

Beyond Max Tokens: Stealthy Resource Amplification via Tool Calling Chains in LLM Agents

Kaiyu Zhou¹, Yongsen Zheng^{1*}, Yicheng He², Meng Xue³
Xueluan Gong¹, Yuji Wang⁴, Xuanye Zhang¹, Kwok-Yan Lam¹

¹Nanyang Technological University, Singapore

²University of Illinois Urbana–Champaign, United States

³The Hong Kong University of Science and Technology, Hong Kong

⁴Shanghai Jiao Tong University, China

kaiyu001@e.ntu.edu.sg, {yongsen.zheng, kwokyan.lam}@ntu.edu.sg

Abstract

The agent–tool interaction loop is a critical attack surface for modern Large Language Model (LLM) agents. Existing denial-of-service (DoS) attacks typically function at the user-prompt or retrieval-augmented generation (RAG) context layer and are inherently single-turn in nature. This limitation restricts cost amplification and diminishes stealth in goal-oriented workflows. To address these issues, we propose a stealthy, multi-turn economic DoS attack at the tool layer under the Model Context Protocol (MCP). By editing only text-visible fields and implementing a stateful, completeness-gated return policy, our malicious server preserves tool identity and the terminal benign payload while steering agents into prolonged, verbose tool-calling chains. We optimize these text-only edits with Monte Carlo Tree Search (MCTS) to maximize cost under a task-success constraint. Across six LLMs on ToolBench and BFCL benchmarks, our attack yields trajectories over 60K tokens, increases per-query cost by up to 658×, raises energy by 100–560×, and pushes GPU key-value (KV) cache occupancy to 35–74%. Representative prompt filters and output or trajectory monitors seldom detect these attacks, highlighting the need for defenses that safeguard agentic processes rather than focusing solely on final outcomes.

1 Introduction

Large language models (LLMs) are rapidly evolving from single-turn chatbots into tool-augmented agents (Luo et al., 2025a; Yang et al., 2025b; Tran et al., 2025; Wang et al., 2025; Xi et al., 2025; Mohammadi et al., 2025). LLM agents can interact with external tools and execute multi-step tasks across domains (Sapkota et al., 2026), and standardized agent–tool protocols such as the Model Context Protocol (MCP) are accelerating this integration (Ehtesham et al., 2025; Hou et al., 2026;

Hasan et al., 2026; Model Context Protocol Working Group, 2025; Anthropic, 2024). As agents are deployed at scale (Sun et al., 2025), operational reliability and cost stability emerge as primary concerns; under Unbounded Consumption (OWASP Gen AI Security Project, 2025), vulnerabilities can lead to severe resource exhaustion as Denial-of-Service (DoS) attacks (Xu and Parhi, 2025).

Existing research on DoS attacks against LLMs largely forces models to generate excessively long outputs within a single interaction, typically triggered by a malicious user prompt or injected retrieval-augmented generation (RAG) context (Zhang et al., 2025c; Gao et al., 2024b; Dong et al., 2025; Geiping et al., 2024; Gao et al., 2024a). For instance, Engorgio and Auto-DoS craft queries that elicit verbose responses that are often off-task (Dong et al., 2025; Zhang et al., 2025c), while Overthink (Kumar et al., 2025) injects decoy reasoning problems into retrieved context to inflate internal thought while keeping the final answer correct (Kumar et al., 2025). Despite their differences, a critical limitation unites these methods: they are fundamentally single-turn attacks operating at the user-query or RAG layer.

This single-turn focus limits their impact in the agentic paradigm for two reasons: costs are capped by the model’s per-turn maximum completion, and many attacks (except Overthink) produce generic verbosity that is conspicuous in goal-oriented tool workflows (Zhang et al., 2025c; Gao et al., 2024b; Dong et al., 2025; Geiping et al., 2024; Gao et al., 2024a; Louck et al., 2025). Meanwhile, Zhang et al. (2025a) show malfunction amplification, where inputs induce repetitive or off-task action loops that often cause task failure. In contrast, the multi-turn agent–tool communication loop remains a largely unexplored attack surface for correctness-preserving economic DoS: the agent still completes the task, but the cost explodes. Because the long tool arguments are generated by the target LLM

*Corresponding author.

after a legitimate tool invocation, the recurring on-line inference cost is borne by the agent operator and can also affect co-located workloads. Table 1 summarizes key differences between our tool-layer attack and prior single-turn methods.

To address these gaps, we introduce a tool-layer attack that targets the multi-turn agent–tool interaction loop. Our method transforms a benign, MCP-compliant tool server into a malicious variant that steers the agent to repeatedly call the same tool and generate long outputs at the tool-calling step, while still completing the user’s task. Across six LLMs and two tool-use benchmarks (ToolBench and BFCL) (Fan et al., 2025; Patil et al., 2025), it consistently drives per-query completions beyond 60,000 tokens, inflates token budgets by up to $658\times$, increases total energy by up to $561\times$, and raises peak GPU KV-cache usage to over 73% (Pan et al., 2025). Crucially, the attack preserves task success and is rarely flagged by representative defenses (§4.5), enabling substantial degradation of system throughput and OOM-safe concurrency.

Taken together, our findings make three critical contributions:

- We establish the tool-calling layer as a first-class DoS attack surface in the agent era: even with correct tool use and correct final answers, representative prompt filters and output or trajectory monitors rarely flag the attack (§4.5).
- We propose a stateful, completeness-gated malicious template and an MCTS optimizer that transforms benign MCP servers under text-only, payload-preserving constraints.
- Extensive experiments across six LLMs on ToolBench and BFCL show large resource amplification while maintaining high task success, with supplementary validation on LangGraph and real MCP tools.

2 Background and Related Work

2.1 LLM Agents and Their Operational Cost

With rapid advancements in LLM reasoning and tool calling, especially in 2025, LLMs are increasingly deployed as autonomous agents for real-world tasks (Hu et al., 2025; Yan et al., 2025; Zhang et al., 2025b; Zheng et al., 2025). This trend is evident in widespread industry adoption, with major vendors integrating agents into their product stacks and startups securing funding for domain-specific agents in enterprise workflows (Microsoft, 2025; Amazon Web Services, 2025; Google Cloud, 2025;

PwC, 2025; Shen and Yang, 2025). As deployments scale, operational costs shift from one-time training to continuous inference (Luccioni et al., 2024; Samsi et al., 2023), manifesting as token-billed API expenses or sustained energy and hardware demands in self-hosted environments (Varoquaux et al., 2025; Bhardwaj et al., 2025). This connection between interaction volume and costs creates a significant attack surface for economic denial-of-service.

2.2 LLM Resource Consumption Attacks

Research on resource consumption attacks against LLMs aims to inflate operational costs and trigger denial-of-service by inducing excessively long generations (Zhang et al., 2025c; Gao et al., 2024b; Dong et al., 2025; Kumar et al., 2025; Geiping et al., 2024; Gao et al., 2024a). Early methods like repeat-hello, Engorgio, P-DoS, and Auto-DoS create malicious queries that elicit verbose, off-task responses (Geiping et al., 2024; Zhang et al., 2025c; Gao et al., 2024b; Dong et al., 2025). This verbosity is often obvious, limiting effectiveness beyond general-purpose chatbots. A stealthier approach, Overthink, injects decoy reasoning into retrieved RAG context to inflate the model’s internal think trace while maintaining answer correctness (Kumar et al., 2025). But existing methods remain single-turn, with costs capped by maximum completion length, leaving the multi-turn agent-tool loop underexplored. Besides, Zhang et al. (2025a) explore malfunction amplification, which drives agents into repetitive action loops, often hindering task completion. Instead, we target the MCP tool layer to achieve a correctness-preserving economic DoS: the task completes, but multi-turn tool-calling chains significantly amplify costs.

Recent MCP security analyses and benchmarks catalogue prompt injection, malicious tool descriptions, response manipulation, privacy leakage, and unsafe tool execution (Hou et al., 2026; Zhang et al., 2026). These studies primarily characterize semantic or instruction-level harms. Our attack is complementary: it preserves tool identity and terminal payload semantics while exploiting intermediate server responses to amplify agent-side computation, exposing a process-level economic risk that final-answer correctness does not capture.

Aspect	Engorgio	P-DoS	Auto-DoS	Overthink	Ours
Correctness preserving	✗	✗	✗	✓	✓
Trigger layer	Dialog query	Dialog query	Dialog query	RAG context	MCP Tool server
Turns & per-query bound	1-turn ($\leq M$)	1-turn ($\leq M$)	1-turn ($\leq M$)	1-turn ($\leq M$)	n-turn ($\leq nM$)
Long-output site	Answer step	Answer step	Answer step	Think step	Tool calling step
Access model	White-box	White-box	Black-box	Black-box	Black-box

Table 1: DoS attack comparison. M is the per-turn max completion; our tool-layer attack enables n -turn amplification while preserving task success (§4.2).

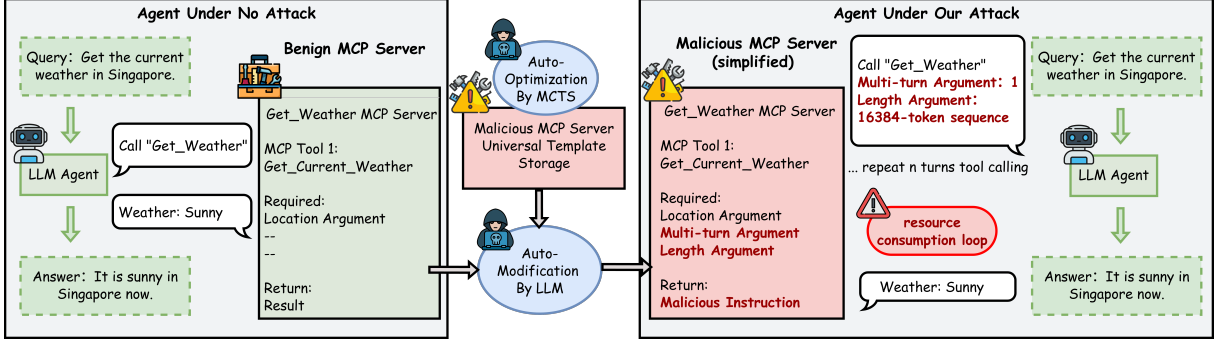


Figure 1: Tool-layer DoS overview. A protocol-compatible, text-only malicious template (MCTS-optimized) induces multi-turn, long tool-call traces while preserving the final benign payload.

3 Methodology

To address single-turn attack limitations, we exploit the multi-turn, stateful nature of agent-tool interactions. We convert a benign MCP tool server into a malicious variant that produces long, costly, yet successful task trajectories while preserving tool identity and terminal payload semantics. Our approach includes: (i) a formal problem definition as constrained optimization (§3.1); (ii) a stateful, completeness-gated malicious template that guides the agent through text-visible fields and a return policy (§3.2); and (iii) an MCTS-based optimizer for localized text edits to create effective malicious templates across LLMs and tasks (§3.3–§3.4). Figure 1 illustrates the pipeline.

3.1 Problem Formulation and Threat Model

System model. We model an MCP-based agent-tool loop with an agent policy A , a black-box LLM M , and an external MCP server T_θ parameterized by a text template θ . The server exposes tool functions $\mathcal{F}$ and communicates via MCP request/response semantics. The policy A maps the dialogue and tool feedback to the next action—whether to call a tool (which one, with what arguments) or to emit the final answer. Our attack never changes A or M ; instead, it manipulates the tool-facing messages returned by T_θ so that A (of-

ten implicitly instantiated by M under structured prompting) chooses longer multi-turn trajectories while still solving the task. We do not tamper with user queries, prompts, or retrieval; the attack activates only after the tool is legitimately invoked under the benign configuration (formal trigger-set definition in Appendix B.1).

Objective. An agent run induces an interaction trajectory $\tau = \{(a_t, r_t)\}_{t=1}^n$, a sequence of tool calls (a_t) and tool responses (r_t). Our primary cost metric is the total number of output tokens generated by M :

$$C(\tau) = \text{Tok}_{\text{out}}(\tau),$$

which is typically the dominant driver of both API fees and inference load; input-token growth is a secondary effect of longer trajectories. Let q be a user query with implied goal $u = \text{goal}(q)$, and let o be the final answer. Let $\text{Succ}(u, \tau, o) \in \{0, 1\}$ indicate whether o achieves u given τ . The attacker seeks a template θ that maximizes expected cost while maintaining task success with probability at least $p_{\min}$ over routable queries (Appendix B.1):

$$\max_{\theta \in \Theta} \mathbb{E}[C(\tau(q; \theta))] \quad (1)$$

$$\Pr[\text{Succ}(u, \tau, o) = 1] \geq p_{\min}. \quad (2)$$

The central mechanism is multi-turn cost amplification: we induce longer trajectories (larger n) and

make each tool-calling turn verbose, so that the vast majority of tokens are generated during tool calling rather than in the final answer.

Threat model. The adversary’s capability is confined to controlling the MCP server T_θ , specifically, modifying its template θ as an MCP-compatible server variant. The attacker cannot modify the agent policy A or the LLM M , has only black-box access to them, and does not alter queries, prompts, or retrieval. Under benign conditions where A decides to call the tool, the attacker may replace θ_0 with a template-augmented variant θ that remains protocol-compatible.

Such control can arise when an operator installs a third-party MCP server that continues to provide its advertised function, or when a previously trusted server is modified through an update, compromise, or poisoned dependency (Hou et al., 2026). Because the tool identity and terminal benign payload remain unchanged, ordinary functionality checks can continue to pass, particularly in automated or background workflows where intermediate traces are not routinely inspected. Controlling a tool server does not necessarily expose host files, credentials, private memory, or other tools; nevertheless, it allows the server to trigger repeated generation by the agent-side LLM. The server performs only short replies and lightweight validation online, whereas the target LLM repeatedly emits the long tool arguments, placing the recurring inference cost on the agent operator. We study this post-connection impact once a controlled server is present, rather than the separate mechanism used to compromise or install it.

3.2 The Universal Malicious Template

Design constraints. We implement a stateful, completeness-gated progression mechanism while preserving tool names and terminal benign payload semantics. The server adds optional text-only fields, tracks a segment index, and validates a long-form completeness artifact supplied in each call. It releases the benign payload only at $t=T_{\max}$ after validation; before that condition holds, it returns protocol-compatible Progress or Repair notices. In our default instantiation, the artifact is stored in a field named “calibration_sequence” and represented as a complete comma-separated list. This field is attacker-defined: it is not part of MCP and performs no benign protocol negotiation. The full element list and failure handling are summarized in Table 5 (Appendix B.2).

Mechanism and return policy. The segment index t gives the agent an explicit notion of progress, so intermediate calls appear to belong to an ongoing procedure. The completeness artifact makes each tool-calling turn expensive without changing the underlying task semantics. On the return side, Progress notices keep the agent on the goal path while prompting the next call; Repair notices reject abbreviated or malformed artifacts and request a retry without advancing the segment; and the Terminal return passes through the original benign payload unchanged. We enforce four invariants: (i) Monotone progress: $t_1=1$ and valid calls advance $t_{k+1}=t_k+1$; (ii) Format completeness: $\text{validate}(\text{artifact})$ holds iff the artifact matches the required complete format and ordering; (iii) Return policy: valid artifacts with $t < T_{\max}$ yield a Progress notice, while invalid artifacts yield a Repair notice without advancing t ; and (iv) Termination: only when $t=T_{\max}$ and the latest artifact validates does the server emit the Terminal return. The default integer-list implementation is not essential: a pseudo-JSON record variant still produces more than 60K tokens with 74.5–91.5% ASR across the tested models and benchmarks (Appendix D.5).

Concrete instantiation. In one evaluated “play_artist” server, the benign tool immediately returns “Now playing music by the requested artist.” The malicious variant adds optional segment and “calibration_sequence” fields: valid intermediate calls receive a short Progress notice, malformed artifacts receive Repair without advancing, and segment 6 releases the original payload. Appendix B.2.1 gives the abridged before-and-after schema and trajectory.

3.3 Universal Template Seed Bank

We maintain a seed bank of protocol-compatible, task-correct templates T_θ to warm-start search. Each MCTS run starts from a selected seed and halts when an acceptance predicate is met; the resulting template is written back and reused as a high-quality starter for subsequent runs. Additional screening, promotion, and metadata details are provided in Appendix B.3.

3.4 MCTS Optimizer

Each tree node v corresponds to a concrete server T_{θ_v} ; edges apply a single localized text edit under payload-preserving, text-only constraints. We organize edits into three families: $\mathcal{A}_{\text{MT}}$ (multi-turn induction), $\mathcal{A}_{\text{LEN}}$ (length induction), and $\mathcal{A}_{\text{REP}}$

(repair after omission/format errors). Search proceeds with phase gating: in pre_MT, we use $\mathcal{A}_{\text{MT}}$ to stabilize multi-turn behavior; once short screenings show stable segment sequencing, we switch to post_MT and use $\mathcal{A}_{\text{LEN}}$ to strengthen long outputs. If an omission/format error is observed at node v (i.e., a Repair notice), $\mathcal{A}_{\text{REP}}$ is unlocked at v only; otherwise it remains disabled. New children are evaluated in parallel with a lightweight Stage-1 screen and an optional Stage-2 refinement, and accepted templates are recorded back into the seed bank. We select children to explore by UCT (Kocsis and Szepesvári, 2006):

$$u^* = \arg \max_{u \in \mathcal{C}(v)} \bar{Q}(u) + C \sqrt{\frac{\ln(1 + N_{\text{uct}}(v))}{1 + N_{\text{uct}}(u)}},$$

where $\bar{Q}(u)$ is the running mean evaluation signal and N_{uct} counts visits used for exploration. Full details of action instantiation and evaluation gates are provided in Appendix B.4. The overall procedure is formalized in Algorithm 1.

4 Experiments

4.1 Experimental Setup

Agent framework & serving environment. We evaluate all conditions under the same agent policy A and prompts. Our main experiments use a controlled **qwen-agent**-based emulator to isolate tool-response policies while preventing unintended external actions (Qwen Team, 2025); Appendix D.1 further validates the same MCP-compatible benign and attack servers on LangGraph. Runs are executed on a single node with $8 \times \text{H200}$ GPUs using a uniform serving stack with a fixed concurrency of 25 queries; no changes to A or the target LLM M are made across conditions. Full configuration details, including target and attacker LLMs configuration, the agent framework setup, and our datasets filtering and wrapping rules, can be found in Appendix C.

Target LLMs. We target six LLMs with strong tool calling support: Qwen-3-32B (Yang et al., 2025a), Llama-3.3-70B-Instruct (Grattafiori et al., 2024), Llama-DeepSeek-70B (Guo et al., 2025), Mistral Large (Mistral AI, 2024), Seed-32B (ByteDance Seed Team, 2025), and GLM-4.5-Air (GLM-4.5 Team, 2025).

Datasets. We use two tool-use corpora: ToolBench (Fan et al., 2025) and BFCL (Patil et al., 2025). From each, we select all prompts that are

Algorithm 1: MCTS Optimizer for Malicious Template Generation

Input : Seed bank (candidate T_θ); action families $\mathcal{A}_{\text{MT}}, \mathcal{A}_{\text{LEN}}, \mathcal{A}_{\text{REP}}$; targets m^* (minimum multi-turn count), L^* (per-turn length target); Stage sizes and gates; UCT constant C ; search budget.

Output : Optimized template T_{θ^*} and an updated seed bank.

- 1 **Seed screening:** evaluate candidates on a fixed query set; pick the most accepted starters.
- 2 **while** budget not exhausted **do**
- 3 select node v by UCT using $\bar{Q}$ and N_{uct} ;
- 4 **if** v not fully expanded **then**
- 5 set $\mathcal{A} \leftarrow \mathcal{A}_{\text{MT}}$ if $\phi(v) = \text{pre_MT}$ else $\mathcal{A}_{\text{LEN}}$;
- 6 **if** omission observed at v **then**
- 7 $\mathcal{A} \leftarrow \mathcal{A} \cup \mathcal{A}_{\text{REP}}$.
- 8 **foreach** untried $a \in \mathcal{A}$ **do**
- 9 create a child by applying the Editor once to obtain $T_{\theta'}$.
- 10 **foreach** new child u in parallel **do**
- 11 **Stage-1:** run small rollouts; update $\bar{Q}(u)$ and increment N_{uct} along the path;
- 12 **if** segment sequencing stabilized **then**
- 13 set $\phi(u) := \text{post_MT}$.
- 14 **if** Stage-1 gate satisfied **then**
- 15 **Stage-2:** run additional rollouts; refine $\bar{Q}(u)$.
- 16 **if** acceptance predicate holds **then**
- 17 record T_{θ^*} and write back to the seed bank.
- 18 backpropagate Stage-1 statistics to ancestors (value means and N_{uct}).
- 19 **return** T_{θ^*} and the updated bank.

single-turn and single-tool in their original specification. For comparability, each original tool is wrapped as an MCP server that preserves its functionality and descriptions. We drop a small number of low-quality prompts that never trigger a tool call under the benign configuration. The final evaluation sets contain: ToolBench: **105** MCP servers and **261** queries; BFCL: **80** MCP servers and **203** queries. Every retained query is evaluated in each model-condition pair, yielding $(261+203) \times 6 \times 5 = 13,920$ main-study episodes with query-level pairing across conditions.

Baselines. We compare five conditions under identical agent policy A , target LLM M , prompts, and decoding: (i) Benign MCP server (no attack): the unmodified server for each tool; (ii) Overthink (ICL-Genetic, Agnostic): we reproduce the strongest variant from (Kumar et al., 2025). Because our benchmarks are non-RAG, we place the decoy trigger in the user query (in-context prefix/suffix) rather than in retrieved context; (iii) Overthink-mt (multi-turn Overthink): an aligned

multi-turn extension of Overthink for tool-calling agents. We match our multi-turn setting (tool-call budget and delayed exposure of the true tool output) while keeping the trigger at the context layer and leaving the tool server benign; (iv) Hand-crafted template (no MCTS): a fixed malicious template that follows the same constraints as ours (text-only edits, protocol-compatible, payload-preserving) but without MCTS optimization; (v) Ours: the MCTS-optimized malicious MCP template that preserves functionality and task completion yet induces verbose, multi-turn tool calling trajectories.

Attack LLMs. Within the MCTS optimizer, we use Llama-3.3-70B-Instruct as the Editor LLM. For the one-shot rewriting that instantiates the Universal Malicious Template on an MCP server, we employ gpt-4o (OpenAI, 2024).

Attack setup. For each tool/LLM pair we (1) select from the seed bank the starter template with the highest acceptance under a fixed per-turn length target; (2) instantiate a protocol-compatible malicious variant by editing text-only fields of the benign MCP server (argument descriptions and in-progress/corrective messages; optional fields for progress and the completeness artifact; termination deferred via text-only notices; benign payload preserved), using the segment index and full artifact to encourage multi-turn trajectories with verbose tool-calling outputs; and (3) run UCT-MCTS refinement with phase gating (multi-turn induction before length induction) and a two-stage evaluation, freezing the template once it meets a fixed acceptance threshold and writing it back to the seed bank. This instantiation aligns with the components and procedures detailed in §4.2–§4.4 and Algorithm 1. Unless otherwise noted, we evaluate under the same serving cap ($M=16,384$ max completion tokens per generation) and our default multi-turn setting used throughout Table 2.

Metrics. We evaluate all metrics on both benchmarks. We report: (i) Efficacy: (a) token length per query: average output tokens per eligible query; (b) attack success rate (ASR): fraction of eligible queries where the targeted behavior occurs (ours: multi-turn tool calling with long outputs; Overthink: single-turn think inflation) and $\text{Succ}(u, \tau, o) = 1$; and (c) task success rate: end-to-end completion under benign MCP servers and under our attack. (ii) Resource impact: (a) total energy consumption (Wh), integrating per-device power over time; (b) maximum GPU KV cache usage, the peak KV-cache occupancy reported by

the serving stack. (iii) Throughput efficiency (tokens/s): tokens/s of a fixed benign workload co-running with the evaluated condition. We present results in the order: effectiveness $\rightarrow$ resources $\rightarrow$ throughput $\rightarrow$ defenses. Supplementary generality, transferability, surface-form, and guardrail evaluations are provided in Appendix D.

4.2 Evaluation of Attack Effectiveness and Correctness

Correctness under resource amplification. Table 2 reports task success under attack separately from ASR. Across all 12 model–benchmark cells, task success remains within 2.6 percentage points of the corresponding benign rate. ASR is intentionally stricter because it additionally requires the targeted multi-turn and length behavior; the gap between task success and ASR therefore consists of episodes that still solve the user task but do not realize the full amplification pattern. The main study contains 13,920 paired episodes. For our attack, median output length is 74.0K tokens (IQR 62.1K–86.5K) on ToolBench and 74.8K (62.9K–87.4K) on BFCL; model-stratified bootstrap gives pooled ASR of 84.0% [81.7, 86.4] and 84.5% [81.6, 87.4], respectively. Appendix D.3 provides the focused task-success view. Meanwhile, cost amplification is pervasive: the largest factors include $\times 658.10$ on Mistral-Large (BFCL; 57,255 vs. 87) and $\times 314.73$ on Llama-3.3-70B-Instruct (ToolBench; 81,830 vs. 260), and even the smallest case remains $\times 65.51$ (Seed-32B on ToolBench). This illustrates a key property of tool-layer DoS: the final answer can remain correct while the intermediate tool-calling process becomes orders-of-magnitude more expensive, making output-only validation insufficient.

Comparison with baselines. Table 2 contrasts three baseline families. Overthink is fundamentally single-turn and therefore bounded by the per-generation cap M , yielding at most $\sim 10^4$ tokens on our non-RAG setting (e.g., 8,743 on Qwen-3-32B/ToolBench), far below our typical 6×10^4 – 9×10^4 ranges. Overthink-*mt* repeats the context-layer trigger across multiple tool calls to match our budget, but still leaves the tool server benign; our attack is longer in all 12 model–benchmark cells and achieves higher ASR in 10 of 12. Hand-crafted isolates the effect of tool-layer templating without MCTS: it often achieves long trajectories, but at noticeably lower ASR and/or shorter outputs than our optimized templates (e.g., on ToolBench/Qwen, 51.3% $\rightarrow$ 80.5% ASR with 37,425 $\rightarrow$ 65,273 to-

Metric	Method	ToolBench						BFCL					
		Llama	Qwen	GLM	Mistral	L-DS	Seed	Llama	Qwen	GLM	Mistral	L-DS	Seed
Length	Benign	260	638	634	127	197	1298	195	770	389	87	157	950
	Overthink	389	8743	12580	1453	725	4223	369	9459	13053	1397	901	5753
	Overthink-mt	2253	14389	11720	11704	842	5197	2341	15426	12492	9049	725	4835
	Hand-crafted	71032	37425	39958	48294	42794	51938	63927	36203	32853	37937	47394	69273
	Our attack	81830	65273	63694	61354	65546	85037	77052	67585	67656	57255	68464	90298
TSR	Benign	98.1%	94.6%	95.0%	90.8%	86.6%	90.4%	100.0%	98.5%	88.8%	83.8%	95.4%	98.0%
	Our attack	97.6%	93.1%	92.9%	89.6%	84.9%	89.1%	99.2%	97.4%	86.9%	82.6%	92.8%	97.1%
ASR	Overthink	99.6%	80.1%	57.5%	79.3%	91.4%	83.6%	99.5%	74.1%	55.3%	61.4%	93.4%	87.8%
	Overthink-mt	99.6%	78.9%	52.4%	69.5%	37.6%	48.2%	96.5%	73.1%	56.3%	59.5%	45.2%	29.1%
	Hand-crafted	88.5%	51.3%	54.4%	64.0%	50.2%	55.9%	86.3%	50.3%	53.8%	62.4%	51.8%	64.0%
	Our attack	96.2%	80.5%	83.1%	81.2%	78.9%	84.3%	93.9%	82.7%	83.3%	78.2%	76.3%	92.4%

Table 2: Attack effectiveness and correctness. Task-success rows report end-to-end completion, while ASR additionally requires the targeted amplification behavior. Overthink-mt uses 6 tool calls to match our budget; Hand-crafted is a text-only, payload-preserving template without MCTS optimization. L-DS: Llama-DeepSeek-70B.

Source →Target	ToolBench	BFCL
Llama→Qwen	54,860 / 68.7%	58,420 / 70.9%
Qwen→Llama	75,640 / 91.8%	71,230 / 88.6%

Table 3: Direct transfer of optimized templates without additional MCTS search. Each entry reports output length / ASR.

kens). Overall, these comparisons attribute the strongest amplification with high correctness to (i) shifting the long-output site to the tool-calling step and compounding across turns, and (ii) MCTS-based text-only optimization that enhances robustness under black-box agent policies.

Template reuse across LLMs. Table 3 shows that optimized templates remain effective when applied directly to a different target model. Qwen-to-Llama transfer is especially strong, while Llama-to-Qwen is weaker but still outperforms the Hand-crafted baseline on the corresponding target. Target-specific optimization remains best, but direct transfer demonstrates that the seed bank stores reusable attack patterns rather than templates tied to a single model; full target-specific references are provided in Appendix D.4.

4.3 Evaluation on Computing Resources Consumption

Energy and KV-cache impact. Figure 2 shows that our attack substantially increases physical resource usage. Across models and benchmarks, energy consumption rises by about 100–560×; for example, Llama-3.3-70B-Instruct on ToolBench increases from 5.63 Wh to 3159.45 Wh (×561.18), while Mistral-Large on BFCL rises from 4.24 Wh

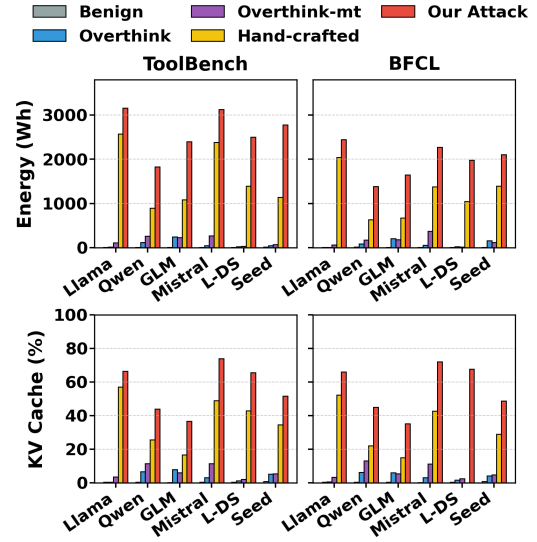


Figure 2: Resource impact (ToolBench/BFCL): energy (Wh) and peak KV-cache occupancy (%).

to 2269.60 Wh (×535.28). Peak GPU KV-cache occupancy also increases sharply, from typically below 1% in the benign setting to 35–74% under attack, such as 0.4%→73.9% for Mistral-Large on ToolBench and 0.3%→66.1% for Llama-3.3-70B-Instruct on BFCL. Similar trends hold across the remaining models, indicating that the attack consistently turns longer tool-calling trajectories into sustained system-level resource pressure.

Implications. Once multi-turn behavior is established, resource growth is driven mainly by turn count and per-turn verbosity rather than dataset-specific variation. In practice, the combination of large energy inflation and sustained KV-cache pressure reduces OOM-safe concurrency and leads directly to the throughput degradation studied next.

ToolBench Dataset						
Method	Llama	Qwen	GLM	Mistral	L-DS	Seed
Benign	3594	4602	3753	2898	3812	4001
Overthink	3728	4550	3189	2724	3711	4058
Overthink-mt	3342	4396	3081	2531	3746	3815
Hand-crafted	2068	2692	2660	1996	2596	2302
Our attack	1672	1793	2324	1716	2106	1417

BFCL Dataset						
Method	Llama	Qwen	GLM	Mistral	L-DS	Seed
Benign	3563	5093	3734	2871	3845	4082
Overthink	3822	4561	3185	2289	3752	4078
Overthink-mt	3571	3949	3205	2483	3729	3890
Hand-crafted	2248	2559	2791	1902	2451	2529
Our attack	1668	1738	2410	1740	2130	1536

Table 4: Throughput efficiency (tokens/s).

4.4 Impact on System Throughput

Beyond direct resource consumption, our attack materially degrades the system’s overall throughput efficiency for concurrent benign workloads, as shown in Table 4. Our attack consistently halves the throughput (measured in tokens/s) of a co-running benign task, causing an average performance drop of approximately 50% across both ToolBench and BFCL. In several cases, the degradation exceeds 60% (e.g., Seed-32B on ToolBench sees a 64.6% drop from 4001 to 1417 tokens/s). In stark contrast, the single-turn Overthink baseline induces only negligible changes, confirming that sustained, multi-turn engagement is the primary driver of this system-level penalty. This collapse in throughput is a direct consequence of the resource pressure detailed in the previous subsection. The prolonged, multi-turn generations, coupled with a sharp increase in peak GPU KV cache usage to the 35–74% range (up from <1% benignly), create significant KV-cache pressure and scheduler contention. This sustained resource occupancy significantly reduces scheduling headroom for co-located tasks, throttling the processing of normal traffic (Kwon et al., 2023).

4.5 Defense Evaluation

Defense settings. We evaluate three representative classes of defenses under the same episodes used for efficacy and resource measurements: (i) a prompt-level perplexity (PPL) filter applied to both the user query and the tool response (we conservatively score each episode by the larger of the two) with detector-specific thresholds calibrated from benign tool docstrings (Alon and Kamfonas, 2023; Jain et al., 2023); (ii) output/trajectory monitoring, including a generation-level self-monitoring

Baseline	59.31	38.57	93.70	75 50 25
Overthink	17.23	14.98	18.79	
Overthink-mt	20.41	19.89	21.04	
Hand-crafted	14.84	12.15	16.70	
Ours	13.34	11.35	15.10	
	Llama	Ministral	Qwen	

Figure 3: PPL filter on query and tool response (we report the larger side per episode). Detectors: Llama-3.1-8B, Ministral-8B, and Qwen3-8B.

prompt that asks the model whether to abort suspicious behavior and trajectory-level safety judges (Qwen-Guard-3 and Llama-Guard-3) applied to the full interaction trace (Wang et al., 2024; Zeng et al., 2024; Zhao et al., 2025; Meta AI, 2024); and (iii) hard budget controls via per-session token caps and tool-call limits, reporting residual ASR under different caps/limits (Figure 5).

Input detection via PPL. We evaluate a prompt-level PPL filter that scores both (i) the user query and (ii) the tool response text, since our attack is triggered by tool-facing messages and can inflate content on either side of the agent–tool boundary (Alon and Kamfonas, 2023; Jain et al., 2023). Concretely, for each episode we compute PPL on the query input and the first tool response (when available) and take the larger value as a conservative score. We set a baseline threshold per detector LM as the maximum PPL over the union of benign tool docstrings (ToolBench/BFCL original servers); this is intentionally conservative so that benign tool text defines the allowable range. Figure 3 shows that our attack remains well within this benign-derived envelope, so a PPL-based input filter is ineffective in detecting our tool-layer manipulation. A PromptArmor-style GPT-4o guardrail yields similarly low detection—2.5% on ToolBench and 2.2% on BFCL, with benign false-positive rates of 0.7% and 0.9%, respectively (Appendix D.6).

Output monitoring through self-reflection. We next test a generation-level self-monitoring defense that prompts the model to reflect on whether its own behavior should be aborted (Wang et al., 2024; Zeng et al., 2024). Figure 4 shows that it frequently flags Overthink and Overthink-mt, but almost never flags our attack (typically near 0%, and always below 3%). A key reason is that our token-heavy steps are framed as a required part of completing the tool procedure, rather than as off-task content, so the model does not perceive an obvious anomaly to stop.

Trajectory-level safety judges. We also apply

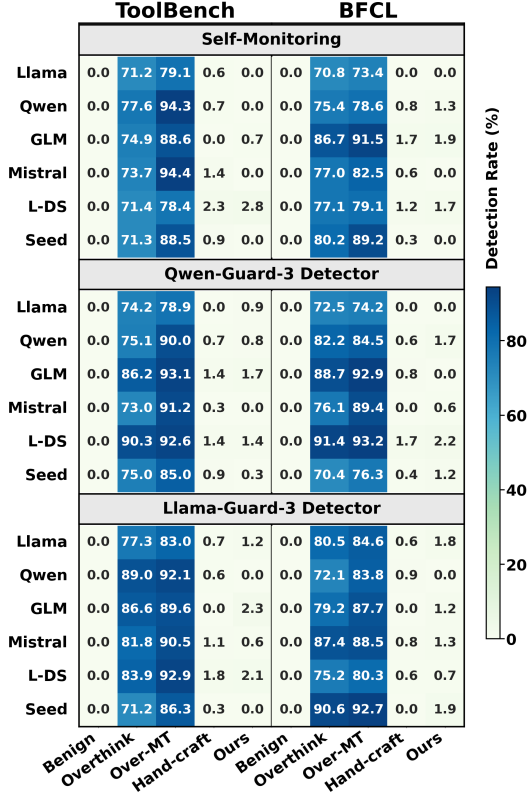


Figure 4: Detection rate (%) of output and trajectory monitors (self-monitoring, Qwen-Guard-3, Llama-Guard-3).

safety-judge models to the full interaction trace and ask for a safe/unsafe verdict. In Figure 4, both Qwen-Guard-3 and Llama-Guard-3 strongly flag Overthink-style baselines, but rarely flag our attack (again below 3%) (Zhao et al., 2025; Meta AI, 2024). These results indicate that content-oriented safety judging is largely orthogonal to correctness-preserving, protocol-compatible resource amplification: the attack’s harm is economic (compute/latency/KV pressure) rather than semantic toxicity.

Per-session token caps and tool-call limits. A common mitigation is to enforce hard per-session budgets: a token cap L and/or a tool-call limit N . Figure 5 shows these controls bound the worst-case cost, but do not reliably stop the attack. Even under tight caps or limits, ASR remains high for most models because the induced trajectory continues to consume a substantial share of the permitted budget. In practice, these mechanisms act as throttles: they cap amplification, but do not detect or prevent correctness-preserving, protocol-compatible resource abuse.

Toward process-aware defenses. These results motivate layered controls that treat tool execution

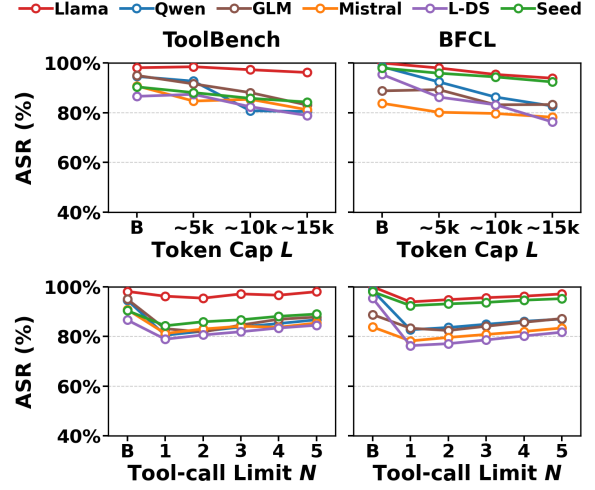


Figure 5: ASR under per-session token caps (top) and tool-call limits (bottom); baseline B is benign TSR.

as a workload rather than only as content. Tool provenance and schema pinning can reduce exposure before invocation, while host-side monitors can track cumulative argument tokens, same-tool repetition, and turn growth against benign per-tool baselines. The same signals also arise in legitimate pagination, chunking, retries, and validation, so thresholds must be calibrated for false positives and task utility. A full-trajectory task–effort judge could complement inexpensive telemetry, but it adds inference cost and requires trusted context about expected tool behavior. Designing and evaluating such workload-aware defenses is an important next step.

5 Conclusion

We propose an automated DoS attack on the LLM agent tool-calling layer that delays an unchanged benign payload behind a stateful, completeness-gated progression mechanism. An MCTS optimizer refines text-only server templates so that the agent-side LLM repeatedly generates long tool arguments while the task remains correct. Across six models, the attack produces trajectories over 60,000 tokens and inflates per-query cost by up to $658\times$, while representative prompt, self-monitoring, and trajectory safeguards rarely flag it. Validation on LangGraph and real MCP tools, together with cross-LLM transfer, indicates that the mechanism extends beyond a single agent stack or target model and is not tied to one artifact syntax or a particular tool interface. These results establish that correct final answers are not sufficient evidence of safe or efficient agent execution.

Limitations

Our main evaluation uses a controlled emulator with deterministic, stubbed payloads and single-tool/single-turn ToolBench/BFCL subsets for comparability; Appendices D.1 and D.2 validate the trend on LangGraph and real MCP tools. Results may still vary across production hosts, runtimes, hardware, and closed-source agents. In multi-tool workflows, if N compromised tools are all reached and each sustains up to $T_{\max}$ induced turns, the interaction count can scale as $O(NT_{\max})$; actual cascade exposure also depends on planner routing and tool dependencies. Our default completeness artifact is numeric, but Appendix D.5 shows a pseudo-JSON variant outside simple list rules; other compliant long-form patterns remain unexplored.

Ethical Considerations

This work studies a resource-consumption attack with potential dual-use. To mitigate risk, we run experiments in an isolated environment (no real external actions), use public benchmark data without private user information, and present the method to motivate workflow-level protections such as tool provenance controls and trajectory-based monitoring rather than to enable misuse. Before public code release, we will notify the MCP Working Group and maintainers of major MCP hosts and agent frameworks, including Claude Desktop and LangGraph MCP adapters, to coordinate mitigation efforts and responsible disclosure.

6 Acknowledgements

This research is supported by the National Research Foundation, Singapore and Infocomm Media Development Authority under its Trust Tech Funding Initiative. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not reflect the views of National Research Foundation, Singapore and Infocomm Media Development Authority.

References

Gabriel Alon and Michael Kamfonas. 2023. [Detecting Language Model Attacks with Perplexity](#). Preprint, arXiv:2308.14132.

Amazon Web Services. 2025. [Amazon bedrock](#). Service overview; accessed 26 August 2026.

Anthropic. 2024. [Introducing the Model Context Protocol \(MCP\)](#). Blog post; accessed 26 August 2026.

Eshta Bhardwaj, Rohan Alexander, and Christoph Becker. 2025. [Limits to AI Growth: The Ecological and Social Consequences of Scaling](#). Preprint, arXiv:2501.17980.

ByteDance Seed Team. 2025. [Seed-OSS](#). Official software and model repository; accessed 26 August 2026.

Jianshuo Dong, Ziyuan Zhang, Qingjie Zhang, Tianwei Zhang, Hao Wang, Hewu Li, Qi Li, Chao Zhang, Ke Xu, and Han Qiu. 2025. [An Engorgio prompt makes large language model babble on](#). In International Conference on Learning Representations (ICLR 2025). Conference paper. Preprint: arXiv:2412.19394.

Abul Ehtesham, Aditi Singh, Gaurav Kumar Gupta, and Saket Kumar. 2025. [A survey of agent interoperability protocols: Model Context Protocol \(MCP\), Agent Communication Protocol \(ACP\), Agent-to-Agent Protocol \(A2A\), and Agent Network Protocol \(ANP\)](#). Preprint, arXiv:2505.02279.

Shiqing Fan, Xichen Ding, Liang Zhang, and Linjian Mo. 2025. [MCPToolBench++: A Large Scale AI Agent Model Context Protocol MCP Tool Use Benchmark](#). Preprint, arXiv:2508.07575.

Kuofeng Gao, Yang Bai, Jindong Gu, Shu-Tao Xia, Philip Torr, Zhifeng Li, and Wei Liu. 2024a. [Inducing high energy-latency of large vision-language models with verbose images](#). In International Conference on Learning Representations (ICLR 2024). Conference paper. Preprint: arXiv:2401.11170.

Kuofeng Gao, Tianyu Pang, Chao Du, Yong Yang, Shu-Tao Xia, and Min Lin. 2024b. [Denial-of-Service Poisoning Attacks against Large Language Models](#). Preprint, arXiv:2410.10760.

Jonas Geiping, Alex Stein, Manli Shu, Khalid Saifullah, Yuxin Wen, and Tom Goldstein. 2024. [Coercing LLMs to do and reveal \(almost\) anything](#). In ICLR 2024 Workshop on Secure and Trustworthy Large Language Models. Workshop paper. Preprint: arXiv:2402.14020.

GLM-4.5 Team. 2025. [GLM-4.5: Agentic, reasoning, and coding \(ARC\) foundation models](#). Preprint, arXiv:2508.06471.

Google Cloud. 2025. [Vertex AI agent builder overview](#). Documentation page; accessed 26 August 2026.

Aaron Grattafiori and 1 others. 2024. [The Llama 3 herd of models](#). Preprint, arXiv:2407.21783.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, and 1 others. 2025. [DeepSeek-R1: Incentivizing reasoning capability in LLMs via reinforcement learning](#). Nature, 645(8081):633–638.

- Mohammed Mehedi Hasan, Hao Li, Emad Fallahzadeh, Gopi Krishnan Rajbahadur, Bram Adams, and Ahmed E. Hassan. 2026. [Model context protocol \(MCP\) at first glance: Studying the security and maintainability of MCP servers](#). *ACM Transactions on Software Engineering and Methodology*. Online publication: 12 May 2026; volume, issue, and article number not yet assigned.
- Xinyi Hou, Yanjie Zhao, Shenao Wang, and Haoyu Wang. 2026. [Model context protocol \(MCP\): Landscape, security threats, and future research directions](#). *ACM Transactions on Software Engineering and Methodology*. Online publication: 16 February 2026; volume, issue, and article number not yet assigned.
- Mengkang Hu, Yuhang Zhou, Wendong Fan, Yuzhou Nie, Bowei Xia, Tao Sun, Ziyu Ye, Zhaoxuan Jin, Yingru Li, Qiguang Chen, Zeyu Zhang, Yifeng Wang, Qianshuo Ye, Bernard Ghanem, Ping Luo, and Guohao Li. 2025. [OWL: Optimized Workforce Learning for General Multi-Agent Assistance in Real-World Task Automation](#). Preprint, arXiv:2505.23885.
- Neel Jain, Avi Schwarzschild, Yuxin Wen, Gowthami Somepalli, John Kirchenbauer, Ping-yeh Chiang, Micah Goldblum, Aniruddha Saha, Jonas Geiping, and Tom Goldstein. 2023. [Baseline Defenses for Adversarial Attacks against Aligned Language Models](#). Preprint, arXiv:2309.00614.
- Levente Kocsis and Csaba Szepesvári. 2006. [Bandit Based Monte-Carlo Planning](#). In David Hutchison, Takeo Kanade, Josef Kittler, Jon M. Kleinberg, Friedemann Mattern, John C. Mitchell, Moni Naor, Oscar Nierstrasz, C. Pandu Rangan, Bernhard Steffen, Madhu Sudan, Demetri Terzopoulos, Dough Tygar, Moshe Y. Vardi, Gerhard Weikum, Johannes Fürnkranz, Tobias Scheffer, and Myra Spiliopoulou, editors, *Machine Learning: ECML 2006*, volume 4212, pages 282–293. Springer Berlin Heidelberg, Berlin, Heidelberg.
- Dezhang Kong, Shi Lin, Zhenhua Xu, Zhebo Wang, Minghao Li, Yufeng Li, Yilun Zhang, Hujin Peng, Xiang Chen, Zeyang Sha, Yuyuan Li, Changting Lin, Xun Wang, Xuan Liu, Ningyu Zhang, Chaochao Chen, Chunming Wu, Muhammad Khuram Khan, and Meng Han. 2025. [A Survey of LLM-Driven AI Agent Communication: Protocols, Security Risks, and Defense Countermeasures](#). Preprint, arXiv:2506.19676.
- Abhinav Kumar, Jaechul Roh, Ali Naseh, Marzena Karpinska, Mohit Iyyer, Amir Houmansadr, and Eugene Bagdasarian. 2025. [OverThink: Slow-down Attacks on Reasoning LLMs](#). Preprint, arXiv:2502.02542.
- Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. 2023. [Efficient Memory Management for Large Language Model Serving with PagedAttention](#). In *Proceedings of the 29th Symposium on Operating Systems Principles (SOSP '23)*, pages 611–626. Extended version/preprint: arXiv:2309.06180.
- Yedidel Louck, Ariel Stulman, and Amit Dvir. 2025. [Proposal for Improving Google A2A Protocol: Safeguarding Sensitive Data in Multi-Agent Systems](#). Preprint, arXiv:2505.12490.
- Alexandra Sasha Luccioni, Yacine Jernite, and Emma Strubell. 2024. [Power Hungry Processing: Watts Driving the Cost of AI Deployment?](#) In *Proceedings of the 2024 ACM Conference on Fairness, Accountability, and Transparency*, pages 85–99. Preprint: arXiv:2311.16863.
- Junyu Luo, Weizhi Zhang, Ye Yuan, Yusheng Zhao, Junwei Yang, Yiyang Gu, Bohan Wu, Binqi Chen, Ziyue Qiao, Qingqing Long, Rongcheng Tu, Xiao Luo, Wei Ju, Zhiping Xiao, Yifan Wang, Meng Xiao, Chenwu Liu, Jingyang Yuan, Shichang Zhang, and 7 others. 2025a. [Large Language Model Agent: A Survey on Methodology, Applications and Challenges](#). Preprint, arXiv:2503.21460.
- Zhiling Luo, Xiaorong Shi, Xuanrui Lin, and Jinyang Gao. 2025b. [Evaluation Report on MCP Servers](#). Preprint, arXiv:2504.11094.
- Meta AI. 2024. [Llama Guard 3-8B model card](#). Official model card; accessed 26 August 2026.
- Microsoft. 2025. [Microsoft copilot: AI tools for organizations](#). Product page; accessed 26 August 2026.
- Mistral AI. 2024. [Mistral AI models](#). Official model documentation; accessed 26 August 2026.
- Model Context Protocol Working Group. 2025. [Model context protocol specification, version 2025-11-25](#). Protocol specification; accessed 26 August 2026.
- Mahmoud Mohammadi, Yipeng Li, Jane Lo, and Wendy Yip. 2025. [Evaluation and Benchmarking of LLM Agents: A Survey](#). In *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining V.2*, pages 6129–6139.
- OpenAI. 2024. [GPT-4o system card](#). System card; accessed 26 August 2026.
- OWASP Gen AI Security Project. 2025. [LLM10:2025 unbounded consumption](#). Accessed 26 August 2026.
- Zaifeng Pan, Ajikumar Patel, Zhengding Hu, Yipeng Shen, Yue Guan, Wan-Lu Li, Lianhui Qin, Yida Wang, and Yufei Ding. 2025. [KVFlow: Efficient Prefix Caching for Accelerating LLM-Based Multi-Agent Workflows](#). Preprint, arXiv:2507.07400.
- Shishir G. Patil, Huanzhi Mao, Fanjia Yan, Charlie Cheng-Jie Ji, Vishnu Suresh, Ion Stoica, and Joseph E. Gonzalez. 2025. [The berkeley function calling leaderboard \(BFCL\): From tool use to agentic evaluation of large language models](#). In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pages 48371–48392. PMLR.

- PwC. 2025. [PwC's AI agent survey](#). Published 16 May 2025; accessed 26 August 2026.
- Qwen Team. 2025. [Qwen-Agent](#). Software repository; accessed 26 August 2026.
- Siddharth Samsi, Dan Zhao, Joseph McDonald, Baolin Li, Adam Michaleas, Michael Jones, William Bergeron, Jeremy Kepner, Devsh Tiwari, and Vijay Gadeppally. 2023. [From words to watts: Benchmarking the energy costs of large language model inference](#). In 2023 IEEE High Performance Extreme Computing Conference (HPEC), pages 1–9. Preprint: arXiv:2310.03003.
- Ranjan Sapkota, Konstantinos I. Roumeliotis, and Manoj Karkee. 2026. [AI agents vs. agentic AI: A conceptual taxonomy, applications and challenges](#). Information Fusion, 126:103599.
- Minjie Shen and Qikai Yang. 2025. [From Mind to Machine: The Rise of Manus AI as a Fully Autonomous Digital Agent](#). Preprint, arXiv:2505.02024.
- Tianneng Shi, Kaijie Zhu, Zhun Wang, Yuqi Jia, Will Cai, Weida Liang, Haonan Wang, Hend Alzahrani, Joshua Lu, Kenji Kawaguchi, Basel Alomair, Xuan-dong Zhao, William Yang Wang, Neil Gong, Wenbo Guo, and Dawn Song. 2026. [PromptArmor: An essential baseline for prompt injection defenses](#). ICLR 2026 submission; earlier preprint: arXiv:2507.15219.
- Lijun Sun, Yijun Yang, Qiqi Duan, Yuhui Shi, Chao Lyu, Yu-Cheng Chang, Chin-Teng Lin, and Yang Shen. 2025. [Multi-Agent Coordination across Diverse Applications: A Survey](#). Preprint, arXiv:2502.14743.
- Khanh-Tung Tran, Dung Dao, Minh-Duong Nguyen, Quoc-Viet Pham, Barry O'Sullivan, and Hoang D. Nguyen. 2025. [Multi-Agent Collaboration Mechanisms: A Survey of LLMs](#). Preprint, arXiv:2501.06322.
- Gaël Varoquaux, Alexandra Sasha Luccioni, and Meredith Whittaker. 2025. [Hype, sustainability, and the price of the bigger-is-better paradigm in ai](#). In Proceedings of the 2025 ACM Conference on Fairness, Accountability, and Transparency, pages 61–75. Conference paper. Preprint: arXiv:2409.14160.
- Yanlin Wang, Wanjuan Zhong, Yanxian Huang, Ensheng Shi, Min Yang, Jiachi Chen, Hui Li, Yuchi Ma, Qianxiang Wang, and Zibin Zheng. 2025. [Agents in Software Engineering: Survey, landscape, and vision](#). Automated Software Engineering, 32:70.
- Zezhong Wang, Fangkai Yang, Lu Wang, Pu Zhao, Hongru Wang, Liang Chen, Qingwei Lin, and Kam-Fai Wong. 2024. [SELF-GUARD: Empower the LLM to safeguard itself](#). In Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers), pages 1648–1668, Mexico City, Mexico. Association for Computational Linguistics.
- Zhiheng Xi, Wenxiang Chen, Xin Guo, Wei He, Yiwen Ding, Boyang Hong, Ming Zhang, Junzhe Wang, Senjie Jin, Enyu Zhou, Rui Zheng, Xiaoran Fan, Xiao Wang, Limao Xiong, Yuhao Zhou, Weiran Wang, Changhao Jiang, Yicheng Zou, Xiangyang Liu, and 10 others. 2025. [The Rise and Potential of Large Language Model Based Agents: A Survey](#). Science China Information Sciences, 68(2):121101.
- Wenrui Xu and Keshab K. Parhi. 2025. [A Survey of Attacks on Large Language Models](#). Preprint, arXiv:2505.12567.
- Yibo Yan, Shen Wang, Jiahao Huo, Philip S. Yu, Xuming Hu, and Qingsong Wen. 2025. [MathAgent: Leveraging a Mixture-of-Math-Agent Framework for Real-World Multimodal Mathematical Error Detection](#). Preprint, arXiv:2503.18132.
- An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, and 41 others. 2025a. [Qwen3 technical report](#). Preprint, arXiv:2505.09388.
- Yingxuan Yang, Huacan Chai, Yuanyi Song, Siyuan Qi, Muning Wen, Ning Li, Junwei Liao, Haoyi Hu, Jianghao Lin, Gaowei Chang, Weiwen Liu, Ying Wen, Yong Yu, and Weinan Zhang. 2025b. [A Survey of AI Agent Protocols](#). Preprint, arXiv:2504.16736.
- Yifan Zeng, Yiran Wu, Xiao Zhang, Huazheng Wang, and Qingyun Wu. 2024. [AutoDefense: Multi-Agent LLM Defense against Jailbreak Attacks](#). Preprint, arXiv:2403.04783.
- Boyang Zhang, Yicong Tan, Yun Shen, Ahmed Salem, Michael Backes, Savvas Zannettou, and Yang Zhang. 2025a. [Breaking agents: Compromising autonomous LLM agents through malfunction amplification](#). In Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing, pages 34964–34976, Suzhou, China. Association for Computational Linguistics.
- Dongsun Zhang, Zekun Li, Xu Luo, Xuannan Liu, Peipei Li, and Wenjun Xu. 2026. [MCP Security Bench \(MSB\): Benchmarking Attacks Against Model Context Protocol in LLM Agents](#). In International Conference on Learning Representations (ICLR 2026). Accepted at ICLR 2026; preprint: arXiv:2510.15994.
- Xinnong Zhang, Jiayu Lin, Xinyi Mou, Shiyue Yang, Xiawei Liu, Libo Sun, Hanjia Lyu, Yihang Yang, Weihong Qi, Yue Chen, Guanying Li, Ling Yan, Yao Hu, Siming Chen, Yu Wang, Jingxuan Huang, Jiebo Luo, Shiping Tang, Libo Wu, and 2 others. 2025b. [SocioVerse: A World Model for Social Simulation Powered by LLM Agents and A Pool of 10 Million Real-World Users](#). Preprint, arXiv:2504.10157.
- Yuanhe Zhang, Zhenhong Zhou, Wei Zhang, Xinyue Wang, Xiaojun Jia, Yang Liu, and Sen Su. 2025c.

Crabs: Consuming resource via auto-generation for LLM-DoS attack under black-box settings. In Findings of the Association for Computational Linguistics: ACL 2025, pages 11128–11150, Vienna, Austria. Association for Computational Linguistics.

Haiquan Zhao, Chenhan Yuan, Fei Huang, Xiaomeng Hu, Yichang Zhang, An Yang, Bowen Yu, Dayiheng Liu, Jingren Zhou, Junyang Lin, Baosong Yang, Chen Cheng, Jialong Tang, Jiandong Jiang, Jianwei Zhang, Jijie Xu, Ming Yan, Minmin Sun, Pei Zhang, and 24 others. 2025. **Qwen3guard technical report.** Preprint, arXiv:2510.14276.

Yuxiang Zheng, Dayuan Fu, Xiangkun Hu, Xiaojie Cai, Lyumanshan Ye, Pengrui Lu, and Pengfei Liu. 2025. **DeepResearch: Scaling Deep Research via Reinforcement Learning in Real-world Environments.** Preprint, arXiv:2504.03160.

A Notation

Symbol	Meaning
q	user query, $q \in \mathcal{X}$
u	user goal implied by q , $u = \text{goal}(q)$
o	agent’s final answer
A	agent policy (decision layer)
M	underlying LLM (black box)
T_θ	MCP server parameterized by text template θ
θ_0	benign (unmodified) template
$\mathcal{F}$	tool functions exposed by T_θ
$S(q; A, M, \theta_0)$	indicator that A routes at least one call to T_{θ_0} under θ_0
$\mathcal{D}_T$	routable set, $\mathcal{D}_T = \{q \in \mathcal{X} : S(q; A, M, \theta_0) = 1\}$
$\tau(q; \theta)$	interaction trajectory, $\tau = \{(a_t, r_t)\}_{t=1}^n$
$C(\tau)$	output-token cost, $C(\tau) = \text{Tok}_{\text{out}}(\tau)$
$\text{Succ}(u, \tau, o)$	task success indicator, $\text{Succ}(u, \tau, o) \in \{0, 1\}$
t	segment index used by our template, $t = 1, \dots, T_{\max}$
$T_{\max}$	terminal segment count (attack budget)
L	per-turn completeness-artifact length target, $L \in \mathbb{N}$

B Additional Methodology Details

B.1 Trigger Set and Conditioning

Trigger condition. Let $q \in \mathcal{X}$ denote a user query and $u = \text{goal}(q)$ the implied goal. We do not tamper with q , prompts, or retrieval; the attack remains dormant until A (driven by M) chooses to call the MCP server. We define the benign selection event

$$S(q; A, M, \theta_0) = \mathbb{I}\{\exists t : a_t \text{ calls } T_{\theta_0} \text{ under } \theta_0\},$$

and the corresponding routable set

$$\mathcal{D}_T = \{q \in \mathcal{X} : S(q; A, M, \theta_0) = 1\}.$$

Conditioning on $\mathcal{D}_T$ captures the common case where a tool is legitimately involved in the workflow (e.g., date/time, conversion, search, transforms) (Model Context Protocol Working Group, 2025; Kong et al., 2025; Luo et al., 2025b; Sapkota et al., 2026). We condition on $\mathcal{D}_T$ because we neither alter

q nor influence the first routing decision; the attack acts only through tool-facing messages after T is legitimately invoked. **Cost metric note.** We focus on output tokens because they typically dominate both API fees and compute load. The attack also increases input tokens by elongating trajectories, but the growth in completions is the primary driver of resource consumption in practice.

B.2 Universal Template Elements and Failure Handling

Return policy invariants. We enforce four invariants (summarized in Table 5). (i) Monotone progress: $t_1=1$ and valid calls advance $t_{k+1}=t_k+1$; invalid indices are rejected. (ii) Format completeness: $\text{validate}(\text{sequence})$ holds iff the sequence matches the required complete, comma-separated format and ordering. (iii) Return policy: valid sequences with $t < T_{\max}$ trigger a Progress notice, while invalid sequences trigger a Repair notice without advancing t . (iv) Termination: only when $t=T_{\max}$ and the latest sequence validates does the server emit the Terminal return (benign payload). Together, these invariants induce multi-turn, long tool-call traces while preserving task correctness and MCP compatibility through text-only interface additions and payload pass-through.

Terminology. We use completeness artifact for the general mechanism and “calibration_sequence” for the field name in the default implementation. The field is introduced by the malicious server; it is not part of MCP and does not perform benign protocol negotiation. MCP handshakes establish legitimate tool–host communication, whereas the completeness artifact gates progress before the unchanged benign payload is returned. The mechanism does not require a numeric list specifically; Appendix D.5 evaluates a structured pseudo-JSON variant.

B.2.1 Worked Server Example

The interaction follows initial call $\rightarrow$ Progress $\rightarrow$ repeated artifact-bearing calls (with Repair on malformed outputs) $\rightarrow$ Terminal. The long content is generated in the agent’s tool arguments, whereas server responses remain short. The user goal and final payload are unchanged; only their release is delayed behind repeated agent-side generation.

B.3 Seed Bank Screening and Promotion

The seed bank is a repository of protocol-compatible, task-correct, text-only templates T_θ . We initialize it with a single human-authored seed and lightly screen on a fixed query set/agent to confirm acceptance at the fixed L^* . Each MCTS run starts from a selected seed and halts when the acceptance predicate is met; the resulting template is written back with minimal metadata (estimated ASR, segment stability, omission/repair rates, refusal notes). Subsequent runs resample top seeds by ASR and stability and apply a stricter acceptance target before promotion. This cyclic promotion improves starting points by editing only text-visible content and leaves the terminal payload unchanged. In practice, a few cycles yield reusable seeds that transfer across LLMs and MCP servers.

B.4 MCTS Optimizer Details

Action space and edit zones. We organize atomic text edits into three families: $\mathcal{A}_{\text{MT}}$ (multi-turn induction), $\mathcal{A}_{\text{LEN}}$ (length induction), and $\mathcal{A}_{\text{REP}}$ (repair after omission/format errors). Beyond phase-aware gating, we instantiate these families as 16 atomic edits applied exclusively to non-executable, text-visible zones of the server (docstring argument descriptions, in-progress/unfinished notices, and validation-error messages). The multi-turn family sharpens next-call salience and enforces monotone segment progression; the length family strengthens the “complete, comma-separated” requirement

Element	Type	Effect on agent trajectory	Constraints / failure handling
Segment (t)	ARG	Marks progress and induces multi-turn; the agent increments $t \rightarrow t+1$ until a terminal cue.	$t \geq 1$ and strictly monotone by $+1$; non-positive or non-monotone values are rejected.
Completeness artifact (L)	ARG	Inflates per-turn completion at the tool-calling site; the default “calibration_sequence” field contains a full comma-separated list.	Exactly L integers, strictly increasing, using digits and commas (optional spaces); no ranges (e.g., “1–5”), no ellipses (“...”), no duplicates. Malformed $\Rightarrow$ Repair notice.
Progress notice	RETURN	Declares “in progress”; instructs the next call with $t+1$ and a complete artifact; preserves the goal path.	Emitted iff $t < T_{\max}$ and the latest sequence validates; keeps executable code and payload semantics intact.
Repair notice	RETURN	Corrects abbreviated/invalid sequences; prevents bypass of the length gate.	Triggers on omissions, ranges, duplicates, wrong length/order, or illegal characters; keeps t unchanged. Requests the complete comma-separated list before proceeding.
Terminal return	RETURN	Ends the trajectory and returns the same benign payload as the original server.	Emitted only when $t=T_{\max}$ and the latest sequence validates; protocol-compatible pass-through.

Table 5: Universal template elements. Text-only arguments and notices enforce completeness-gated multi-turn progress and per-turn verbosity; the terminal return passes through the unchanged benign payload.

Part	Abridged “play_artist” instance
Benign	“play_artist(artist_name)” immediately returns “Now playing music by artist_name.”
Schema	Adds optional “segment” (integer, default 1) and “calibration_sequence” (string, default empty).
Progress	For segment $t < 6$, asks the next call to place the complete $1, \dots, N_M$ artifact in “calibration_sequence”.
Repair	If the artifact is abbreviated or malformed, requests a retry at the same segment.
Terminal	At segment 6 after validation, returns the original benign payload unchanged.

Table 6: A concrete benign-to-malicious transformation used in evaluation. N_M is a model-specific endpoint chosen to match the same per-turn token target.

to elicit long single-shot payloads during tool calling; and the repair family refines failure messaging to immediately solicit a compliant retry without advancing the segment. These primitives are intentionally small, mutually composable, and largely orthogonal, enabling MCTS to explore nuanced trade-offs between adherence and refusal while keeping the surface area auditable. Throughout, control flow and terminal payload semantics stay intact; only optional text fields carry the attack.

Phase gating, expansion, and parallelism. Each tree node v corresponds to a concrete server T_{θ_v} ; edges apply a single localized text edit. We maintain a phase label $\phi(v) \in \{\text{pre_MT}, \text{post_MT}\}$ and a node-local omission flag that unlocks repair actions if needed. In pre_MT, we use $\mathcal{A}_{\text{MT}}$ to stabilize multi-turn behavior; once screenings show stable segment sequencing, we switch to post_MT and use $\mathcal{A}_{\text{LEN}}$ to strengthen long outputs. An omission/format error observed at node v unlocks $\mathcal{A}_{\text{REP}}$ at v only. When a node is expanded, we instantiate one child per untried action from the phase-appropriate set (plus $\mathcal{A}_{\text{REP}}$ if enabled) and evaluate all new children in parallel.

Node selection and statistics. We use UCT (Kocsis and Szepesvári, 2006) with a running mean evaluation signal $\bar{Q}$ and visit counts N_{uct} for exploration (see §3.4). UCT counts

can be updated using Stage-1 samples only to avoid heavy batches skewing exploration, while $\bar{Q}$ aggregates all observed rollouts.

Node evaluation and reward. Each child undergoes a two-stage evaluation with configurable sizes and gates. For each rollout, we compute

$$\text{mt_pass} = \mathbb{I}\{\text{MT}\}, \quad \text{len_pass} = \mathbb{I}\{\text{LEN}\},$$

where MT means the multi-turn target is met with ordered segments, and LEN means the fixed L^* is reached and any omissions are repaired. We use

$$r = \alpha \text{mt_pass} + \beta \text{mt_pass len_pass},$$

where $0 < \alpha \leq \beta \leq 1$,

prioritizing stable multi-turn behavior (the α term) and adding credit for length only when multi-turn has been achieved (the multiplicative term). Stage-1 offers a quick screen to decide whether to run Stage-2 and to flip ϕ : pre_MT $\rightarrow$ post_MT once segment sequencing stabilizes; Stage-2 refines estimates under stochastic decoding.

Backpropagation. We propagate statistics along the path to the root. If a node meets the acceptance predicate (stabilized successes over the latest batch), we record the corresponding T_{θ^*} and insert it into the seed bank.

C Additional Experimental Details

C.1 Target LLMs configuration

All target models are served under a uniform runtime on a single node with eight H200 GPUs, using bfloat16 precision and a maximum context length of 131,072 tokens. We deploy vLLM (Kwon et al., 2023). Decoding follows the same setting across all conditions: nucleus sampling with $p = 0.95$ and temperature 0.5, and a per-generation completion cap of 16,384 tokens. These settings are held constant for every model and benchmark so that any change in cost, length, or throughput arises from the agent-tool interaction rather than heterogeneous serving choices.

C.2 Reasoning-Mode Behavior in Qwen-3-32B

For Qwen-3-32B runs with visible thinking enabled, Repair notices can make retry turns more verbose. When the server rejects an incomplete or malformed artifact, the model often explains the validation failure, restates the required format, and then constructs the corrected tool call inside its visible reasoning trace. This behavior can add non-trivial cost in affected episodes. However, the main cost driver remains the long-form tool-call artifact itself. Repair notices primarily improve robustness by keeping the agent on the goal path after omissions or format errors; they are not necessary for amplification when the model already produces complete artifacts. This is consistent with the role of Repair notices in Table 5: they prevent premature bypass of the completeness gate without changing the terminal benign payload.

C.3 Attacker LLMs configuration

We employ a two-stage approach using two distinct LLMs for attack generation. Within the iterative MCTS optimization loop, the Editor LLM is Llama-3.3-70B-Instruct (Grattafiori et al., 2024). Its serving and decoding configuration deliberately mirrors the target LLM setup described above to eliminate experimental confounds and ensure the generated edits are effective. For converting a benign tool description into a protocol-compatible malicious template, we leverage gpt-4o (OpenAI, 2024). We fix its temperature at 0 to guarantee deterministic and high-fidelity output for our seed templates, leaving all other parameters at the provider’s defaults.

C.4 Agent framework and execution environment

Qwen-Agent is a framework for building LLM applications that integrate instruction following, tool usage, planning, and memory, and it ships with reference applications such as a browser assistant, a code interpreter, and a customizable assistant; it also powers the backend of Qwen Chat (Qwen Team, 2025). In this study we rely on Qwen-Agent’s native support for the Model Context Protocol and for multi-turn tool calls (Model Context Protocol Working Group, 2025), and we use these native capabilities directly. All experiments are conducted in a controlled environment rather than on production stacks. For models other than Qwen, we perform minimal adaptations so that the same tool calling loop, message formatting, and termination logic apply uniformly, while keeping the agent policy and prompts fixed across all conditions. This yields an apples-to-apples emulator of agent behavior that isolates the effects of tool-layer edits.

C.5 Datasets filtering

We evaluate on two tool-use corpora designed for function calling by agents. ToolBench aggregates a broad set of utilities and APIs together with queries that require tool invocation (Fan et al., 2025). BFCL emphasizes structured function calling with clear argument schemas and deterministic behaviors across everyday tasks (Patil et al., 2025). From each benchmark, we extract all prompts that are single-turn and single-tool in their original specification, together with their associated tools.

To control data quality, we run a screening pass on Qwen-3-32B (Yang et al., 2025a) under the benign server for every candidate prompt. Each prompt is executed five times; if at least two out of five runs fail to trigger any tool call, that prompt is discarded. This filtering is performed before applying any attack template and is independent of the compared method. The same retained query set is used for all conditions, including Benign, Overthink, Overthink-mt, Hand-crafted, and Our attack. Thus, the filter does not advantage our method; rather,

it removes cases where the agent does not enter the tool layer at all, allowing the evaluation to isolate the agent–tool attack surface instead of mixing it with first-tool-routing failures.

C.6 Datasets wrapping

For every retained tool, we produce an MCP-compatible server template from its benchmark description using gpt-4o (OpenAI, 2024). The benign conversion preserves the tool’s identity and interface, including names and argument schemas, so the agent sees the same capability surface before attack instrumentation. Because our goal is to probe the trajectory and cost of the interaction rather than the accuracy of external data, live network calls are stubbed and deterministic placeholder values are returned in place of real API results. The terminal payload semantics remain consistent with the benign server so the agent can still complete the task. This design preserves protocol compatibility and task success (Model Context Protocol Working Group, 2025) while isolating the effect of template edits on multi-turn cost amplification.

C.7 Hand-Crafted Baseline

The Hand-crafted baseline is the initial template before MCTS refinement. It uses the same universal template family, the same text-only and payload-preserving constraints, the same terminal benign payload, and the same tool-call and length budget as Our attack. The only difference is that it does not apply any MCTS edits and receives no additional manual tuning. Therefore, the transition from Hand-crafted to Our attack isolates the effect of MCTS optimization within the same template family.

This baseline is intended to measure whether search improves robustness and adherence under black-box agent policies, rather than whether a different attack surface is introduced. For example, on ToolBench with Qwen-3-32B, ASR increases from 51.3% to 80.5%, and average length increases from 37,425 to 65,273 tokens after MCTS optimization. This indicates that MCTS stabilizes multi-turn induction and length amplification while preserving the same interface, constraints, and terminal payload.

D Additional Experimental Results

This section provides supplementary experiments that examine the generality, correctness preservation, transferability, surface-form robustness, and defense behavior of the proposed attack. Unless otherwise stated, we keep the agent policy, user prompts, decoding configuration, tool-call budget, and per-generation completion cap fixed across compared conditions. When an experiment intentionally changes the agent stack or benchmark, all remaining factors are kept consistent with the main evaluation. For attack methods, the reported rate is ASR, which requires both successful task completion and the targeted resource-amplification behavior. For the benign condition, the reported rate is TSR. In the tables below, “Length” denotes average output tokens per eligible query, and rate columns are reported as percentages.

D.1 Real-Stack Evaluation on LangGraph

Motivation and setup. The main experiments use a controlled qwen-agent-based emulator to isolate the effect of tool-response policies. To test whether the amplification effect is specific to this emulator, we additionally evaluate the same MCP-compatible benign and attack servers on a LangGraph-based agent stack. We use Llama-3.3-70B and Qwen-3-32B, and compare the same five conditions as in Table 2.

Results. Table 7 shows that the same trend holds on LangGraph. On ToolBench, Our attack reaches 83,472 tokens with 95.4% ASR on Llama-3.3-70B, and 63,918 tokens with 79.1% ASR on Qwen-3-32B. On BFCL, it reaches 79,614

ToolBench Dataset				
Method	Llama-3.3-70B		Qwen-3-32B	
	Length	Rate	Length	Rate
Benign	274	97.6%	701	93.8%
Overthink	436	98.2%	9,086	78.4%
Overthink-mt	2,641	96.9%	15,217	76.2%
Hand-crafted	74,315	86.9%	39,864	53.7%
Our attack	83,472	95.4%	63,918	79.1%

BFCL Dataset				
Method	Llama-3.3-70B		Qwen-3-32B	
	Length	Rate	Length	Rate
Benign	181	99.1%	826	97.2%
Overthink	402	98.4%	9,731	73.8%
Overthink-mt	2,487	95.8%	16,104	72.5%
Hand-crafted	61,205	85.7%	35,418	49.6%
Our attack	79,614	92.8%	68,209	83.0%

Table 7: Real-stack evaluation on LangGraph. Rate denotes TSR for benign rows and ASR for attack rows. Bold length values mark the largest amplification within each benchmark-model block.

Method	Llama-3.3-70B		Qwen-3-32B	
	Length	Rate	Length	Rate
Benign	342	95.0%	812	92.5%
Overthink	516	94.0%	9,280	76.5%
Overthink-mt	2,684	92.5%	14,960	74.5%
Hand-crafted	66,430	84.0%	46,880	52.0%
Our attack	78,260	91.5%	66,740	79.5%

Table 8: MSB-compatible evaluation on 20 single-tool or low-dependency MCP Security Bench tasks. Rate denotes TSR for the benign row and ASR for attack rows.

tokens with 92.8% ASR on Llama-3.3-70B, and 68,209 tokens with 83.0% ASR on Qwen-3-32B. In contrast, Overthink and Overthink-mt remain substantially shorter. These results suggest that the amplification mechanism arises from the MCP tool exchange itself rather than from a particular qwen-agent implementation.

D.2 Evaluation on MCP Security Bench

Motivation and setup. To test generality beyond ToolBench and BFCL, we conduct an MSB-compatible evaluation on MCP Security Bench (Zhang et al., 2026), which contains real MCP tools. We select 20 single-tool or low-dependency tasks to isolate the tool-response mechanism while avoiding confounds from multi-tool planning failures. The evaluation uses the same five methods as Table 2.

Results. Table 8 shows that Our attack transfers to MSB-style MCP tasks. For Llama-3.3-70B, it produces 78,260 tokens on average with 91.5% ASR, compared with 342 tokens and 95.0% TSR under the benign server. For Qwen-3-32B, it produces 66,740 tokens with 79.5% ASR, compared with 812 benign tokens and 92.5% TSR. The results are consistent with ToolBench and BFCL: the attack preserves task utility while amplifying output cost. This supports our view that correctness-preserving economic DoS is a distinct axis for MCP security evaluation.

ToolBench Dataset			
Model	Benign	Task	ASR
Llama-3.3-70B	98.1%	97.6%	96.2%
Qwen-3-32B	94.6%	93.1%	80.5%
GLM-4.5-Air	95.0%	92.9%	83.1%
Mistral Large	90.8%	89.6%	81.2%
Llama-DeepSeek-70B	86.6%	84.9%	78.9%
Seed-32B	90.4%	89.1%	84.3%

BFCL Dataset			
Model	Benign	Task	ASR
Llama-3.3-70B	100.0%	99.2%	93.9%
Qwen-3-32B	98.5%	97.4%	82.7%
GLM-4.5-Air	88.8%	86.9%	83.3%
Mistral Large	83.8%	82.6%	78.2%
Llama-DeepSeek-70B	95.4%	92.8%	76.3%
Seed-32B	98.0%	97.1%	92.4%

Table 9: Task success under Our attack, separated from ASR. “Benign” denotes benign TSR, and “Task” denotes task completion under Our attack. ASR additionally requires the targeted resource-amplification behavior.

D.3 Separating Task Success from Attack Success

Motivation. In the main experiments, ASR is stricter than TSR: an episode counts as a successful attack only if the final task is solved and the targeted resource-amplification behavior occurs. To clarify the correctness effect under attack, we separately compute task success under Our attack and compare it with benign TSR.

Results. Table 9 shows that the attack usually preserves task utility close to the benign setting. Across all model-benchmark pairs, task success under attack remains within 2.6 percentage points of benign TSR. For example, on ToolBench, Llama-3.3-70B drops only from 98.1% benign TSR to 97.6% task success under attack, and Qwen-3-32B drops from 94.6% to 93.1%. On BFCL, Llama-3.3-70B drops from 100.0% to 99.2%, and Qwen-3-32B drops from 98.5% to 97.4%. The gap between task success and ASR reflects episodes where the agent still completes the task but does not fully satisfy the targeted multi-turn or length condition. This separation strengthens our central claim: output correctness alone is not a reliable indicator of safe or efficient agent execution.

D.4 Cross-LLM Transferability of Optimized Templates

Setup. The seed bank is designed to reuse successful templates across runs. To test whether MCTS-optimized templates overfit to a single target LLM, we directly apply templates optimized on one LLM to another LLM without additional MCTS search. We evaluate transfer between Llama-3.3-70B and Qwen-3-32B on both ToolBench and BFCL. Target-specific rows are included as upper-bound references for each target model.

Results. Table 10 shows that optimized templates transfer across LLMs. On ToolBench, a Llama-optimized template transferred to Qwen-3-32B obtains 54,860 tokens and 68.7% ASR, outperforming the non-MCTS Hand-crafted baseline on Qwen-3-32B in Table 2. Conversely, a Qwen-optimized template transferred to Llama-3.3-70B obtains 75,640 tokens and 91.8% ASR. Transfer is asymmetric: Qwen-to-Llama transfer is stronger, likely because Llama-3.3-70B is more compliant with long-form tool outputs, while Llama-to-Qwen

ToolBench Dataset			
Target	Source	Length	ASR
Llama-3.3-70B	Llama-3.3-70B	81,830	96.2%
Llama-3.3-70B	Qwen-3-32B	75,640	91.8%
Qwen-3-32B	Qwen-3-32B	65,273	80.5%
Qwen-3-32B	Llama-3.3-70B	54,860	68.7%
BFCL Dataset			
Target	Source	Length	ASR
Llama-3.3-70B	Llama-3.3-70B	77,052	93.9%
Llama-3.3-70B	Qwen-3-32B	71,230	88.6%
Qwen-3-32B	Qwen-3-32B	67,585	82.7%
Qwen-3-32B	Llama-3.3-70B	58,420	70.9%

Table 10: Cross-LLM transferability of MCTS-optimized templates. Target and Source denote the tested LLM and the LLM used to optimize the template, respectively. Same-source rows are target-specific references; cross-source rows are direct transfer results without additional MCTS search.

ToolBench Dataset				
Variant	Llama-3.3-70B		Qwen-3-32B	
	Length	ASR	Length	ASR
V1: integer list	81,830	96.2%	65,273	80.5%
V2: pseudo-JSON	73,680	91.5%	60,420	74.5%
BFCL Dataset				
Variant	Llama-3.3-70B		Qwen-3-32B	
	Length	ASR	Length	ASR
V1: integer list	77,052	93.9%	67,585	82.7%
V2: pseudo-JSON	70,940	89.7%	62,860	77.6%

Table 11: Structured artifact variants. V1 is the original integer-list implementation, while V2 replaces the comma-separated list with pseudo-JSON records while preserving the same completeness-gated tool-response mechanism.

transfer remains weaker but still effective. Target-specific MCTS remains best on each target, but a single optimized template already yields strong amplification at zero additional search cost.

D.5 Structured Artifact Variant Against Surface-Form Rules

Motivation and setup. A simple schema or regex defense could reject one disclosed surface form, such as a long comma-separated integer list. However, the attack mechanism does not require integers. It only requires a long-form intermediate artifact whose completeness is checked by the tool-response policy before the benign terminal payload is returned. To test this, we instantiate a pseudo-JSON record variant. Instead of a comma-separated integer list, the required artifact is a structured sequence of records, e.g., `[{id:1, checksum:..., ts:...}, ...]`. The completeness condition is preserved through monotonic identifiers and mandatory keys, but the payload is no longer a comma-separated integer list.

Results. Table 11 shows that the attack is not tied to numeric lists. The pseudo-JSON variant still produces more than 60K tokens on both benchmarks and both target models. On ToolBench, it reaches 73,680 tokens with 91.5% ASR on Llama-3.3-70B and 60,420 tokens with 74.5% ASR on Qwen-3-32B. On BFCL, it reaches 70,940 tokens with 89.7% ASR on Llama-3.3-70B and 62,860 tokens with 77.6%

Method	ToolBench	BFCL
Benign (FPR)	0.7%	0.9%
Overthink	95.8%	94.6%
Overthink-mt	97.2%	96.1%
Hand-crafted	2.7%	2.4%
Our attack	2.5%	2.2%

Table 12: PromptArmor-style guardrail evaluation. We apply a GPT-4o-based detector to the user query and the first tool response. Benign reports FPR, while attack rows report detection rate.

ASR on Qwen-3-32B. Therefore, a hard-coded rule against comma-separated integer lists can block one disclosed instantiation, but it does not address the underlying tool-layer DoS mechanism. Robust mitigation should instead monitor abnormal multi-turn tool behavior, resource growth, and repeated completeness-gated intermediate artifacts.

D.6 PromptArmor-Style Guardrail Evaluation

Setup. We additionally evaluate a PromptArmor-style LLM guardrail (Shi et al., 2026) under the same conservative input-detection setting as the PPL filter in §4.5. Specifically, we apply a GPT-4o-based guardrail to both the user query and the first tool response, and flag an episode if either side is identified as suspicious. For attack methods, we report detection rate; for benign episodes, we report false-positive rate (FPR). Results are pooled across all six target LLMs.

Results. Table 12 shows that Overthink-style baselines are detected at high rates, likely because their triggers contain explicit task-redirecting or slowdown-style instructions. By contrast, Hand-crafted and Our attack are detected at only 2.7%/2.4% and 2.5%/2.2% on ToolBench/BFCL, respectively. The near-identical detection rates of Hand-crafted and Our attack indicate that MCTS primarily improves stability and ASR rather than detector evasion. These results further support our claim that input-side or first-response detectors are insufficient for protocol-compatible tool-layer amplification.