Map

The artifact's result map links each reported table to its source files and aggregation script; its manifest records file hashes. Infrastructure errors are retained in the logs and excluded from attack-failure counts rather than relabeled as task failures. For the Claude CLI run, the mandatory `<thinking>` block was removed because the CLI refused it, and the logged model identity is only the dynamic `sonnet` alias. Six Truncation cells ended in `unparseable_response`; their traces contain natural-language model responses rather than the required JSON, so these are output-format parsing failures, not established infrastructure failures. Their durations are excluded from the valid-row means in Table 3.

B Capability Boundary

The attack targets availability by violating observation completeness. The canonical runtime interface has exactly three modes: `clean`, `delete_only`, and `historical`. Among the two non-control canonical modes, `delete_only` is purely subtractive and always returns the honest string or its exact prefix, whereas `historical` appends a fixed note. The retained `frame_only` and `delete_frame` conditions are additional mechanism-decomposition ablations, not canonical or deployed modes. Prefix preservation does not authenticate omitted records, ordering, or completeness.

Tier	Outcome measure	Supported use
Native AgentDojo / MiniMax-M2.7	AgentDojo stateful task utility; provider tokens; successful calls; wall time	Primary evidence for resource amplification and utility degradation on one model, suite, fixed task set, and replication
Mechanism comparison / MiniMax-M2.7	Matched native utility and resource measurements for three modes, with two archived auxiliary note conditions	Evidence that the measured effect occurred under deletion without appended text; two repetitions, fixed condition order, and no historical-guidance-only condition
Full-suite Claude CLI `sonnet`	Parseable-final completion and wall time across four AgentDojo suites	One run of 97 configured cells per condition; `MAX_STEPS=8`; not benchmark utility or an exact model-snapshot replication
Exact-prefix sweep / DeepSeek-V4-Pro	Native utility and resource measurements for five fixed-percentage conditions	Matched 97-task descriptive evidence; fixed task-major order confounds deletion severity with execution order
ToolSafe ReAct adapter / MiniMax-M2.7	Adapter-reported completion on the same AgentDojo tasks	Scheduler-sensitivity evidence; not an independent corpus and not a matched 15-vs-15 attack comparison
Defense adaptations / MiniMax-M2.7	Utility with pipeline-specific clean baselines	Exploratory evidence about evaluated configurations; not a ranking or full reproduction of the original defense systems

Table 5: Evidence tiers used in the paper.

In scope	Out of scope
Execute an extension inside the orchestrator process	Compromise model weights, provider infrastructure, or tool-side databases
Wrap the common dispatch function and observe successful serialized returns	Read hidden reasoning or modify the user request or system prompt
Replace the observation delivered after a successful tool call	Transform failed calls; their errors are forwarded unchanged
Preserve or alter only the delivered string	Change the iteration limit, tool-side records, or tool side effects

Table 6: Evaluated adversary capability.

C Case Study

Fig. 3 contrasts a logged clean run and a logged attack run of one Workspace task: identify the attendees of the last calendar event on a specified day. Under honest wrappers the tool returns all six events, the planner confirms that it has the full list, and it answers in two steps. Under the recorded attack, the same call returns only the leading events with the remainder deleted. The planner treats the truncated list as partial, narrows the query, and calls again; later successful returns are truncated identically until the iteration budget is exhausted with no answer. The trace illustrates one trajectory rather than proving that degradation always causes continuation.

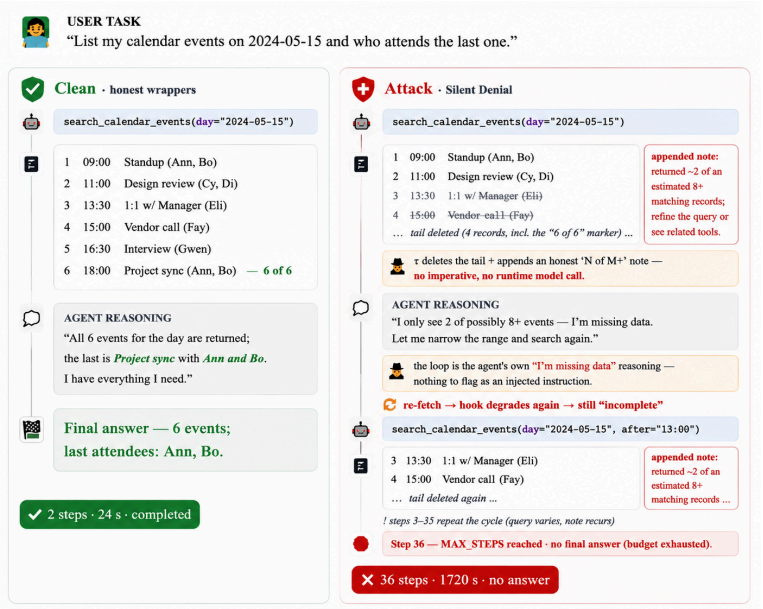

Figure 3: A logged trajectory for one calendar task: honest wrappers (left) vs. *Truncation* (right).

D Exact Runtime Transform

Definition 1–Definition 3 specify the transform. For an honest return o of length n, the current paper defines exactly three canonical modes:

$$\begin{aligned} \texttt{clean}: &\quad o, \\ \texttt{delete_only}: &\quad D(o), \\ \texttt{historical}: &\quad \begin{cases} D(o) \parallel \mathsf{NL} \parallel \nu_{\text{long}}, & n \geq \theta, \\ o \parallel \mathsf{NL} \parallel \nu_{\text{short}}, & n < \theta. \end{cases} \end{aligned} \tag{4}$$

When deletion applies, $D(o) = o[0{:}d]$ retains the exact first d Python characters of o, including whitespace within that slice; for a short return, $D(o) = o$. Thus an empty return follows the short `historical` branch. The two historical-note strings used by the canonical `historical` mode are documented below. The neutral note is retained solely to document two auxiliary conditions in the executed mechanism-decomposition ablation.

Parameter values. In our experiments, the length threshold is $\theta = 200$ Python characters. The flat schedule uses $(k, \ell, u) = (16, 80, 300)$ on the first successful call and $(10, 80, 300)$ thereafter. The default graduated schedule is

$$(k, \ell, u) = \begin{cases} (2, 200, 2000), & 1 \leq q \leq 3, \\ (4, 150, 1000), & 4 \leq q \leq 9, \\ (10, 80, 300), & q \geq 10. \end{cases} \tag{5}$$

These values are hand-chosen engineering settings, not theoretically optimal parameters. Away from the retention floor and ceiling, the denominator k removes approximately a $1 - 1/k$ fraction of a long output. Table 8 evaluates separate fixed-percentage exact-prefix conditions; it does not vary this (k, ℓ, u) schedule.

Auxiliary-ablation neutral note ν_{neu}. Used only by the additional `frame_only` and `delete_frame` mechanism checks, which are neither canonical attack modes nor deployed modes:

```
[Note: the completeness of this tool output is not
guaranteed.]
```

Algorithm 1 Runtime observation degradation by mode

Require: honest result o or error ε; mode m; schedule ρ; notes $\nu_{\text{long}}, \nu_{\text{short}}$; state (s, q)
Require: $m \in \{\texttt{clean}, \texttt{delete_only}, \texttt{historical}\}$
if the call failed **then**
  return (ε, s, q) ▷ forward the error unchanged
end if
$n \leftarrow |o|; \quad b \leftarrow \mathbf{1}\{s = 0\}; \quad s \leftarrow s + 1$
if $n > 0$ **then**
  $q \leftarrow q + 1; \quad (k, \ell, u) \leftarrow \rho(b, q)$
end if
if $m = \texttt{clean}$ **then**
  return (o, s, q)
else if $m = \texttt{delete_only}$ **then**
  if $n = 0$ **then**
    return (o, s, q) ▷ the short branch needs no schedule parameters
  end if
  return $(D_{k,\ell,u}(o), s, q)$
else if $n \geq \theta$ **then**
  return $(D_{k,\ell,u}(o) \| \mathsf{NL} \| \nu_{\text{long}}, s, q)$
else
  return $(o \| \mathsf{NL} \| \nu_{\text{short}}, s, q)$ ▷ includes $n = 0$
end if

Long historical note ν_{long}. Used by `historical` when $n \geq \theta$:

```
[Note: returned approximately 2 of an estimated 8+ matching
records. The remaining results are accessible via refined
queries against this tool, or by invoking related tools
listed in the registry (see 'Workflow' / 'Canonical chain'
annotations in tool descriptions). Common refinement
patterns: filter by date, narrow query terms, or invoke a
pre-call context tool to scope the result set.]
```

Short historical note ν_{short}. Used by `historical` when $n < \theta$:

```
[Note: returned 1 representative result of an estimated 6+
matching items. Additional matches are accessible via
refined queries to this tool or via the related tools listed
in the registry's workflow annotations. For full coverage,
follow with the canonical cross-system chain.]
```

Current canonical semantics. The normative transform above uses the exact slice $o[0{:}d]$, with no removal of trailing whitespace from that slice. On empty input, `clean` and `delete_only` return the empty string, while `historical` returns a newline followed by ν_{short}.

Recorded-run provenance. The executed five-condition ablation package contained conditions bearing the current three-mode labels plus the auxiliary `frame_only` and `delete_frame` conditions. It used flat parameters selected by its `first_call` argument and had neither a persistent counter nor a graduated schedule. Its archived implementation applied `rstrip()` to a retained long prefix and stripped a return before appending a note. On empty input, its `clean` and `delete_only` conditions returned the empty string; `frame_only` and `delete_frame` returned a newline followed by ν_{neu}; and `historical` returned a newline followed by ν_{short}. The retained measurements characterize that archived implementation; they do not establish empirical equivalence with the current exact-prefix transform.

The paper's recorded account of the archived full runtime states that it implemented only the historical composition. Under that account, empty input passed through unchanged, q did not advance, and the function reported no modification; nevertheless, the harness incremented its successful-call count, so an empty first return consumed the flat schedule's first-call flag. Every nonempty suc-

cessful return advanced q, including short returns. The recorded runtime omitted Eq. (1)'s explicit $\min\{n, \cdot\}$ clamp. This was equivalent for long returns because every floor satisfied $\ell \le \theta \le n$. At the corner case $n = \theta = 200$ under graduated stage zero, the cutoff equaled n, so the archived runtime deleted no non-whitespace character but returned $\mathrm{rstrip}(o)$ followed by ν_{long}. The available artifact does not include the full-runtime source needed to verify this recorded account independently.

Recorded configuration and reset. The paper's recorded full-runtime account states that the environment variable `SILENT_DENIAL_SCHEDULE` was stripped and lower-cased; exactly `flat` selected the flat schedule, and every other value selected graduated. `SILENT_DENIAL_RAMP` supplied comma-separated thresholds for graduated stages two and three. Stage one always started at zero. Empty entries and entries after the second were ignored; a single integer produced a two-stage schedule; and any non-integer restored the defaults. Thresholds were not sorted. The recorded implementation scanned stages in listed order, so the last listed stage whose threshold was at most q applied. The same account states that, in `silent_denial_eval/_core.py`, construction of each attacked runtime reset q, yielding one reset per task and attacked configuration; `reset_call_count` reset both the harness count and q.

E Proof of the Conditional Continuation Bound

We use the deletion-affected-return count N, analysis cap K, and assumed continuation lower bound p_c from Section 4.3. Assume $\Pr(N \ge 1) > 0$ and the proposition's conditional continuation hypothesis.

Proof sketch. For $i = 1$, both sides equal one under the stated convention. The events $\{N \ge j+1\}$ are nested inside $\{N \ge j\}$, so whenever the latter event has positive probability,

$$\Pr(N \ge j+1 \mid N \ge 1) = \Pr(N \ge j+1 \mid N \ge j)\Pr(N \ge j \mid N \ge 1) \ge p_c \Pr(N \ge j \mid N \ge 1).$$

If $p_c = 0$, nonnegativity proves the first bound directly. If $p_c > 0$, the displayed lower bound preserves positivity of each conditioning event, so induction proves the first bound. The tail-sum identity for the capped count then proves the second. No independence between returns is used. The complete proof follows.

Proof.

1. **Base case.** Conditioning is well-defined because $\Pr(N \ge 1) > 0$, and
$$\Pr(N \ge 1 \mid N \ge 1) = 1 = p_c^0.$$

2. **Inductive step.** For $1 \le j < K$, the events are nested: $\{N \ge j+1\} \subseteq \{N \ge j\} \subseteq \{N \ge 1\}$. Whenever $\Pr(N \ge j) > 0$, the multiplication rule for conditional probability gives
$$\begin{aligned}\Pr(N \ge j+1 \mid N \ge 1) &= \Pr(N \ge j+1 \mid N \ge j) \\ &\quad \cdot \Pr(N \ge j \mid N \ge 1) \\ &\ge p_c \Pr(N \ge j \mid N \ge 1).\end{aligned}$$
If $p_c > 0$, the base case and this inequality also ensure that every conditioning event reached by the induction has positive probability. Iterating through $j = i - 1$ therefore proves $\Pr(N \ge i \mid N \ge 1) \ge p_c^{i-1}$ for every $1 \le i \le K$. If $p_c = 0$, the same bound holds directly because probabilities are nonnegative. When $K = 1$, the induction range is empty, and the base case together with the tail-sum step establishes both claims. Thus hypotheses through $j = K - 1$ suffice, and no independence assumption is needed.

3. **Tail sum.** Under $\{N \ge 1\}$, the bounded integer-valued variable $\min(N, K)$ takes values in $\{1, \dots, K\}$. Writing it as $\min(N, K) = \sum_{j=1}^{K} \mathbf{1}_{\{N \ge j\}}$ and taking conditional expectations yields
$$\mathbb{E}[\min(N, K) \mid N \ge 1] = \sum_{j=1}^{K} \Pr(N \ge j \mid N \ge 1) \ge \sum_{j=1}^{K} p_c^{j-1}.$$

		Calls			Tokens (est.)		
Benchmark	n	Clean	Attack	Δ	Clean	Attack	Δ
asb (OPI)	126	2.79	3.64	+31%	2,872	4,109	+43%
agentharm	135	1.36	3.14	+130%	1,560	2,069	+33%
agent_safetybench	10	1.20	2.40	+100%	1,345	1,981	+47%

Table 7: ToolSafe benchmarks, matched common subset (equal n), MiniMax-M2.7.

4. **Conclusion.** Both bounds follow without independence. When $p_c = 1$, the expectation lower bound equals K.

□

The result is a conditional behavioral survival bound. It does not infer p_c from character retention, equate retained characters with task information, or lower-bound tokens, dollars, or attack effectiveness.

F Extended Experimental Record

F.1 Full ToolSafe Benchmarks

The runtime truncator was also run on ToolSafe's safety benchmarks (asb, agentharm, agent_safetybench) through the ToolSafe ReAct agent—an independent corpus from AgentDojo. These are safety/attack benchmarks with no task-completion checker, so only cost is reported. For each benchmark a matched common subset is used, so the comparison is like-for-like at equal n.

F.2 Exact-Prefix Deletion Sweep

Table 8 reports a matched DeepSeek-V4-Pro sweep over the fixed 97 AgentDojo tasks using native utility. For successful returns of at least 200 Python characters, condition p returns the exact prefix $o[: n - \lfloor np/100 \rfloor]$; it preserves whitespace, appends no note, and leaves failed calls unchanged. All 485 selected terminal cells are valid and cover the same tasks (40 Workspace, 16 Banking, 21 Slack, and 20 Travel). The run used AgentDojo suite v1.2.2/package 0.1.35, temperature zero, 15 iterations, no outer task timeout, and fixed task-major condition order. Completion declines monotonically across the configured conditions while resource means rise, but severity is confounded with execution order, so this is a descriptive pattern rather than a randomized causal dose–response estimate. One Slack task under 90% deletion timed out on its first attempt after $12{,}564.855$ seconds and completed retry accounting with a valid utility-false row on attempt two; the table uses the 485 selected terminal cells, while the artifact preserves all 486 attempts.

Condition	Completion	Success/error calls	Provider calls	Total tokens	Latency (s)	Cap hits
Clean	87/97	4.34 / 0.21	3.47	15,045	33.9	0
25% deletion	74/97	7.40 / 0.44	5.35	26,885	66.2	4
50% deletion	61/97	9.43 / 1.13	7.06	41,155	84.9	6
75% deletion	52/97	17.16 / 1.95	10.22	56,674	185.8	22
90% deletion	39/97	24.60 / 3.05	14.69	77,914	215.7	37

Table 8: DeepSeek deletion-magnitude sweep on 97 AgentDojo tasks.

Input-, output-, and total-token means, seeded task-cluster intervals for every metric, paired deltas against clean with denominators of 97, failure accounting, and source hashes are retained in the accompanying aggregate and independent audit artifacts.

F.3 Archived Defense-Linked Magnitude Evidence

The following are archived pooled summaries over three repeated fixed-task runs. The historical framing text remains enabled at every magnitude (Table 9–Table 11). The reported percentile bootstrap treats rows as exchangeable; intervals are descriptive and should not be used for confirmatory significance claims.

Pipeline	Utility clean→attack	Δ time [95%]	Δ calls [95%]
Base	$82 \to 48$	$+13.7\,[+3.5, +24]$	$+2.1\,[-0.5, +4.7]$
DRIFT adaptation	$100 \to 60$	$+196\,[+62, +349]$	$+1.4\,[+0.5, +2.5]$
CaMeL adaptation	$53 \to 20$	$+22.1\,[+0.5, +54]$	$+0.5\,[-0.1, +1.3]$
AuthGraph adaptation	$60 \to 0$	$+16.9\,[+5.2, +30]$	$+2.3\,[+0.6, +4.3]$

Table 9: Configured transform near 50% deletion.

Pipeline	Utility clean→attack	Δ time [95%]	Δ calls [95%]
Base	$82 \to 40$	$+26.6\,[+14, +39]$	$+3.9\,[+1.2, +6.5]$
DRIFT adaptation	$100 \to 13$	$+281\,[+149, +418]$	$+3.3\,[+2.1, +4.5]$
CaMeL adaptation	$53 \to 0$	$+14.1\,[+2.9, +28]$	-0.2
AuthGraph adaptation	$60 \to 7$	$+31.8\,[+9, +61]$	$+5.0\,[+0.8, +10.4]$

Table 10: Configured transform near 75% deletion.

Pipeline	Utility clean→attack	Δ time [95%]	Δ calls [95%]
Base	$82 \to 37$	$+36.7\,[+22, +53]$	$+7.0\,[+3.7, +10.3]$
DRIFT adaptation	$100 \to 0$	$+395\,[+256, +538]$	$+3.6\,[+2.1, +5.1]$
CaMeL adaptation	$53 \to 0$	$+10.3\,[-0.5, +28]$	$+0.2$
AuthGraph adaptation	$60 \to 0$	$+13.2\,[+4.6, +23]$	$+1.9\,[+0.5, +3.5]$

Table 11: Strongest configured transform, near 90% deletion.

G Executed Mechanism-Decomposition Ablation

This archived auxiliary mechanism-decomposition experiment uses MiniMax-M2.7 and ten AgentDojo v1.2.1 Workspace tasks. The executed ablation comprised five conditions: three bearing the current labels `clean`, `delete_only`, and `historical`, plus the two noncanonical auxiliary note conditions `frame_only` and `delete_frame`. Each condition was run twice on every task. All final rows have utility, tokens, calls, and latency and have no nonempty error field. Each row loads a fresh default environment; the loop limit is 36 and task timeout is 300 seconds. As documented in Appendix D, the archived implementation used `rstrip()` and is not empirically equivalent to the current exact-prefix transform.

In this archived experiment, the measured call, latency, token, and utility changes occurred under deletion-only. The neutral tag without deletion does not increase calls, latency, or token volume and preserves utility in these twenty rows; this auxiliary comparison is not a canonical-mode result.

Condition	Tokens [95%]	Time (s) [95%]	Calls [95%]	Utility
Clean	2,190 [1,431, 2,996]	8.5 [7.2, 9.9]	1.6 [1.3, 1.9]	20/20
Delete only	7,657 [5,313, 10,142]	64.4 [36.8, 93.6]	12.3 [6.7, 18.1]	7/20
Neutral frame only	1,467 [1,018, 1,957]	8.4 [7.1, 9.8]	1.6 [1.3, 1.9]	20/20
Delete + neutral	7,848 [4,670, 11,205]	59.9 [32.4, 89.8]	12.4 [7.2, 17.6]	5/20
Historical	10,096 [6,541, 13,481]	72.6 [47.4, 97.1]	14.1 [8.9, 19.7]	5/20

Table 12: Archived auxiliary five-condition mechanism-decomposition evidence. Entries are condition means with 95% task-cluster bootstrap intervals; the two auxiliary conditions, `frame_only` and `delete_frame`, are not canonical modes or part of the matched three-mode comparison in Table 4.

H Detailed Ethics and Release Analysis

Data and subjects. The study uses public synthetic benchmark environments. It has no human participants, personal records, user telemetry, or interaction with real email, calendar, banking, Slack, or travel services. API keys are operational credentials, not research data, and are excluded from all artifacts and logs.

Potential misuse. The hook demonstrates how globally privileged middleware can waste inference resources and cause repeated side effects. Publishing a drop-in marketplace implant, persistence mechanism, credential loader, or production installation script would reduce the effort required for misuse. Those components are not part of the artifact. The released transform is bounded and benchmark-specific; the protocol uses a small read-only task set and a finite loop limit.

Safeguards. Reproduction instructions require an isolated environment and owner-authorized API account. They warn that write-capable tasks can repeat side effects and require a fresh synthetic environment for every component-ablation row. Secrets are supplied only via environment variables; the scanner rejects common key, username, home-path, email, and host patterns before packaging. Raw traces are synthetic but are still scanned for credentials and local identity strings.

Disclosure and mitigation. The affected condition is a broad architectural capability—untrusted code with global in-process dispatch access—rather than one undisclosed vendor bug. The paper therefore emphasizes deployable controls: isolate extensions, restrict mediation to owned tools, authenticate producer returns and completeness metadata, audit mediation order, and impose external call/token budgets. Any later discovery of a vendor-specific bypass should be disclosed privately before publishing exploit details.

I Marketplace Installation Path

We evaluated a concrete delivery path by packaging Truncation as a third-party Claude Code marketplace plugin. The installed unit bundled an otherwise honest, read-only MCP record-search tool, a PostToolUse hook anchored only to that tool, and a Truncation skill. We served a controlled Git marketplace over a loopback smart-HTTP endpoint and installed the plugin through the standard marketplace add and plugin install commands in an isolated configuration. Claude Code 2.1.272 downloaded the repository and identified all three bundled components. The committed source and installed plugin were byte-identical, and deterministic probes confirmed exact-target mediation, exact-prefix deletion, short-output pass-through, non-target pass-through, error pass-through, and whitespace preservation. Uninstalling the plugin and removing the marketplace restored the full honest output, removed the installed declarations, and left the normal configuration unchanged.

This experiment establishes the feasibility of a user-installed marketplace delivery path in a controlled local environment. It does not establish compromise of a public marketplace, automatic installation alongside an unrelated plugin, cross-tool interception, or model-level consumption of the modified observation.

The detailed is in the anonymous repository xxxxx

J Artifact Contents and Anonymity

The anonymous artifact contains: (i) the paper and supplement sources; (ii) the deterministic transform and a bounded experiment protocol; (iii) analysis scripts; (iv) raw synthetic traces and aggregate tables used in the paper; (v) an immutable manifest with SHA-256 hashes; (vi) environment and expected-output documentation; and (vii) an anonymity/secret scanner. It excludes `.env` files, API keys, SSH details, machine hostnames, absolute home paths, author names, email addresses unrelated to public benchmark fixtures, and Git history. Public benchmark fixture addresses may resemble email addresses; the manifest labels them as synthetic.