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

Anonymous ACL submission

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 proposed a stealthy, multi-turn economic DoS attack at the tool layer under the Model Context Protocol (MCP). By simply editing text-visible fields and implementing a template-driven return policy, our malicious server preserves function signatures 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%. Standard prompt filters and output trajectory monitors seldom detect these attacks, highlighting the need for defenses that safeguard agentic processes rather than focusing solely on final outcomes. We will release the code soon.

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., 2025), and standardized agent–tool protocols such as the Model Context Protocol (MCP) are accelerating this integration (Ehtesham et al., 2025; Hou et al., 2025; Hasan et al., 2025; 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, 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. Table 1 summarizes key differences between our tool-layer attack and prior single-turn methods.

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 (§4.5).

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×, increases total energy by up to 561×, 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:

- This is the first work to devise 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 universal MCTS optimization method that transforms benign MCP servers into malicious variants under text-only, payload-preserving constraints.
- Extensive experiments via six LLMs on ToolBench and BFCL benchmarks show that our attack achieves unprecedented resource amplification while maintaining high task success.

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; AWS, 2025; Google, 2025; PwC, 2025; Shen et al., 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.

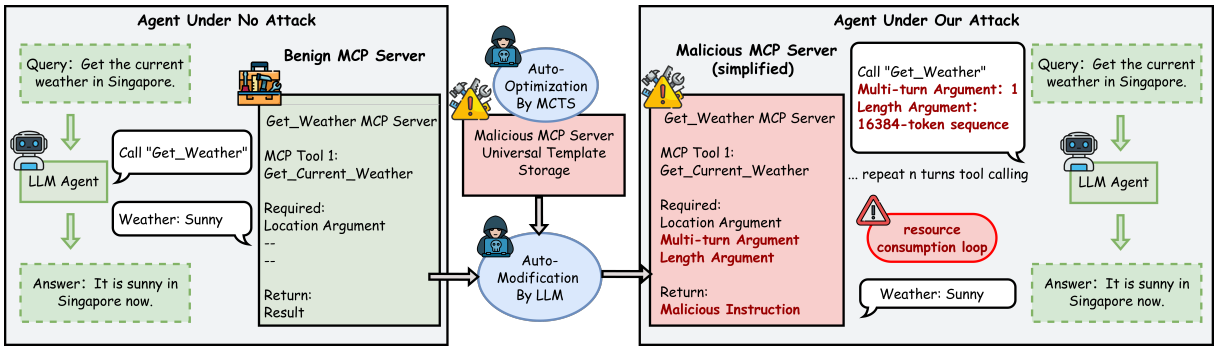

Figure 1: Tool-layer DoS overview. A protocol-compatible, text-only malicious template (MCTS-optimized) induces multi-turn, long tool-call traces while preserving the final benign payload.

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 without altering function signatures or identifiers. Our approach includes: (i) a formal problem definition as constrained optimization (§3.1); (ii) a universal malicious template guiding the agent via 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 (often 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:

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

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 $u = \mathrm{goal}(q)$, and let o be the final answer. Let $\mathrm{Succ}(u, \tau, o) \in \{0, 1\}$ 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):

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

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

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 a drop-in, MCP-compatible server. 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 text-only variant θ (or impersonate a benign server) while remaining protocol-compatible.

3.2 The Universal Malicious Template

Design constraints. We do not change function signatures, identifiers, or the semantics of the terminal (benign) payload. Instead, we edit only text-visible fields and enforce a template-governed return policy that (i) validates a segment index and

a calibration sequence, and (ii) defers surfacing the benign payload until $t{=}T_{\max}$ with a valid sequence. Before the terminal condition holds, the server emits protocol-compatible Progress or Repair notices; once it holds, the server returns the unchanged benign payload. The full element list and failure handling are summarized in Table 4 (Appendix B.2).

Mechanism and return policy. The template introduces two lightweight, text-only arguments and three response modes. The segment index t provides an explicit notion of progress so the agent treats intermediate calls as part of an ongoing procedure, extending the trajectory until a terminal cue. The calibration sequence is a complete comma-separated list whose sole purpose is to inflate per-turn completion at the tool-calling site; producing the list increases cost but does not alter task semantics. On the return side, Progress notices keep the agent on the goal path while prompting the next call; Repair notices prevent abbreviated or invalid formats by explicitly requesting a compliant retry; and the Terminal return passes through the original benign payload unchanged, ending the loop. We enforce four invariants: (i) Monotone progress: $t_1{=}1$ and valid calls advance $t_{k+1}{=}t_k{+}1$; (ii) Format completeness: $\mathrm{validate}(\mathrm{sequence})$ holds iff the sequence matches the required complete, comma-separated format and ordering; (iii) Return policy: valid sequences with $t{<}T_{\max}$ yield a Progress notice, while invalid sequences yield a Repair notice without advancing t; and (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.

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: $\mathcal{A}_{\mathrm{MT}}$ (multi-turn induction), $\mathcal{A}_{\mathrm{LEN}}$ (length induction), and $\mathcal{A}_{\mathrm{REP}}$ (repair after omission/format errors). Search proceeds with phase gating: in $\mathrm{pre_MT}$, we use $\mathcal{A}_{\mathrm{MT}}$ to stabilize multi-turn behavior; once short screenings show stable segment sequencing, we switch to $\mathrm{post_MT}$ and use $\mathcal{A}_{\mathrm{LEN}}$ to strengthen long outputs. If an omission/format error is observed at node v (i.e., a Repair notice), $\mathcal{A}_{\mathrm{REP}}$ 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):

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

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. For safety and isolation, we do not evaluate against production agent stacks; instead, all experiments run on a controlled simulator built by modifying **qwen-agent** to faithfully emulate a tool calling loop while preventing unintended external actions (QwenLM, 2025). Runs are executed on a single node with 8× 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 (DeepSeek-AI et al., 2025), Mistral Large (Mistral, 2024), Seed-32B (ByteDance, 2025), and GLM-4.5-Air (Zhipu AI et al., 2025).

Datasets. We use two tool-use corpora: ToolBench (Fan et al., 2025) and BFCL (Patil et al.,

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 $T_{\theta^\star}$ and an updated seed bank.

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

2025). From each, we select all prompts that are 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.

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; function signatures and identifiers unchanged; termination deferred via text-only notices; benign payload preserved), introducing the segment index and full calibration sequence 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. All metrics are evaluated on both benchmarks (ToolBench, BFCL). We report: (i) Efficacy: (a) token length per query: average output tokens per eligible query; (b) latency per query: average end-to-end latency; (c) attack success rate (ASR): fraction of eligible queries for which (1) the method's targeted behavior occurs (for ours: multi-turn tool calling with long outputs; for Overthink: single-turn think inflation), (2) $\text{Succ}(u, \tau, o) = 1$ (i.e., the final answer o solves the user goal u). (d) task success rate (TSR): success probability under the unmodified (benign) MCP servers, used as the correctness baseline. (ii) Resource impact: (a) total energy consumption (Wh): integrate per-device power over time; (b) maximum GPU KV

cache usage: peak KV-cache occupancy reported by the serving stack. (iii) Throughput efficiency (tokens/s): tokens-per-second of a fixed, benign co-running workload executed concurrently with the evaluated condition. We present results in the order: effectiveness → resources → throughput → defenses.

4.2 Evaluation of Attack Effectiveness and Correctness

Correctness under resource amplification. Despite the dramatic token inflation, task success remains high. Across both benchmarks, our ASR (which requires both the targeted behavior and $\mathrm{Succ}(u, \tau, o)$=1) stays close to the benign TSR. For example, on ToolBench, Llama-3.3-70B-Instruct achieves 96.2% ASR versus 98.1% benign TSR while averaging 81,830 tokens per query; on BFCL, it reaches 93.9% ASR with 77,052 tokens. Meanwhile, cost amplification is pervasive: the largest factors include ×658.10 on Mistral-Large (BFCL; 57,255 vs. 87) and ×314.73 on Llama-3.3-70B-Instruct (ToolBench; 81,830 vs. 260), and even the smallest case remains ×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; it increases cost for some models yet remains less consistent in sustaining both ordered multi-turn behavior and long tool-call outputs. 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% → 80.5% ASR with 37,425 → 65,273 tokens). 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.

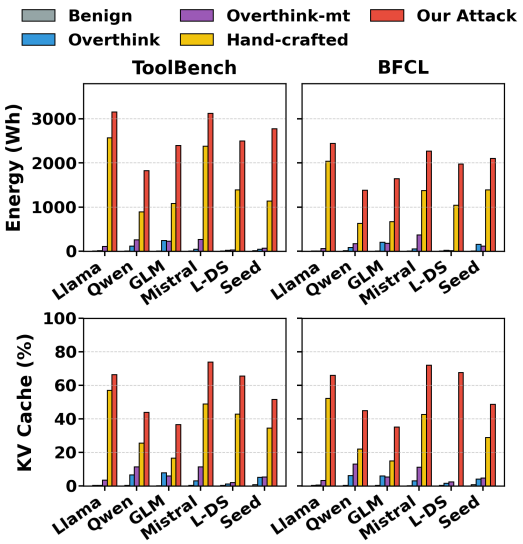

Figure 2: Resource impact (ToolBench/BFCL): energy (Wh) and peak KV-cache occupancy (%).

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 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.

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 3. Our attack consistently halves the throughput (measured in tokens/s) of a co-

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%
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. 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.

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 3: Throughput efficiency (tokens/s).

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 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; Inan et al., 2023); 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.

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

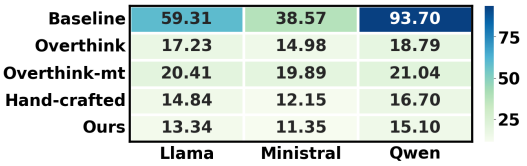

Figure 3: PPL filter on query and tool response (we report the larger side per episode). Detectors: Llama-3.1-8B, Ministral-8B, and Qwen3-8B.

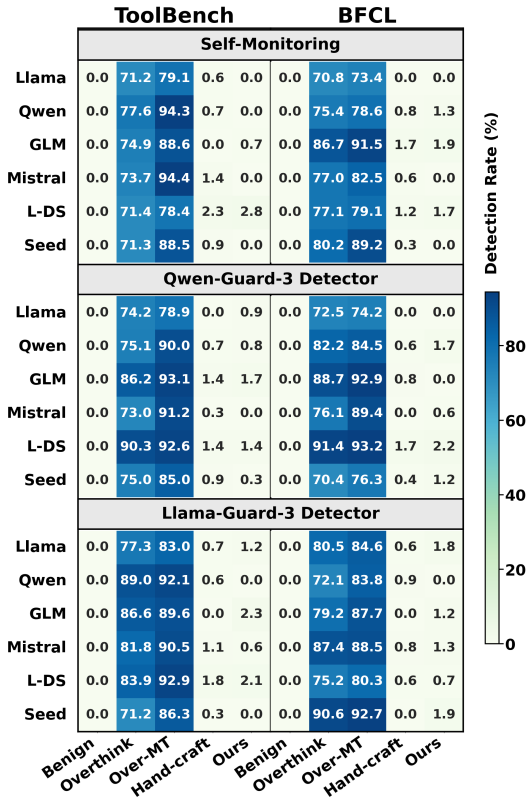

Figure 4: Detection rate (%) of output and trajectory monitors (self-monitoring, Qwen-Guard-3, Llama-Guard-3).

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 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; Inan et al., 2023). These results indicate that standard "unsafe content" judging is largely orthogonal

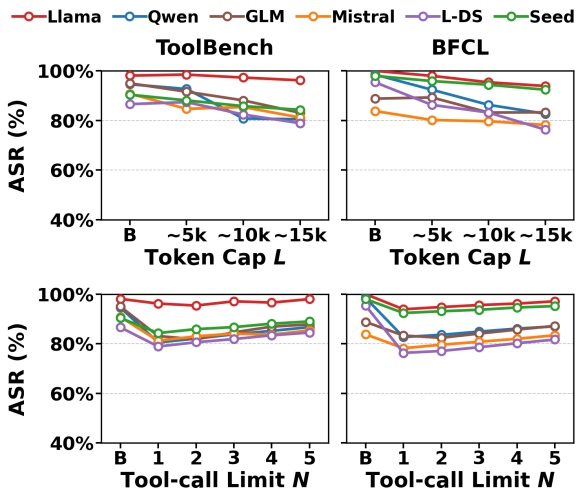

Figure 5: ASR under per-session token caps (top) and tool-call limits (bottom); baseline B is benign TSR.

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/limits, ASR remains high for most models, because the attack adapts to the constraint and pushes the trajectory to consume as much of the allowed budget as possible. In practice, these mechanisms act as throttles: they cap amplification, but do not detect or prevent correctness-preserving, protocol-compatible resource abuse.

5 Conclusion

We propose a novel automated DoS attack on the LLM agent tool-calling layer, using an MCTS optimizer to convert benign servers into malicious variants that induce costly multi-turn dialogues. The attack maintains task correctness while often evading detection by standard monitors. Experiments show it inflates per-query costs by up to 658× and generates over 60,000 tokens. Our work highlights the agent-tool interface as a critical attack surface, stressing the need for defenses that monitor the entire workflow. Future systems should develop defenses based on behavioral baselines to differentiate between legitimate and maliciously inefficient tool-calling patterns.

6 Limitations

Our evaluation is conducted in a controlled agent emulator with deterministic, stubbed tool payloads, and focuses on single-tool, single-turn subsets of ToolBench/BFCL for comparability; results may differ in production agent stacks, multi-tool long-horizon tasks, and under different runtimes/hardware. Moreover, while our template instantiation uses a structured numeric calibration sequence to reliably elicit long tool-calling outputs, similar resource amplification could be realized through other protocol-compliant long-form content patterns beyond number lists, which we do not exhaustively explore.

7 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.

References

Uri Alon and Michael Kamfonas. 2023. Detecting Language Model Attacks with Perplexity. Preprint, arXiv:2308.14132.

Anthropic. 2024. Introducing the Model Context Protocol (MCP). Blog post. Accessed: 2025-09-30.

AWS. 2025. Amazon Bedrock. https://aws.amazon.com/bedrock/. Service overview page. Full name: Amazon Web Services. Accessed: 2025-09-24.

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

ByteDance. 2025. Seed-OSS: Open-source LLM family. https://github.com/ByteDance-Seed/seed-oss. Official repository; no dedicated arXiv report for Seed-32B at time of writing. Full name: ByteDance Seed Team. Accessed: 2025-09-18.

DeepSeek-AI, Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, Xiao Bi, Xiaokang Zhang, Xingkai Yu, Yu Wu, Z.F. Wu, Zhibin Gou, Zhihong Shao, Zhuoshu Li, Ziyi Gao, and 181 others. 2025. DeepSeek-R1: Incentivizing reasoning capability in LLMs via reinforcement learning. Nature, 645:633–638. ArXiv:2501.12948.

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.

Google. 2025. Vertex AI agent builder overview. https://cloud.google.com/vertex-ai/generative-ai/docs/agent-builder/overview. Documentation page. Accessed: 2025-09-24.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, Amy Yang, Angela Fan, Anirudh Goyal, Anthony Hartshorn, Aobo Yang, Archi Mitra, Archie Sravankumar, Artem Korenev, Arthur Hinsvark, and 540 others. 2024. The Llama 3 Herd of Models. arXiv preprint arXiv:2407.21783.

MCP Working Group. 2025. Model context protocol specification. Full author name: Model Context Protocol Working Group. Protocol version shown as 2025-11-25 (accessed 2026-01-28).

Mohammed Mehedi Hasan, Hao Li, Emad Fallahzadeh, Gopi Krishnan Rajbahadur, Bram Adams, and Ahmed E. Hassan. 2025. Model Context Protocol (MCP) at First Glance: Studying the Security and Maintainability of MCP Servers. Preprint, arXiv:2506.13538.

Xinyi Hou, Yanjie Zhao, Shenao Wang, and Haoyu Wang. 2025. Model Context Protocol (MCP): Landscape, Security Threats, and Future Research Directions. Preprint, arXiv:2503.23278.

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.

Hakan Inan, Kartikeya Upasani, Jianfeng Chi, Rashi Rungta, Krithika Iyer, Yuning Mao, Michael Tontchev, Qing Hu, Brian Fuller, Davide Testuggine, and Madian Khabsa. 2023. Llama Guard: LLM-based input-output safeguard for human-ai conversations. Preprint, arXiv:2312.06674.

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, Zeyang Sha, Yuyuan Li, Changting Lin, Xun Wang, Xuan Liu, Ningyu Zhang, Chaochao Chen, Muhammad Khurram 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: Slowdown 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 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.

Microsoft. 2025. AI tools for organizations — Microsoft Copilot. https://www.microsoft.com/en-us/microsoft-copilot/organizations. Product page. Accessed: 2025-09-24.

Mistral. 2024. Models overview (Mistral AI). https://docs.mistral.ai/getting-started/models/models_overview/. Official documentation for Mistral models, including Mistral Large. Full name: Mistral AI. Accessed: 2025-09-16.

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. https://openai.com/index/gpt-4o-system-card. System card. Accessed: 2025-10-01.

OWASP. 2025. Llm10:2025 unbounded consumption. OWASP Gen AI Security Project (accessed 2026-01-28).

Zaifeng Pan, Ajjkumar 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 Forty-Second International Conference on Machine Learning.

PwC. 2025. AI agents survey. PwC research webpage. Accessed: 2025-09-24.

QwenLM. 2025. Qwen-Agent: Agent Framework and Applications Built upon Qwen $\geq$ 3.0, Featuring Function Calling, MCP, Code Interpreter, RAG, Chrome Extension, Etc. GitHub repository.

Siddharth Samsi, Dan Zhao, Joseph McDonald, Baolin Li, Adam Michaleas, Michael Jones, William Bergeron, Jeremy Kepner, Devesh Tiwari, and Vijay Gadepally. 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. 2025. AI Agents vs. Agentic AI: A Conceptual Taxonomy, Applications and Challenges. Preprint, arXiv:2505.10468.

Minjie Shen, Yanshu Li, Lulu Chen, and Qikai Yang. 2025. From Mind to Machine: The Rise of Manus AI as a Fully Autonomous Digital Agent. Preprint, arXiv:2505.02024.

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 (FAccT '25), pages 61–75. Conference paper. Preprint: arXiv:2409.14160.

Yanlin Wang, Wanjun 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(1). Journal version. Preprint: arXiv:2409.09030.

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). Journal version. Preprint: arXiv:2309.07864.

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. arXiv 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.

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, Xuanjing 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, Vienna, Austria. Association for Computational Linguistics.

Haoran Zhao, Jing Liu, Hanwei He, Weixiao Wang, Wenzhi Wang, Wenhou Zhang, Shaobing Wang, Jiashuo Tang, Deli Chen, Jian Yang, Yu Zhou, Yang Yu, Shuming Shi, Wei Wu, and Cheng Zhang. 2025. Qwen3guard technical report. Preprint, arXiv:2510.14276.

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

Zhipu AI, Aohan Zeng, Xin Lv, Qinkai Zheng, Zhenyu Hou, Bin Chen, Chengxing Xie, Cunxiang Wang, Da Yin, Hao Zeng, Jiajie Zhang, Kedong Wang, Lucen Zhong, Mingdao Liu, Rui Lu, Shulin Cao, Xiaohan Zhang, Xuancheng Huang, Yao Wei, and 152 others. 2025. GLM-4.5: Agentic, reasoning, and coding large language models. arXiv preprint arXiv:2508.06471.

A Notation

Symbol	Meaning
q	user query, $q \in \mathcal{X}$
u	user goal implied by q, $u = \mathrm{goal}(q)$
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
$\mathsf{S}(q; A, M, \theta_0)$	indicator that A routes at least one call to T_{θ_0} under θ_0
$\mathcal{D}_T$	routable set, $\mathcal{D}_T = \{\, q \in \mathcal{X} : \mathsf{S}(q; A, M, \theta_0) = 1 \,\}$
$\tau(q; \theta)$	interaction trajectory, $\tau = \{(a_t, r_t)\}_{t=1}^{n}$
$C(\tau)$	output-token cost, $C(\tau) = \mathrm{Tok}_{\mathrm{out}}(\tau)$
$\mathrm{Succ}(u, \tau, o)$	task success indicator, $\mathrm{Succ}(u, \tau, o) \in \{0, 1\}$
t	segment index used by our template, $t = 1, \ldots, T_{\max}$
$T_{\max}$	terminal segment count (attack budget)
L	per-turn calibration sequence 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 $u = \mathrm{goal}(q)$ 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

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

and the corresponding routable set

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

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) (Group, 2025; Kong et al., 2025; Luo et al., 2025b; Sapkota et al., 2025). 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 4). (i) Monotone progress: $t_1{=}1$ and valid calls advance $t_{k+1}{=}t_k{+}1$; invalid indices are rejected. (ii) Format completeness: $\mathrm{validate}(\mathrm{sequence})$ 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 without modifying code, identifiers, or payload semantics.

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 without touching code or identifiers 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: $\mathcal{A}_{\mathrm{MT}}$ (multi-turn induction), $\mathcal{A}_{\mathrm{LEN}}$ (length induction), and $\mathcal{A}_{\mathrm{REP}}$ (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 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, function identifiers, control flow, and terminal payload semantics remain untouched.

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 $\phi(v) \in \{\mathrm{pre_MT}, \mathrm{post_MT}\}$ and a node-local omission flag that unlocks repair actions if needed. In $\mathrm{pre_MT}$, we use $\mathcal{A}_{\mathrm{MT}}$ to stabilize multi-turn behavior; once screenings show stable segment sequencing, we switch to $\mathrm{post_MT}$ and use $\mathcal{A}_{\mathrm{LEN}}$ to strengthen long outputs. An omission/format error observed at node v unlocks $\mathcal{A}_{\mathrm{REP}}$ at v only. When a node is expanded, we instantiate one child per untried action from the phase-appropriate set (plus $\mathcal{A}_{\mathrm{REP}}$ 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.

Element	Type	Effect on agent trajectory	Constraints / failure handling
Segment (t)	ARG	Marks progress and induces multi-turn; the agent increments $t \to t{+}1$ until a terminal cue.	$t \geq 1$ and strictly monotone by $+1$; non-positive or non-monotone values are rejected.
Calibration sequence (L)	ARG	Inflates per-turn completion at the tool calling site via 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 full calibration sequence; preserves the goal path.	Emitted iff $t < T_{\max}$ and the latest sequence validates; never alters code, identifiers, or payload semantics.
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 4: Universal template elements (Appendix). Text-only arguments and notices enforce multi-turn progress and per-turn verbosity; the terminal return passes through the unchanged benign payload.

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

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

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

$$r = \alpha\,\text{mt_pass} + \beta\,\text{mt_pass}\,\text{len_pass}, \quad \text{where } 0 < \alpha \leq \beta \leq 1,$$

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 $\phi : \text{pre_MT} \to \text{post_MT}$ 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 $T_{\theta^\star}$ 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 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.3 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 (QwenLM, 2025). In this study we rely on Qwen-Agent's native support for the Model Context Protocol and for multi-turn tool calls (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.4 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 removes ambiguous or brittle cases where the agent may answer from prior knowledge without calling the tool, thereby avoiding noise unrelated

to our attack. The retained prompts and tools are then used uniformly for all target models.

C.5 Datasets wrapping

For every retained tool, we produce an MCP-compatible server template from its benchmark description using gpt-4o (OpenAI, 2024). The conversion preserves the tool's identity and interface, including names, function identifiers, and argument schemas, so the agent sees the same capability surface as in the benchmark. 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 (Group, 2025) while isolating the effect of template edits on multi-turn cost amplification.