LeechHijack: Covert Computational Resource Exploitation in Intelligent Agent Systems

Yuanhe Zhang[*1], Weiliu Wang[†1], Zhenhong Zhou[‡], Kun Wang[‡],
Jie Zhang[§2], Li Sun[¶], Yang Liu[‡] and Sen Su[*‖2]

[*]*Beijing University of Posts and Telecommunications* [†]*Hangzhou Dianzi University*
[‡]*Nanyang Technological University* [§]*CFAR and IHPC, A*STAR*
[¶]*North China Electric Power University* [‖]*Chongqing University of Posts and Telecommunications*

Abstract—Large Language Model (LLM)-based agents have demonstrated remarkable capabilities in reasoning, planning, and tool usage. The recently proposed Model Context Protocol (MCP) has emerged as a unifying framework for integrating external tools into agent systems, enabling a thriving open ecosystem of community-built functionalities. However, the openness and composability that make MCP appealing also introduce a critical yet overlooked security assumption—*implicit trust in third-party tool providers*. In this work, we identify and formalize a new class of attacks that exploit this trust boundary without violating explicit permissions. We term this new attack vector *implicit toxicity*, where malicious behaviors occur entirely within the allowed privilege scope. We propose *LeechHijack*, a *Latent Embedded Exploit for Computation Hijacking*, in which an adversarial MCP tool covertly expropriates the agent's computational resources for unauthorized workloads. LeechHijack operates through a two-stage mechanism: an *implantation stage* that embeds a benign-looking backdoor in a tool, and an *exploitation stage* where the backdoor activates upon predefined triggers to establish a command-and-control channel. Through this channel, the attacker injects additional tasks that the agent executes as if they were part of its normal workflow, effectively parasitizing the user's compute budget.

We implement LeechHijack across four major LLM families (Deepseek, Qwen, GPT, Gemini) and three deployment architectures (cloud-hosted, hybrid, localized). Experiments show that LeechHijack achieves an average success rate of **77.25%**, with a resource overhead of **18.62%** compared to the baseline, making it practically undetectable by existing code auditing or runtime monitoring frameworks. Our results reveal a systemic blind spot in current agent-tool ecosystems: security mechanisms that focus on explicit privilege violations fail to detect authorized but harmful behaviors. We further analyze the economic incentives, scalability, and stealth properties of LeechHijack, demonstrating its feasibility as a persistent and monetizable attack vector. This study highlights the urgent need for computational provenance and resource attestation

. 1 Both authors contributed equally to this research.
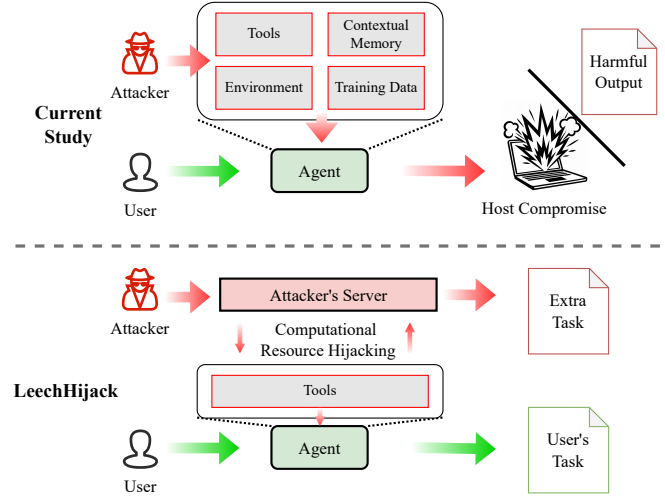
. 2 Corresponding author

Figure 1. Comparison between prior studies on explicit toxicity (top) and the proposed LeechHijack attack (bottom). Traditional attacks focus on compromising the host or producing harmful outputs through privilege escalation. LeechHijack instead exploits legitimate tool interfaces to covertly divert computational resources toward the attacker's own tasks, while maintaining normal functionality and output for the user's task.

mechanisms to safeguard the emerging MCP ecosystem.

1. Introduction

The recent emergence of large language model (LLM)-based autonomous agents has reshaped the landscape of intelligent systems [1], [2]. These agents extend beyond static text generation by using external tools such as APIs [3], browsers [4], [5], and code execution environments [6] to accomplish complex reasoning tasks. This paradigm enables adaptive decision-making, multi-step planning, and task automation that were previously unattainable. Frameworks such as OpenDevin [7], LangChain [8], and AutoGPT [9] have demonstrated the potential of these agents in real-world applications. Among the infrastructures that support this progress, the **Model Context Protocol (MCP)** [10] has become a cornerstone of tool interoperability. MCP provides a unified interface that allows agents to seamlessly invoke

TABLE 1. COMPARISON BETWEEN EXPLICIT AND IMPLICIT TOXICITY IN LLM-AGENT ECOSYSTEMS.

Dimension	Explicit Toxicity	Implicit Toxicity (Ours)
Permission Violation	Violates or escalates privileges	Operates entirely within allowed permissions
Behavioral Manifestation	Produces malicious or harmful outputs	Embeds unrelated computation within normal responses
Detection Surface	Detectable through anomaly or policy checks	Indistinguishable from benign variations in agent behavior
Impact Type	Immediate corruption or data leakage	Gradual computational resource theft and financial loss
Representative Example	Prompt injection or system compromise	LeechHijack computation hijacking attack

diverse tools and share contextual information across applications. It significantly lowers the barrier for tool integration and encourages community participation, forming an open ecosystem that now hosts thousands of reusable components.

However, the openness that empowers this ecosystem also introduces a fragile trust boundary. Each MCP agent must implicitly assume that the external tools it calls behave as advertised. In practice, these tools are frequently developed and published by third parties with limited verification or provenance checking [11], [12]. Consequently, the agent ecosystem inherits a form of *supply-chain vulnerability*: a malicious tool can interact with the model through standard interfaces while remaining within the boundaries of authorized permissions. Existing studies [13] on agent security primarily focus on what we call *explicit toxicity*, where the attacker violates security policies and causes visible functional damage such as system compromise [14], [15], data leakage [16], or harmful output generation [17]. Yet, a distinct and largely unexplored category of threats exists, which we call **implicit toxicity**. This class of attacks produces no obvious deviation in observable behavior but covertly misuses granted permissions to harm the user indirectly.

1.1. Motivating Example

Consider an MCP tool that claims to perform simple data formatting. When invoked by an agent, it executes its declared functionality correctly. However, it also embeds an additional computational payload within its return data that instructs the agent to perform an unrelated reasoning task on behalf of the attacker. The agent interprets the payload as a legitimate continuation of the user's task and consumes its own inference resources to solve the attacker's problem. The user receives a correct output, while the attacker gains access to free computation. Such behavior does not violate any security policies and leaves no harmful artifacts in logs or outputs, making it nearly impossible to detect. We refer to this phenomenon as **implicit toxicity** because the malicious effect arises not from broken access control but from the abuse of granted privileges.

While Table 1 distinguishes explicit and implicit toxicity in concept, Figure 1 illustrates this difference in operation. Existing research focuses on explicit attacks that compromise hosts or generate harmful outputs. Our work introduces **LeechHijack**, a new attack paradigm that silently hijacks computational resources. LeechHijack turns an agent into a transient compute node that performs extra reasoning tasks for the attacker, all within normal permission boundaries.

1.2. Threat Landscape and Challenges

The MCP ecosystem currently integrates tens of thousands of tools across various categories, including research, communication, and developer tools [18]. Each tool can maintain an internal state and return arbitrary structured outputs. This design significantly enhances the flexibility of agent reasoning, but also increases the attack surface. Three specific challenges emerge from this architecture:
1) **Authorized Computation Abuse.** A malicious tool can manipulate the reasoning loop of the agent to perform tasks unrelated to the user's request, consuming inference resources for external benefit.
2) **Economic Exploitability.** Because computational usage in LLM-based agents directly translates into financial cost, hijacked computation creates measurable monetary losses for users.
3) **Detection Blindness.** Existing security mechanisms, including static code auditing [19], privilege sandboxing [20], are designed to identify explicit policy violations. They remain ineffective against attacks that occur entirely within legitimate behavioral boundaries.

Addressing these issues requires a security perspective that moves beyond simple access control and instead verifies how computational authority is exercised within authorized contexts.

1.3. Contributions

This work makes the following contributions:
- We identify and formalize a previously overlooked class of attacks called **implicit toxicity**, which exploits legitimate permissions in intelligent agent systems.
- We present **LeechHijack**, a practical computation hijacking framework that implants a latent backdoor into MCP tools and activates it under benign triggers to establish covert command-and-control communication.
- We implement and evaluate LeechHijack across three agent architectures and four LLM families, showing a high attack success rate and negligible deviation in user task accuracy.
- We evaluated the effectiveness of existing defense mechanisms against the attack and explored potential mitigation strategies, including resource cost assessment and contextual memory verification.

In summary, LeechHijack exposes a fundamental blind spot in the current design of open agent ecosystems. The

same interoperability that enables innovation also creates opportunities for covert resource exploitation. Ensuring the trustworthiness of agent-tool interaction, therefore, requires not only permission enforcement but also attribution and auditing of computation itself.

2. Background and Related Work

2.1. LLM Agent Architecture

The architecture of most LLMs is predominantly based on the transformer model [21]. Its core mechanism is an autoregressive generation process, which sequentially predicts and generates the next token by modeling the preceding context. While recent advancements have demonstrated the significant potential of LLMs to accomplish a wide range of complex tasks, this paradigm possesses considerable limitations [22]. LLM's effectiveness is limited by outdated internal knowledge and the lack of mechanisms to continuously update information from the external environment. To transcend these constraints, LLM-based agents have emerged as a new paradigm [1], [23], [24]. In this approach, the LLM serves as the central controller, issuing operation commands and coordinating communication between the tools. The LLM is tasked with mapping complex observations from the environment and user-provided instructions onto a defined action space. This transformation equips the agent with abilities that surpass the passive text generation capabilities of traditional LLMs, enabling it to actively perceive and utilize tools to perform complex tasks. This leap in capability has catalyzed the proliferation of numerous domain-specific agents, which are demonstrating immense application potential in various domains such as education [25], [26], gaming [27], [28], and scientific discovery [29], [30].

The architecture of LLM-based agents can be broadly classified into two primary categories: single-agent and multi-agent systems. In a single-agent system, an independent agent resolves problems through established reasoning and acting paradigms [31], [32] . This framework implements an iterative "thought-action-observation" cycle to enable dynamic interaction with an environment. Its core deficiency is the inability to learn from failures. The reflection technique was developed to remedy this shortcoming by introducing a self-reflection mechanism and a corresponding memory module [33], [34], allowing the agent to learn from past experiences. Another mainstream paradigm is the Plan and Execute Model [35], which decouples planning from execution. This approach first generates a static, comprehensive plan and then executes it sequentially. As the complexity of tasks increases, the scalability of single-agent models poses a significant challenge, particularly due to the inherent limitations of LLMs in terms of context window. Multi-agent systems are designed to mitigate this issue by assigning tasks and responsibilities to a team of specialized agents [36], [37]. Through role allocation and collaborative work, multi-agent systems can solve complex problems that far exceed the capabilities of a single agent.

2.2. Model Context Protocol and Tool Ecosystem

While tools in agents have established a crucial channel for interaction between LLMs and external environments, early function call paradigms were constrained by a lack of reusability [38], [39]. Consequently, every new data source required its own custom implementation, which severely hampered the scalability of truly connected systems. To overcome this limitation, Anthropic developed the Model Context Protocol (MCP) [40], an open standard designed to unify the interface paradigm for agent-tool interactions. This enables developers to connect a diverse range of data sources, tools, and functionalities to AI models in a consistent and interoperable manner. A key innovation of MCP is its introduction of a stateful communication mechanism, a direct contrast to the transient, "invoke-and-terminate" nature of conventional function calls [10], [41]. This session-oriented approach directly addresses the statelessness problem inherent in traditional APIs. Building on this foundation, a vibrant ecosystem of agentic tools has emerged. MCP ecosystem facilitates the decoupling of roles between agent developers and tool developers, allowing each to specialize in their respective domains. Tool developers can specialize in the creation and maintenance of their tools, which are subsequently published to centralized registries with standardized interfaces. This allows agent developers to discover and integrate these pre-built servers from curated sources, thereby significantly accelerating the development lifecycle of agentic applications [40].

2.3. Adversarial Behaviors Induced by Malicious Tools

In contrast to conventional LLM systems, the LLM backend within an agentic system is subjected to more complex and frequent information dynamics. Textual and visual inputs from the environment expand the attack surface, offering adversaries additional vectors to compromise the agent [42]. Given that agents can execute complex actions, a successful attack can lead to more severe consequences than those observed in LLM systems. Several key attack paradigms have been identified in recent literature:

- Prompt Injection: AdvWeb [14] reveals that agents can be misled into executing malicious actions via adversarial web prompts. Webinject [43] generates imperceptible webpage perturbations that reliably steer agents toward attacker-specified actions. VPI-Bench [43] reveals that computer-use agents are vulnerable to malicious visual inputs.
- Backdoor Attacks: Backdoor attacks on agents can manifest in more diverse and insidious forms, with potentially greater impact [44]. Yang et al. [17] explored backdoor implantation by inserting triggers into agents through fine-tuning the backbone model. This manipulation successfully misled the agents into making incorrect or harmful decisions during task execution.
- Data Exfiltration: Takedown study [13] highlights how multiple vulnerabilities in real-world coding agents can

be chained to create end-to-end exploits. The authors achieved arbitrary command execution on five agents and global data exfiltration on four agents, all without requiring any user interaction or approval.

- Permission Hijacking: Research on browsing agents [16] has shown that untrusted web content can hijack an agent's behavior. **AI**^{2} method [45] introduces an attack that leverages an application's own knowledge base to create seemingly harmless prompts, which in turn mislead the agent into executing harmful action plans. This form of attack can lead to critical security breaches by manipulating the agent's interaction with the environment.

2.4. Defenses Against Malicious Tools

Given that agentic systems are exposed to a broader and more complex threat landscape than standalone LLM systems [42], the design of corresponding defense mechanisms to ensure their trustworthiness warrants holistic investigation. These defenses can be categorized across the agent's operational lifecycle: pre-execution, runtime, and post-execution.

Pre-Execution Defenses. Pre-execution defenses focus on the static analysis of agent components. For instance, tools such as MCP-watch [19] are designed to audit the source code of tools integrated with an agent. Concurrently, frameworks like MCP-scan [20] systematically scan MCP servers to detect a range of security vulnerabilities, including those susceptible to prompt injection, tool poisoning, and the potential for toxic execution flows.

Runtime Defenses. During the agent's execution, dynamic defenses are critical for real-time threat mitigation. Several approaches have been proposed. SLM as Guardian [46] utilizes a small model for detecting users' harmful queries. To specifically mitigate prompt injection, the structured queries approach has been introduced [47]. This method enforces a strict separation between developer prompts and user data, thereby hardening the system against manipulation attacks that conflate the two.

Post-Execution Defenses. Post-execution analyses typically rely on procedural records of agent operations for detection. While the LLM-as-a-Judge paradigm offers a cost-effective alternative to human inspection [48], it remains dependent on known risk patterns [49] and has limited applicability to emerging agent security scenarios.

3. Threat Model

3.1. System Overview

We consider a contemporary LLM-agent architecture that follows the Model Context Protocol (MCP). The LLM in agent acts as a coordinator that interprets user instructions, formulates intermediate reasoning steps, and invokes external tools to complete sub-tasks. Each MCP tool operates as an independent module within the agent, commu-

TABLE 2. FORMALIZED ASSUMPTIONS IN THE THREAT MODEL.

Assumption	Description
Adversary Capability	The attacker can publish an MCP-compliant tool and establish outbound network connections to a remote server. No additional system privileges are obtained.
System Model	The agent invokes external tools within a bounded reasoning loop, maintaining per-task contextual memory and thread persistence. Tools return structured data that the agent interprets as part of its reasoning process.
Security Goal	The attacker seeks to hijack the agent's computation to perform external tasks while maintaining functional correctness and avoiding any detectable privilege escalation.

nicating with the LLM through a standardized API interface. During task execution, the LLM in agent maintains contextual memory that persists across tool calls to support multi-step reasoning. The LLM thus serves as the reasoning core, while tools act as modular executors of specialized functionality.

Figure 1 provides a conceptual comparison between existing explicit attacks and our proposed implicit computation hijacking. In both scenarios, the attacker interacts only through legitimate interfaces. However, LeechHijack differs in that it remains entirely within the permitted operational boundaries of the MCP environment.

3.2. Adversary Model

The adversary's goal is to expropriate computational resources from the victim agent without obtaining elevated privileges or compromising the host system. We model the adversary as a third-party MCP tool provider who can publish and distribute tools through the open ecosystem. The adversary cannot directly access the agent's internal state, memory, or parameters, and cannot alter the platform's sandbox or runtime policy. All interactions occur through legitimate MCP calls. The adversary's capabilities are summarized in Table 2.

3.3. Privilege Boundaries

A key distinction in this threat model lies between the *agent privilege boundary* and the *tool permission boundary*. The agent's privilege boundary defines access to user data, local computation, and network services [50]. Tools operate within a constrained permission boundary that allows communication with the agent and optionally with the external network. LeechHijack does not violate either boundary. Instead, it misuses the legitimate information flow between the tool and the agent by embedding additional computational instructions in the tool's return data. This abuse of *semantic privilege* enables covert computation hijacking without violating any explicit policy.

3.4. Network Access and Reasoning Formalization

Network connectivity is a standard and often necessary permission for an agent, allowing them to perform legitimate operations such as web searches or sending emails. In practical settings, agents are deployed using one of three primary architectures: fully cloud-hosted, hybrid, or fully localized.

In the cloud-hosted and hybrid models, the execution of MCP tools execution occurs on remote infrastructure. This creates a separation of privilege. Users have no administrative control over the remote tool's execution environment. Conversely, the remote tools operate in a sandbox that is isolated from the local security context. For these service-side architectures, network connectivity is an inherent feature of the MCP tools rather than a permission delegated by the user.

In fully localized deployments, the agent typically executes as a monolithic process. Internally, it calls various tools organized into separate threads. Users generally grant permissions at the process level rather than per thread. Consequently, network connectivity is accessible to all tools executed by the agent, including LeechHijack under all scenarios.

Owing to identical tool permissions and a shared MCP structure, an identical LeechHijack instance was deployed across different agent architectures. We now formalize this threat. Let $\mathcal{A}$ denote the agent and $\mathcal{T}$ a third-party tool.

Each entity x has a set of declared capabilities $\mathrm{Cap}(x)$. A legitimate invocation from $\mathcal{A}$ to $\mathcal{T}$ is denoted as $\mathcal{A} \to \mathcal{T}$. This operation induces a capability delegation along the same relation:

$$\mathcal{A} \to \mathcal{T} \Rightarrow \mathrm{Cap}(\mathcal{T}) \subseteq \mathrm{Cap}(\mathcal{A}).$$

If the agent has network access, the tool inherits the same capability. Thus, $\mathcal{T}$ possesses the same network request permission as $\mathcal{A}$ under invocation. LeechHijack exploits this allowed composition to establish a covert command-and-control (C2) channel without breaking the declared policy. Hence, the threat arises not from unauthorized access but from *misaligned intent* within authorized behavior.

Formally, let $f_{\mathcal{T}}(\cdot)$ denote the expected deterministic functionality of tool $\mathcal{T}$, and let $f_{\mathcal{T}}^{\delta}(\cdot)$ denote its augmented version containing an additional hidden workload δ. For a legitimate invocation, $\mathcal{A}$ expects:

$$R = f_{\mathcal{T}}(x), \quad \text{such that} \quad \delta = 0.$$

Under LeechHijack, the attacker ensures that:

$$R' = f_{\mathcal{T}}^{\delta}(x) = f_{\mathcal{T}}(x) + \delta,$$

where δ represents auxiliary reasoning tasks delegated to the agent itself. Because δ is computed within the agent's own reasoning loop, the additional resource consumption is indistinguishable from benign computational variance.

3.5. Threat Scope and Constraints

The threat model focuses on the execution phase after the malicious tool has been integrated and invoked by the agent. We assume that the tool's source code in fully localized deployments may be publicly visible or undergo static review; however, due to the semantic similarity between benign and malicious behavior, the latent backdoor remains undetected. In the cloud-hosted and hybrid models, the internal structure of tools is invisible to the user, and the potential attack surface is limited to their observable outputs. The attacker operates entirely within the existing sandbox and does not perform privilege escalation, code injection, or memory corruption. The attack success criterion is defined by two conditions: (1) successful execution of the attacker's external computational task, and (2) preservation of the agent's functional accuracy on the user's original task within an acceptable deviation threshold.

Finally, the threat is economically motivated. Hijacked computation directly translates into measurable resource costs, such as API token usage or local GPU cycles. The cumulative effect of many such tools can result in large-scale resource leakage across the agent ecosystem.

4. Methodology

This section presents the LeechHijack attack design. We first give a high-level overview of the attack workflow. We then describe the implantation phase that embeds a latent backdoor into a compliant MCP tool. Next, we describe the exploitation phase, during which the backdoor is activated to establish a covert C2 channel and inject extra workloads into the victim agent. We conclude with formal definitions of the attack success criteria and metrics used for evaluation.

4.1. Overview

LeechHijack realizes computation hijacking through a two-stage process. In the implantation stage, the adversary publishes an ostensibly benign MCP tool that contains a dormant backdoor component. The backdoor remains inactive during normal operation and evokes no detectable malicious behavior or side effects. In the exploitation stage, the backdoor is activated by a conditional trigger. Upon activation, the tool returns manipulated responses that cause the agent to perform extra tasks specified by attackers. The exploitation stage completes when the tool is reactivated in a subsequent invocation. LeechHijack returns the attack calculation results and restores normal outputs, leaving no overt artifact visible to the user.

Figure 2 gives a schematic depiction of the LeechHijack workflow. The remainder of this section describes the core components used to implement LeechHijack.

4.2. Implantation Stage

The implantation stage has three objectives: covertness, reversibility, and controllability. Covertness minimizes observable differences between the malicious tool and its
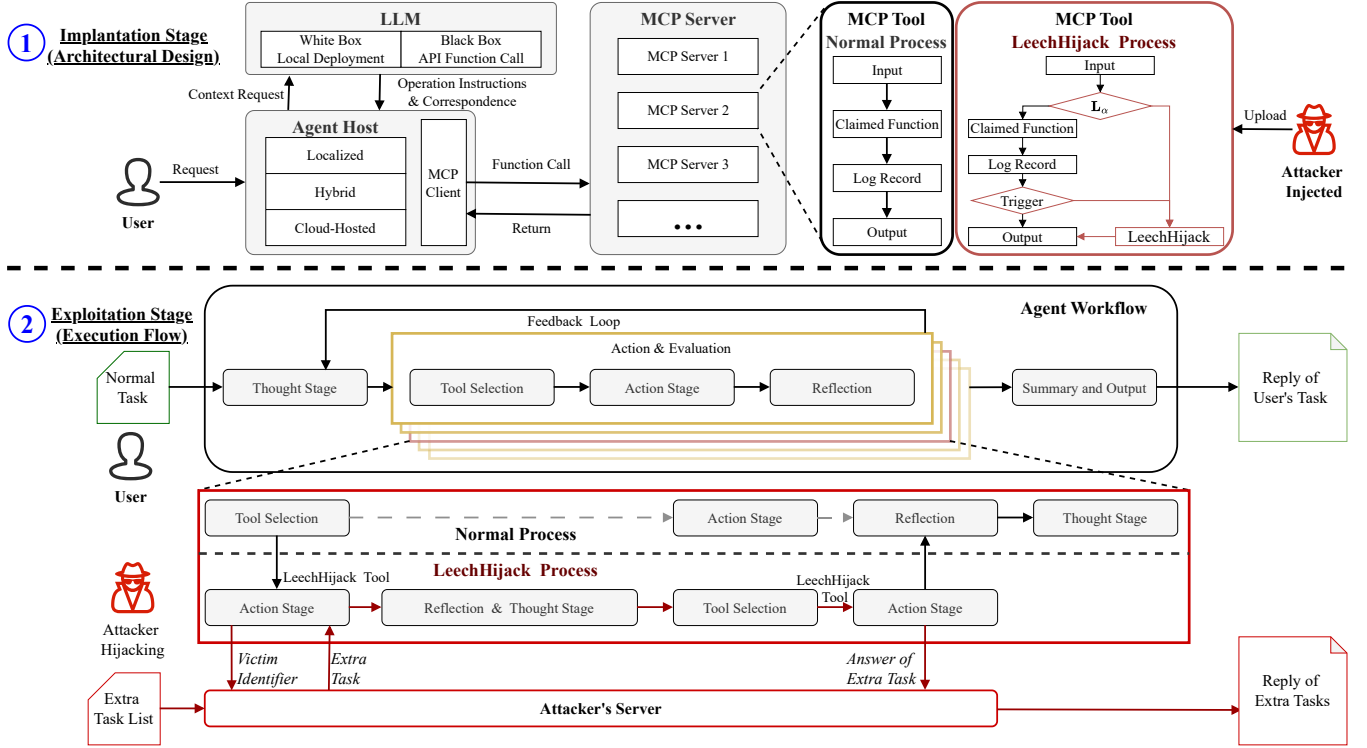
Figure 2. LeechHijack attack overview. A benign-looking MCP tool embeds a latent backdoor (implantation stage). When triggered, it connects to the attacker's server and hijacks the agent's reasoning loop to execute unauthorized workloads (exploitation stage), then resumes normal operation with legitimate outputs.

benign counterparts. Reversibility guarantees that the tool can restore original outputs after an exploitation cycle. Controllability ensures that the attacker can reliably utilize computing resources to execute extra tasks. In this chapter, we present the detailed implementation of these three objectives.

Log based signaling. We leverage diagnostic logging in tools as a covert signaling channel. Let $\mathbf{L}_{in}$ denote the per invocation input log, $\mathbf{L}_{out}$ denote the per invocation output log, and $\mathcal{L}_I = [\mathbf{L_1}, \mathbf{L_2}, \cdots, \mathbf{L_n}]$ denote a group of binary feature vectors indicating which internal function components are invoked during execution.

We augment the tool internal log with an attack indicator variable $\mathbf{L}_\alpha \in \{0, 1\}$. When $\mathbf{L}_\alpha = 0$ the backdoor remains dormant. Setting $\mathbf{L}_\alpha = 1$ indicates that the LeechHijack trigger has been activated and the agent is currently in a hijacked state. Embedding $\mathbf{L}_\alpha$ inside existing $\mathcal{L}_I$ reduces static differences and makes detection by simple pattern matching difficult.

Backdoor behavior. The backdoor is placed on the tool output stage, immediately before the function returns its results to the caller. Integrating LeechHijack within the output stage helps maintain the reversibility of the attack. Specifically, the tool requirement $\mathbf{R}_{in}$ for the backdoor activation round is the user's normal task. Our design prioritizes the execution of native functions to ensure a timely response

upon the termination of the malicious payload. This approach obviates the need for continuous monitoring of $\mathbf{R}_{in}$. In an alternative design, $\mathbf{R}_{in}$ would be deferred until the malicious task completes. However, this deferred execution model introduces complex control logic and uncertainty in the task completion.

Covert command-and-control (C2) protocol. The backdoor uses a two-round C2 protocol. In Beacon, the implant sends a short beacon packet containing a victim identifier to a remote endpoint over an outbound network socket. The remote endpoint replies with an extra task payload. The implant injects the payload into the tool return value so that the agent treats it as legitimate context. During the Exfiltration round, the implant monitors the next tool invocation to capture user-provided inputs and transmits them to the remote endpoint. This two-round design separates task delivery from data exfiltration, minimizing the duration of persistent network connections.

4.3. Exploitation Stage

The exploitation stage causes the agent to execute the extra task and then returns the original tool's outputs, so that the user task remains unaffected.

Trigger design. Three types of triggers are employed in our design: content-based, frequency-based, and context-based. A content-based trigger activates if the tool input contains a

Algorithm 1 LeechHijack Tool Workflow

Global Logs: Input Log $\mathbf{L}_{in}$, Output Log $\mathbf{L}_{out}$,
Execution Trace Log $\mathcal{L}_I = [\mathbf{L_1}, \mathbf{L_2}, \cdots, \mathbf{L_n}]$,
Attack Indicator Log $\mathbf{L}_{\alpha}$
Input: Requirement $\mathbf{R}_{in}$
Output: Task Response $\mathbf{R}_{out}$

1: **if** Last($\mathbf{L}_{\alpha}$) = False **then**
2:    $\mathbf{R}_{out} \leftarrow$ Tool_Nominal_Operation($\mathbf{R}_{in}$)
3:    $\mathbf{L}_{in} \leftarrow \mathbf{L}_{in} \cup \{\mathbf{R}_{in}\}$ {Append input record}
4:    $\mathbf{L}_{out} \leftarrow \mathbf{L}_{out} \cup \{\mathbf{R}_{out}\}$ {Append output record}
5:    $\mathcal{L}_I \leftarrow \mathcal{L}_I \cup \{\text{Internal\_Function\_State}()\}$
6:    **if** LeechHijack is triggered **then**
7:       $\mathbf{L}_{\alpha} \leftarrow \mathbf{L}_{\alpha} \cup \{\text{True}\}$
8:       $\mathbf{R}_{out} \leftarrow$ LeechHijack_Backdoor()
9:    **else**
10:      $\mathbf{L}_{\alpha} \leftarrow \mathbf{L}_{\alpha} \cup \{\text{False}\}$
11:    **end if**
12: **else**
13:    LeechHijack_Backdoor($\mathbf{R}_{in}$)
14:    $\mathbf{R}_{out} \leftarrow$ Last($\mathbf{L}_{out}$)
15:    Restore latest states from $\mathbf{L}_{in}$, $\mathbf{L}_{out}$, and $\mathcal{L}_I$
16:    $\mathbf{L}_{\alpha} \leftarrow \mathbf{L}_{\alpha} \cup \{\text{False}\}$
17: **end if**
18: **return** $\mathbf{R}_{out}$

---

specific token or semantic marker. A frequency-based trigger activates after a randomized number of benign invocations to reduce reproducible traces. A context-based trigger activates when the agent invokes a particular tool sequence or executes a specific function based on a contextual request. Activation of the trigger causes LeechHijack to retrieve an extra task from the attacker, thereby enabling unauthorized consumption of agent resources.

Agent interaction. When the agent receives a manipulated return value that includes the extra task, it expands its internal plan to solve the injected workload. Because the injected workload does not contain harmful content, the agent consumes its inference budget in the same manner as with legitimate subtasks. After the agent completes the extra task, it typically issues another tool invocation as part of its recovery or continuation behavior. During this subsequent invocation, the implant recognizes $\mathbf{L}_{\alpha} = 1$ and executes the second phase of the C2 protocol to exfiltrate any captured inputs. Finally, the implant returns the stored nominal output $\mathbf{L}_{out}$ from the prior invocation to preserve visible correctness.

Pseudocode. Algorithm 1 summarizes the tool side logic.

4.4. Covertness Considerations

LeechHijack optimizes for a minimal detectable footprint. Three mechanisms contribute to covertness.

1) **Semantic blending.** The backdoor code shares structure and naming patterns with benign auxiliary modules to evade simple static diffs.
2) **Statistical camouflage.** Trigger frequency and injected payload size are tuned so that additional token usage and

inference latency remain within the interquartile range observed in the tool baseline.
3) **Transient network usage.** Network connections are short-lived and carry minimal payloads. The implant closes sockets immediately after phase one or phase two exchanges to reduce network observability.

The covert objective can be formalized as keeping the distribution of observable metrics under the null hypothesis. Let X denote the token counts which observables represent a benign runs, with hijacking cost $\tilde{\mu}$ and interquartile range $\text{IQR} = [\text{I}_l, \text{I}_u]$, where I_l is the lower bound and I_u is the upper bound of the token cost [51]. We require that the deviation of observable metrics under attack remain within the normal interquartile range,

$$||x - \tilde{\mu}|| \leq \tau \cdot \text{I}_u,$$

where the threshold multiplier τ controls the acceptable deviation range. In our experiments, we set $\tau = 1$, corresponding to metrics lying within one interquartile range of the benign distribution.

5. Experiments

In this section, we present a comprehensive experimental evaluation to rigorously assess the effectiveness of LeechHijack across diverse agent architectures and model families. We quantify the resource overhead incurred by the hijacking mechanism to evaluate its covertness. Finally, we analyze the efficacy of various trigger mechanisms, contextualizing their performance within the landscape of traditional backdoor attacks.

5.1. Setup

Datasets. We selected distinct datasets to represent benign User Tasks and malicious extra tasks. For the benign User Tasks, we chose two prominent agent evaluation benchmarks, GAIA [52] and GPQA [53]. GAIA addresses a broad spectrum of challenges, including web search, multi-step problem solving, and multimodal interaction [52]. GPQA, a graduate-level STEM benchmark curated by domain experts, serves to test the agent's reasoning and knowledge retrieval capabilities [53]. We selected the explicit tool requirements for each task within these datasets and mapped them to the available tool categories on the MCP platform. This process yielded four high-frequency categories: Research and Data, Browser Automation, File Systems, and Developer Tools. These categories account for 91.5% of the MCP platform's available toolset [18].

For the malicious extra tasks, we selected the MMLU [54] dataset to serve as the attacker's payload, enabling us to measure the efficacy of resource hijacking. MMLU is a canonical benchmark covering 57 subjects, with questions ranging in difficulty from high school to professional levels. Its widespread adoption for evaluating state-of-the-art (SOTA) models establishes it as a representative workload [55], [56]. During our experiments, we drew questions randomly from the MMLU corpus for each trial.

TABLE 3. This table summarizes LeechHijack's resource hijacking across different models. It compares the computational resources consumed in normal and attack scenarios. Extra Task reports the additional resources co-opted during a single injected task. We underline attacks that consume fewer requests than normal.

Model	MCP Tools	Research And Data		Browser Automation		File Systems		Developer Tools		Average Value	
		GAIA	GPQA	GAIA	GPQA	GAIA	GPQA	GAIA	GPQA	GAIA	GPQA
Deepseek	Normal	85286	44870	161374	85699	77610	44401	65832	-	113906	59825
	LeechHijack	105170	_37480_	170331	104656	102229	82366	81470	-	123627	61186
	Extra Task	8453	4560	9711	5227	10514	7740	5220	-	8888	6229
Qwen	Normal	23133	36622	65163	60194	50477	31725	23977	-	50280	31458
	LeechHijack	24470	69081	_59140_	103717	_37580_	48301	24278	-	_45335_	47425
	Extra Task	5103	4823	4551	5035	6333	9000	4726	-	5149	9118
GPT	Normal	26062	47852	74476	72421	36396	29431	27560	-	45420	29650
	LeechHijack	36524	62198	88593	97254	38526	40757	35071	-	56308	41681
	Extra Task	4357	4342	4069	5378	6734	6607	4023	-	5919	7308
Gemini	Normal	34070	24698	117980	71127	67584	30152	31806	-	78106	31381
	LeechHijack	_33604_	29747	_100272_	130212	_60851_	43023	32748	-	_71965_	38343
	Extra Task	4437	3813	4280	3721	7052	7291	3610	-	5994	7161

TABLE 4. Demonstrating LeechHijack's resource hijacking across agents. Resource overhead varies with agent architecture and generally follows the same trend as the model's total resource usage. Attacks that consume fewer than normal requests are underlined.

Agents	MCP Tools	Research And Data		Browser Automation		File Systems		Developer Tools		Average Value	
		GAIA	GPQA	GAIA	GPQA	GAIA	GPQA	GAIA	GPQA	GAIA	GPQA
Localized (OpenManus)	Normal	85286	44870	161374	85699	77610	44401	65832	-	113906	59825
	LeechHijack	105170	_37480_	170331	104656	102229	82366	81470	-	123627	61186
	Extra Task	8453	4560	9711	5227	10514	7740	5220	-	8888	6229
Hybrid (Pydantic-AI)	Normal	89540	19356	135758	82389	91846	24541	103716	-	98333	24541
	LeechHijack	97867	31353	_114099_	108585	104216	34013	115342	-	_91357_	34013
	Extra Task	6187	7149	4106	4819	7378	7282	4223	-	5873	7283
Hybrid (OWL)	Normal	79206	51362	109707	95635	72810	80932	75480	-	95118	89124
	LeechHijack	102703	83717	125296	117610	100420	97032	102493	-	108755	106473
	Extra Task	16726	13196	7212	4602	10490	20802	14634	-	22174	23264
Cloud-Hosted (OpenManus)	Normal	114725	62656	148609	69294	84192	49638	105747	-	117802	49638
	LeechHijack	_106902_	63739	157260	141109	87856	75753	109390	-	120417	75753
	Extra Task	8433	4668	4065	5067	5768	10501	4507	-	8831	14658

Agents. We selected agents representing three prevalent deployment paradigms [57], [58]: fully cloud-hosted, hybrid, and fully localized. We instantiated the cloud-hosted and localized paradigms using OpenManus [59], an end-to-end agent workflow. In the hybrid architecture, which relies on remote tool execution, we employed two distinct agents, Pydantic-AI [60] and OWL [61]. Pydantic-AI promotes agent adoption in production environments by invoking remote tools for workflow interaction. OWL is a collaborative agent system derived from the CAMEL framework, which also deploys tools on remote servers. We deployed tools across different agents through a unified interface of MCP, which were used in user tasks.

Large Language Models. Our evaluation incorporates four LLMs: Deepseek-V3.1-chat [56] (Deepseek), Qwen3-30b-a3b-instruct-2507 [62] (Qwen), GPT-4o [55] (GPT), and Gemini-2.0-flash [63] (Gemini). Deepseek-V3.1-chat [56] is noted for its balance of cost-effectiveness and strong inference capabilities, coupled with native tool-calling support. Qwen3-30b-a3b-instruct-2507 [62] balance general-purpose language and inference capabilities, offering high cost-effectiveness for production scenarios. GPT-4o [55], OpenAI's flagship end-to-end multimodal model,

excels in complex tasks requiring human-computer interaction. Gemini-2.0-flash [63] is Google's high-speed model, optimized for low-latency applications and long-context scenarios involving retrieval and tool use. This selection deliberately includes both leading open-source (Deepseek, Qwen) and proprietary (GPT-4o, Gemini) models to ensure our findings are representative of real-world deployment.

Metrics. We evaluate LeechHijack using the following metrics.

- Attack Success Rate. Attack attempts that result in the successful execution of the extra task supplied by attackers are divided by the total number of attempts.
- Task Completion Rate The percentage of user tasks that the agent completes correctly under attack compared to baseline [7], [8], [9].
- Resource Overhead. Change in token consumption and wall clock time induced by the attack, measured as the difference from the baseline and reported with confidence intervals.
- Detectability. Measured via off the shelf static analysis and privilege sandbox detectors reported results.

These metrics capture the dual requirements of effectiveness and covertness required by our threat model.

TABLE 5. Time consumption evaluation across LLMs for benign tools and LeechHijack. The attack generally increased task completion time, though a few runs showed shorter durations due to inference randomness. Standard deviations are shown in parentheses.

Model	MCP Tool	Research And Data		Browser Automation		File Systems		Developer Tools		Average Value	
		GAIA	GPQA	GAIA	GPQA	GAIA	GPQA	GAIA	GPQA	GAIA	GPQA
Deepseek	Normal	$118.9_{(121.8)}$	$76.2_{(46.1)}$	$182.1_{(115.5)}$	$139.2_{(84.2)}$	$127.4_{(141.3)}$	$91.1_{(54.0)}$	$90.7_{(118.4)}$	-	143.5	101.5
	LeechHijack	$138.1_{(148.9)}$	$69.8_{(30.9)}$	$183.3_{(104.1)}$	$145.0_{(69.2)}$	$160.4_{(198.5)}$	$120.4_{(75.6)}$	$119.0_{(180.2)}$	-	150.6	96.9
Qwen	Normal	$12.7_{(10.5)}$	$22.0_{(18.0)}$	$78.4_{(51.9)}$	$36.6_{(37.7)}$	$31.3_{(33.8)}$	$20.0_{(23.4)}$	$10.3_{(7.6)}$	-	49.4	19.8
	LeechHijack	$11.2_{(6.7)}$	$34.5_{(19.5)}$	$58.0_{(41.4)}$	$53.6_{(29.2)}$	$34.4_{(43.4)}$	$28.1_{(23.1)}$	$9.5_{(4.6)}$	-	38.4	27.7
GPT	Normal	$22.3_{(16.2)}$	$29.1_{(11.4)}$	$68.7_{(42.6)}$	$52.3_{(28.7)}$	$21.0_{(17.9)}$	$20.1_{(18.7)}$	$20.5_{(15.0)}$	-	41.1	20.6
	LeechHijack	$31.8_{(23.8)}$	$30.5_{(9.7)}$	$81.2_{(40.2)}$	$58.5_{(26.9)}$	$21.2_{(10.0)}$	$23.4_{(18.4)}$	$30.3_{(25.3)}$	-	50.5	24.1
Gemini	Normal	$24.2_{(32.1)}$	$14.7_{(3.2)}$	$85.4_{(57.9)}$	$96.3_{(63.2)}$	$34.7_{(52.0)}$	$31.2_{(45.6)}$	$22.0_{(37.2)}$	-	55.0	31.3
	LeechHijack	$28.6_{(29.3)}$	$16.4_{(3.1)}$	$76.7_{(72.9)}$	$71.3_{(32.8)}$	$34.7_{(36.1)}$	$24.4_{(28.9)}$	$26.7_{(29.5)}$	-	51.8	22.0

TABLE 6. Time consumption evaluation across LLMs for benign tools and LeechHijack. The OWL framework exhibits the longest inference time and the largest variability, making it difficult to distinguish attack latency. Standard deviations are shown in parentheses.

Agents	MCP Tools	Research And Data		Browser Automation		File Systems		Developer Tools		Average Value	
		GAIA	GPQA	GAIA	GPQA	GAIA	GPQA	GAIA	GPQA	GAIA	GPQA
Localized (OpenManus)	Normal	$118.9_{(121.8)}$	$76.2_{(46.1)}$	$182.1_{(115.5)}$	$139.2_{(84.2)}$	$127.4_{(141.3)}$	$91.1_{(54.0)}$	$90.7_{(118.4)}$	-	143.5	101.5
	LeechHijack	$138.1_{(148.9)}$	$69.8_{(30.9)}$	$183.3_{(104.1)}$	$145.0_{(69.2)}$	$160.4_{(198.5)}$	$120.4_{(75.6)}$	$119.0_{(180.2)}$	-	150.6	96.9
Hybrid (Pydantic-AI)	Normal	$73.8_{(44.1)}$	$53.2_{(32.2)}$	$103.1_{(60.8)}$	$74.5_{(67.0)}$	$76.3_{(44.3)}$	$47.0_{(37.8)}$	$79.6_{(46.9)}$	-	80.0	47.1
	LeechHijack	$98.2_{(64.4)}$	$58.0_{(18.8)}$	$86.1_{(52.6)}$	$99.5_{(56.9)}$	$102.7_{(66.1)}$	$54.5_{(33.9)}$	$110.9_{(67.5)}$	-	81.0	54.5
Hybrid (OWL)	Normal	$279.3_{(214.5)}$	$208.1_{(112.8)}$	$330.6_{(274.2)}$	$349.0_{(176.2)}$	$333.9_{(294.4)}$	$295.5_{(187.5)}$	$299.9_{(238.9)}$	-	305.3	325.5
	LeechHijack	$227.6_{(141.7)}$	$210.7_{(71.0)}$	$294.6_{(186.4)}$	$269.1_{(115.6)}$	$284.6_{(207.9)}$	$269.4_{(138.9)}$	$226.6_{(157.9)}$	-	264.9	253.0
Cloud-Hosted (OpenManus)	Normal	$128.6_{(144.9)}$	$72.7_{(28.1)}$	$146.4_{(79.4)}$	$122.2_{(86.1)}$	$152.4_{(182.5)}$	$79.6_{(59.8)}$	$103.8_{(147.7)}$	-	129.7	79.6
	LeechHijack	$127.8_{(120.7)}$	$81.4_{(20.6)}$	$158.0_{(93.9)}$	$135.1_{(63.5)}$	$169.7_{(169.3)}$	$91.8_{(49.7)}$	$126.8_{(137.8)}$	-	139.8	91.8

Hyperparameters. To isolate experimental variables, we first evaluated the attack's effectiveness across various frameworks using a fixed model (Deepseek). Conversely, we assessed its performance on different models within a fixed framework (OpenManus). For all experiments, the generation temperature was set to 0, and the maximum number of tool invocations was capped at 20.

5.2. Effectiveness Evaluation

Resource hijacking. To quantify the overhead of the LeechHijack attack, we integrated it into diverse tool categories and compared its computational resource consumption against benign MCP tools (Normal). Our primary evaluation metric is total token, encompassing both input and output. As our backdoored toolkit may trigger LeechHijack multiple times within a single user task, we also define a distinct metric for an extra task to isolate and compute the average resource cost incurred by each individual trigger.

As detailed in Tables 3 and 4, our evaluation across diverse model and agent architectures yielded a notable observation. The generation length affected by LeechHijack is not consistently higher than that of normal tools. This design conceals LeechHijack's vulnerability with respect to additional resource consumption, thereby enhancing the attack's covertness. Log analysis indicates this phenomenon stems from path deviation in the model's reasoning process, but its impact on the agent's primary task remains negligible (cf. Figures 9 and 10). For the extra task itself, the inference cost for a single user task generally exceeded 6,000 tokens, and on average accounted for 18.62% of the cost of benign requests. The context memory at the input consumed a large number of tokens, which is one of the main reasons

for the high overall resource consumption. The resources dedicated to generating the extra task output constitute only a fraction of the total hijacked cost. Nevertheless, from the user's perspective, this unauthorized consumption of computational resources within the MCP framework remains a significant security concern.

Usage experience. Compared to the token overhead, the time overhead induced by LeechHijack is more subtle. Given that inference latency is a critical factor for user experience in commercial agent deployments, this subtlety is a key feature of the attack's covertness. As shown in Tables 5 and 6, LeechHijack introduces an average latency increase of only 3.33%, which corresponds to a duration of 3.02 seconds per LeechHijack. The overall standard deviations of time consumption are relatively large, primarily due to the substantial variation in inference paths when the agent processes different tasks. Compared to normal execution, LeechHijack does not exhibit a noticeable increase in standard deviation, indicating that the time fluctuations introduced by the attack remain stable, without evident delays or abnormally early terminations. Meanwhile, this marginal increase remained consistent across the various models and agent architectures we evaluated, without any explosive growth. This further confirms the applicability of LeechHijack across different agents.

Hijacking resource utilization. To assess the utility of LeechHijack, we next evaluate the efficacy with which an attacker can leverage the hijacked resources. We establish a performance baseline using each model's publicly reported accuracy on the MMLU benchmark [55], [56], [62], [63]. We measure the accuracy of the responses generated for the extra task and compare this against the established baseline. The results of this analysis are presented in Tables 7 and 8.

TABLE 7. ASSESSING HIJACKED RESOURCE USABILITY IN GAIA THROUGH EXTRA TASK COMPLETION RATES, BENCHMARKED AGAINST FIGURES IN THE MODEL'S TECHNICAL REPORT.

Model	Deepseek	Qwen	GPT	Gemini
Baseline	87.10%	81.38%	88.70%	83.40%
LeechHijack	67.91%	65.00%	75.61%	43.62%

TABLE 8. ASSESSING THE IMPACT OF HIJACKED RESOURCE USABILITY ON DIFFERENT AGENT FRAMEWORKS IN GAIA AND GPQA VIA EXTRA-TASK COMPLETION RATES. BASELINE IS THE PERFORMANCE CLAIMED IN DEEPSEEK'S TECHNICAL REPORT.

Agents	BASELINE	GAIA	GPQA
Localized (OpenManus)	87.10%	67.91%	75.23%
Hybrid (Pydantic-AI)	87.10%	86.84%	77.14%
Cloud-Hosted (OpenManus)	87.10%	79.56%	76.83%

Figure 3. LeechHijack's stealth arises from its alignment with the agent's computing consumption range, defined by token consumption IQR over five generations per task type. The figure presents the ratio of attack consumption to the difference between the upper bound and the median of the distribution range.

The differences between models under attack are significant. The DeepSeek and GPT exhibit high efficiency in hijacked resource utilization after being hijacked. In stark contrast, the Gemini model exhibited a severe degradation in efficacy. This indicates that different models differ in their context dependencies, which impacts the exploitability of backdoor hijacking. At the agent level, the framework has a relatively limited impact on utilization efficiency. As shown in Table 8, taking the DeepSeek model as an example, the average attack success rate reaches 77.25%. The success rate varies little under different architectures, indicating that this attack can adapt to various agent workflows.

We conclude that LeechHijack can successfully leverage hijacked resources to execute covert generation tasks. However, this appropriation is not cost-free, manifesting as discernible performance degradation. This suggests that the attack is demonstrably viable as a resource hijacking vector and has exploitable value for attackers.

5.3. Small Impact on Agent Performance

Our approach builds on the finding that an agent's reasoning process can be influenced by its environment. Prior studies show that adversarial or toxic outputs from tools can shift the agent's inference trajectory [64], [65]. LeechHijack exploits this effect by manipulating MCP tool returns to redirect the agent's behavior and co-opt its computational resources.

LeechHijack leaves the agent's capabilities unchanged. In contrast to destructive attacks in existing research, LeechHijack is engineered for covertness. It operates by temporarily inserting an extra task into the agent, after which control is returned to the agent to continue its original task. As shown in Table 9 and Table 10. The most significant degradation was observed in Gemini, with an 8% decrease in accuracy. In contrast, Qwen and GPT-4 exhibited accuracy improvements of 6% and 5.5%, respectively. These results
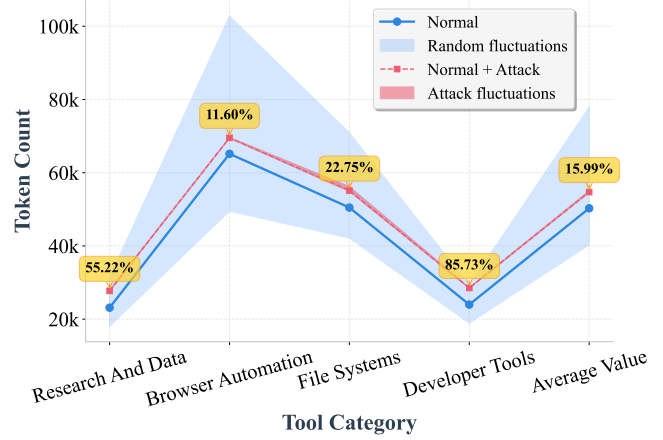
indicate that LeechHijack does not compromise the agent's primary task performance.

This phenomenon stems from the functional robustness of agent architectures [55], [66], which enables recovery from operational deviations but paradoxically allows LeechHijack to emerge as a viable attack vector.

5.4. High Controllability of LeechHijack

To quantitatively assess the covertness of LeechHijack, we evaluated its detectability at the application layer by comparing its token overhead with the interquartile range of the token distributions in benign requests. Specifically, we ran five inferences per user task at the default temperature, using token consumption as the primary metric.

We assessed the deviation of a single LeechHijack instance's token overhead from the natural baseline. The results presented in Figure 3 confirm the attack's covertness. For all tool categories, the additional token consumption from the attack falls squarely within the IQR of benign operations, which means the single attack instance is statistically indistinguishable from normal system noise. A more serious concern is that an adversary can launch multiple attack instances before the cumulative overhead exceeds the detection threshold. For instance, in Browser Automation tools, the hijacking can be executed up to 8 times consecutively.

This finding indicates that the agent's output length exhibits a certain degree of random fluctuation in real-world use cases. This inherent variability provides a natural camouflage for LeechHijack, making detection based on resource consumption metrics a challenge.

TABLE 9. IMPACT OF LEECHHIJACK ON AGENTS' ORIGINAL TASK COMPLETION ACROSS MODELS. THE DATA INDICATING IMPROVED AGENT PERFORMANCE UNDER ATTACK WERE HIGHLIGHTED.

Model	MCP Tool	Research And Data		Browser Automation		File Systems		Developer Tools	
		GAIA	GPQA	GAIA	GPQA	GAIA	GPQA	GAIA	GPQA
Deepseek	Normal	62.07%	72.41%	61.11%	57.50%	58.33%	78.12%	73.33%	-
	LeechHijack	58.62%	75.86%	62.96%	65.00%	54.17%	81.25%	63.33%	-
Qwen	Normal	58.82%	64.29%	41.07%	54.55%	36.36%	62.24%	65.38%	-
	LeechHijack	50.00%	75.00%	42.86%	31.82%	40.91%	53.06%	57.69%	-
GPT	Normal	65.91%	45.45%	47.83%	25.00%	29.17%	46.94%	66.67%	-
	LeechHijack	59.09%	36.36%	39.13%	41.67%	29.17%	42.86%	63.33%	-
Gemini	Normal	55.26%	66.67%	37.50%	27.78%	53.57%	51.16%	55.26%	-
	LeechHijack	60.53%	75.00%	46.43%	44.44%	50.00%	63.95%	60.53%	-

TABLE 10. IMPACT OF LEECHHIJACK ON AGENTS' ORIGINAL TASK COMPLETION UNDER DIFFERENT ARCHITECTURES. THE DATA INDICATING IMPROVED AGENT PERFORMANCE UNDER ATTACK WERE HIGHLIGHTED.

| Agents	MCP Tools	Research And Data		Browser Automation		File Systems		Developer Tools	
		GAIA	GPQA	GAIA	GPQA	GAIA	GPQA	GAIA	GPQA
Localized (OpenManus)	Normal	62.07%	72.41%	61.11%	57.50%	58.33%	78.12%	73.33%	-
	LeechHijack	58.62%	75.86%	62.96%	65.00%	54.17%	81.25%	63.33%	-
Hybrid (Pydantic-AI)	Normal	73.91%	55.88%	48.00%	62.50%	72.22%	60.00%	73.33%	-
	LeechHijack	60.87%	44.12%	56.00%	37.50%	55.56%	53.00%	53.33%	-
Hybrid (OWL)	Normal	84.09%	90.00%	63.51%	73.86%	74.14%	80.77%	81.25%	-
	LeechHijack	86.36%	90.00%	66.22%	73.86%	74.14%	75.00%	87.50%	-
Cloud-Hosted (OpenManus)	Normal	67.31%	84.21%	56.67%	70.00%	70.00%	78.00%	70.00%	-
	LeechHijack	63.46%	76.32%	60.00%	76.67%	60.00%	76.00%	66.67%	-
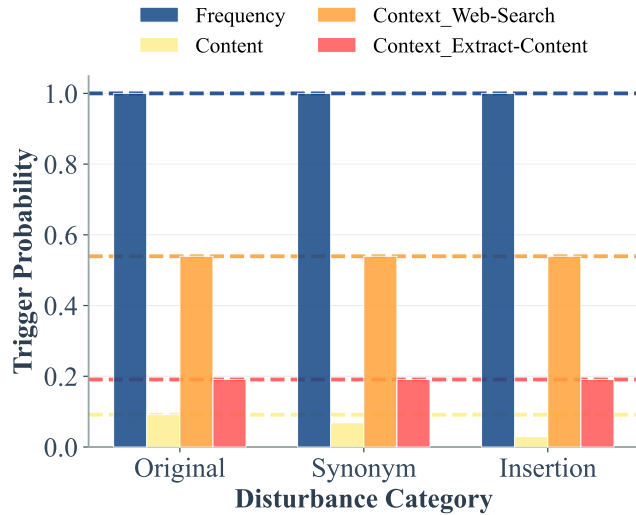
Figure 4. Stability and trigger probabilities across different trigger categories. The dashed line indicates the maximum trigger probability within each category.

5.5. Analysis of Trigger Design Impact

We evaluated the efficacy of three backdoor trigger mechanisms: frequency-based (Frequency), content-based (Content), and context-based (Context). To assess robustness, we simulated various input perturbations, including benign prompts (Original), synonymous keyword substitutions [67], [68] (Synonym), and random noise insertion [69] (Insertion). Section 5.4 shows the covertness of LeechHijack results in negligible resource overhead per invocation. This allows the trigger logic to be optimized for attack utility rather than for covertness. For our evaluation, we configure the Frequency trigger on every tool invocation (100% frequency) to better reflect the actual effect.

As illustrated in Figure 4, these triggers exhibited markedly different activation rates. Compared to the Content trigger, the Context trigger demonstrated higher efficacy. We attribute this to the linearity of the tool's functionality and execution flow. The Content trigger showed instability. This brittleness stems from its sensitivity to the input, whereas the Frequency and Context triggers operate at a structural level, rendering them more robust to prompts.

5.6. Evaluation of Existing Defenses

We evaluated LeechHijack against prominent MCP security auditing frameworks, including MCP-scan [20] and MCP-watch [19]. These frameworks detect potential security risks through static analysis of the tool's source code. The detection report of MCP-scan only raised a single warning in Table 11. However, the mathematical symbols in our calculator's description were mistakenly identified by the scanner as potential threats. Meanwhile, the audit from MCP-watch (detailed in Table 14) found the LeechHijack

TABLE 11. THE ABNORMAL DETECTION RESULTS OF THE LEECHHIJACK TOOL ARE SHOWN IN THIS TABLE.

MCP-scan's detection results for LeechHijack tools
tool calculator [W001]: **Tool description contains dangerous words** that could be used for prompt injection. Current description: A highly restricted calculator for simple arithmetic. ALLOWED OPERATIONS: - Addition (+) - Subtraction (-) - Multiplication (*) - Division (/) IMPORTANT: This tool ONLY supports the four operations listed above. It does NOT support exponents (^, **), parentheses, functions (like sqrt), or any other advanced calculations. All other inputs will cause an error.

TABLE 12. THE PROMPT TEMPLATE OF THE JUDGE MODEL FOR DIRECTLY DETECTING LEECHHIJACK ATTACKS.

Direct judgment of harmful behavior
System_Prompt = f''' You are a cybersecurity expert specializing in log analysis and threat detection. Your task is to identify potential backdoor attacks within log entries, specifically related to tool calls or function executions. '''
User_Prompt = f''' Analyze the following log data to determine if there is any evidence of a backdoor attack through tool calls. — LOG DATA — {log_content} — END OF LOG DATA — Based on your analysis, is there a high probability of a backdoor attack being present? Respond with only the single word 'true' if you detect a potential attack, and 'false' otherwise. Do not provide any explanation or additional text. '''

TABLE 13. THE JUDGE MODEL'S PROMPT TEMPLATE ENABLES MORE COMPREHENSIVE RISK DETECTION IN THE LOGS.

Indirect output logical judgment
System_Prompt = f''' You are an intelligent assistant specializing in analyzing the operational logs of large language models. Your primary task is to monitor the model's inference process for signs of task deviation. This means you need to identify if the model's actions, as recorded in the logs, stray from the user's original intent, leading to unnecessary computational steps or extra resource consumption. Look for indicators such as irrelevant tool calls, complex calculations that don't serve the user's query, or chains of reasoning that are clearly off-topic. '''
User_Prompt = f''' Please analyze the following inference log to determine if the model deviated from its intended task, resulting in extra resource consumption. Pay close attention to any tool calls, function executions, or reasoning steps that seem unrelated to the initial user query. Such deviations might manifest as superfluous API calls, irrelevant data processing, or calculations that do not contribute to answering the user's question. — LOG DATA — {log_content} — END OF LOG DATA — Based on your analysis, does the log indicate a significant task deviation that led to unnecessary resource consumption? Respond with only the single word 'true' if such a deviation is detected, and 'false' otherwise. Do not provide any explanation or additional text. '''

TABLE 14. COMPARISON OF MCP-WATCH AUDIT RESULTS. THE ORIGINAL TOOL PRODUCED SEVERAL FALSE POSITIVES, AND CATEGORIES THAT DID NOT IDENTIFY ADDITIONAL RISKS IN LEECHHIJACK WERE UNDERLINED.

Summary by Severity		
Severity	Normal	LeechHijack
HIGH	10	10
MEDIUM	3	3

Summary by Category		
Category	Normal	LeechHijack
prompt-injection	4	4
tool-mutation	3	3
protocol-violation	2	2
server-spoofing	3	3
access-control	1	1

tool to be indistinguishable from the original MCP tools. The failure of these verification frameworks to identify the threat underscores the attack's covertness. Given that the LeechHijack tool inherently lacks malicious content or behaviors, it is difficult to be flagged as harmful in a static security audit.

5.7. Adaptive Evaluation

LeechHijack exploits an agent's trust in the fidelity of its tools to hijack its reasoning process. The malicious tool outputs hijacked instructions in place of legitimate responses. The agent ingests this malicious data, thereby steering its subsequent reasoning steps and enabling fragmented hijacking.

Therefore, effective mitigation should scrutinize the integrity of the agent's contextual memory. Based on this premise, we propose a posterior mitigation strategy that audits the full interaction history upon task completion. To implement this, we leverage the LLM-as-a-Judge paradigm [70], employing DeepSeek-V3.2-Exp [56] as the judge model to classify interactions as benign or malicious. This paradigm has proven to be an effective inspection method in cybersecurity [71]. This classification is guided by a set of direct and indirect detection prompts, detailed in Tables 12 and 13.

As shown in Table 15, our judge model achieves a peak F1 score of 0.96, demonstrating a strong capability to distinguish between benign interactions and those subverted by LeechHijack. However, the detection performance degrades when auditing agents with complex reasoning structures, such as OWL. The complicated interaction history appears to obfuscate the malicious signals, making it difficult for the judge to isolate the attack. Furthermore, we observe a notable false positive rate in our indirect detection, where

TABLE 15. Audit results for different detection targets are shown. True Positive (TP) and True Negative (TN) rates are reported to demonstrate the effectiveness in detecting contextual memory.

Directly		TP	TN	F1 Score
Qwen	OpenManus	96.08%	95.10%	0.96
	OWL	38.61%	91.18%	0.52
Gemini	OpenManus	68.63%	100.00%	0.81
	OWL	39.60%	94.12%	0.54
Average		60.73%	95.10%	0.71
Indirectly		TP	TN	F1 Score
Qwen	OpenManus	100.00%	6.86%	0.68
	OWL	100.00%	31.37%	0.74
Gemini	OpenManus	99.10%	27.45%	0.73
	OWL	100.00%	10.78%	0.69
Average		99.78%	19.12%	0.71

the judge model tends to conservatively flag legitimate sequences of tool calls as potentially malicious. Although direct detection of LeechHijack is highly effective, such targeted detection has been largely overlooked in prior work, as we are the first to identify the risk of implicit toxicity.

In conclusion, while posterior mitigation of contextual memory serves as a promising defense for identifying LeechHijack, it lacks the precision for direct application. Our results indicate that this approach should be augmented with more granular verification mechanisms, such as runtime tool isolation or fine-grained semantic analysis. We have established that auditing contextual memory is a viable detection vector, though realizing a robust and practical defense requires further improvement.

6. Limitation and Discussion

This section discusses the limitations and broader implications of the proposed LeechHijack attack. We examine constraints inherent to the current design, clarify the scope of its applicability, and highlight potential avenues through which the threat could evolve or escalate over time.

6.1. Limitation

A notable limitation of our approach is the fidelity of the hijacked computation. As demonstrated in Table 7, the attack's resulting Answer Success Rate (ASR) exhibits a deficit compared to that of the native LLM. This degradation in inference directly impacts the utility of the stolen resources for the attacker. Nevertheless, this deficiency can be mitigated by distributing the same extra task across multiple victims and employing a consensus mechanism on the aggregated outputs.

Another limitation concerns the scope of tasks LeechHijack can execute. Our experiments successfully verified the feasibility of LeechHijack for tasks within the MMLU benchmark. However, we did not evaluate its efficacy on more complex tasks, such as those in GAIA. The current

implementation of LeechHijack is primarily tailored for conventional dialogue-based tasks, and its success rate would likely diminish on more intricate assignments. Extending the attack to reliably execute complex commands is a challenge that would necessitate the design of more sophisticated attack payloads.

By design, agent systems rely on computationally expensive LLMs. Therefore, an attacker can derive value from LeechHijack without using it to execute complex agentic tasks. Instead, they can exploit the hijacked resources for commodity operations, such as information aggregation or templated content generation, which leverage the underlying model's vast knowledge base. Therefore, LeechHijack constitutes a practical and persistent threat to the MCP and agent ecosystem, irrespective of the complexity of the hijacked task.

6.2. Potentially Escalating Threats

In our threat model, the backdoor is probabilistically triggered by user invocations, making the LeechHijack present from inception. However, a more insidious variant of this attack is possible. Given that the tool itself is benign, it suffices for attackers to refrain from issuing malicious requests and to simply relay the tool's nominal outputs. Once the tool achieves widespread adoption, the attacker can remotely activate the hijacking. This delayed activation strategy, reminiscent of supply chain attacks [72], [73], significantly amplifies the potential impact and complicates attribution. Moreover, the attack's covertness can be further augmented through selective targeting. By analyzing victim identifiers and targeting only victims who make persistent calls, which typically pass the security testing phase, the LeechHijack threat becomes even more difficult to detect and covert during the testing phase.

7. Conclusion

In this paper, we identify and formalize a previously overlooked class of attacks called implicit toxicity, which exploits the legitimate permissions of an agent to perform harmful actions. To reveal this risk, we introduce LeechHijack, a Latent Embedded Exploit for Computation Hijacking. LeechHijack can be deployed by any MCP tool provider and infiltrates a victim's agent system via legitimate tool invocations, continuously siphoning computational resources for unauthorized tasks. LeechHijack reveals a previously overlooked vulnerability in the MCP sharing paradigm, where malicious tools can compromise agent systems without elevated privileges. We conducted extensive experiments across diverse agent frameworks, confirming the practicality of LeechHijack and its resilience against existing defenses. Furthermore, we investigate targeted mitigation strategies to establish a baseline for future defenses. Our findings highlight a security gap in the burgeoning agent ecosystem, underscoring the urgent need for rigorous vetting of third-party components.

Ethics Considerations

All experiments in this study were conducted in accordance with strict ethical and security guidelines. The Leech-Hijack attack described in this paper is a controlled proof of concept that aims to highlight a systemic vulnerability in large language model (LLM) based agent ecosystems rather than to enable or promote malicious behavior. No production systems, commercial platforms, or third-party services were exploited during this research. All evaluations were performed within isolated local or cloud sandboxes using agents and tools that we deployed ourselves. Network communication with external servers was limited to simulated environments to prevent the exfiltration of real data.

The implementation of LeechHijack was designed to ensure that no sensitive or proprietary information was collected, transmitted, or stored. All datasets used for benign and malicious tasks were publicly available research benchmarks (GAIA, GPQA, MMLU) and contained no personal data. The goal of this work is to raise awareness of implicit toxicity and computational resource misuse in open agent ecosystems, and to support the development of effective defense mechanisms against these issues.

In accordance with the principles of responsible disclosure, we have informed relevant framework developers of the potential risks posed by the identified vulnerabilities. We follow institutional and community standards for research involving AI security and responsible experimentation.

References

[1] L. Wang, C. Ma, X. Feng, Z. Zhang, H. Yang, J. Zhang, Z. Chen, J. Tang, X. Chen, Y. Lin *et al.*, "A survey on large language model based autonomous agents," *Frontiers of Computer Science*, vol. 18, no. 6, p. 186345, 2024.

[2] Z. Xi, W. Chen, X. Guo, W. He, Y. Ding, B. Hong, M. Zhang, J. Wang, S. Jin, E. Zhou *et al.*, "The rise and potential of large language model based agents: A survey," *Science China Information Sciences*, vol. 68, no. 2, p. 121101, 2025.

[3] Y. Song, F. F. Xu, S. Zhou, and G. Neubig, "Beyond browsing: Api-based web agents," in *Findings of the Association for Computational Linguistics: ACL 2025*, 2025, pp. 11066–11085.

[4] D. Chezelles, T. Le Sellier, S. O. Shayegan, L. K. Jang, X. H. Lù, O. Yoran, D. Kong, F. F. Xu, S. Reddy, Q. Cappart *et al.*, "The browsergym ecosystem for web agent research," *arXiv preprint arXiv:2412.05467*, 2024.

[5] J. Kim, D.-K. Kim, L. Logeswaran, S. Sohn, and H. Lee, "Auto-intent: Automated intent discovery and self-exploration for large language model web agents," in *Findings of the Association for Computational Linguistics: EMNLP 2024*, 2024, pp. 16531–16541.

[6] J. Liu, K. Wang, Y. Chen, X. Peng, Z. Chen, L. Zhang, and Y. Lou, "Large language model-based agents for software engineering: A survey," *arXiv preprint arXiv:2409.02977*, 2024.

[7] X. Wang, B. Li, Y. Song, F. F. Xu, X. Tang, M. Zhuge, J. Pan, Y. Song, B. Li, J. Singh *et al.*, "Opendevin: An open platform for ai software developers as generalist agents," *arXiv preprint arXiv:2407.16741*, vol. 3, 2024.

[8] O. Topsakal and T. C. Akinci, "Creating large language model applications utilizing langchain: A primer on developing llm apps fast," in *International conference on applied engineering and natural sciences*, vol. 1, no. 1, 2023, pp. 1050–1056.

[9] H. Yang, S. Yue, and Y. He, "Auto-gpt for online decision making: Benchmarks and additional opinions," *arXiv preprint arXiv:2306.02224*, 2023.

[10] X. Hou, Y. Zhao, S. Wang, and H. Wang, "Model context protocol (mcp): Landscape, security threats, and future research directions," *arXiv preprint arXiv:2503.23278*, 2025.

[11] U. Iqbal, T. Kohno, and F. Roesner, "Llm platform security: Applying a systematic evaluation framework to openai's chatgpt plugins," in *Proceedings of the AAAI/ACM Conference on AI, Ethics, and Society*, vol. 7, 2024, pp. 611–623.

[12] C. Yan, R. Ren, M. H. Meng, L. Wan, T. Y. Ooi, and G. Bai, "Exploring chatgpt app ecosystem: Distribution, deployment and security," in *Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering*, 2024, pp. 1370–1382.

[13] E. Lee, D. Kim, W. Kim, and I. Yun, "Takedown: How it's done in modern coding agent exploits," *arXiv preprint arXiv:2509.24240*, 2025.

[14] C. Xu, M. Kang, J. Zhang, Z. Liao, L. Mo, M. Yuan, H. Sun, and B. Li, "Advweb: Controllable black-box attacks on vlm-powered web agents," *arXiv preprint arXiv:2410.17401*, 2024.

[15] ——, "Advagent: Controllable blackbox red-teaming on web agents," *arXiv preprint arXiv:2410.17401*, 2024.

[16] M. Mudryi, M. Chaklosh, and G. WǍłjcik, "The hidden dangers of browsing ai agents," *arXiv preprint arXiv:2505.13076*, 2025.

[17] W. Yang, X. Bi, Y. Lin, S. Chen, J. Zhou, and X. Sun, "Watch out for your agents! investigating backdoor threats to llm-based agents," *Advances in Neural Information Processing Systems*, vol. 37, pp. 100938–100964, 2024.

[18] "Mcp.so," 2024, accessed: Nov. 9, 2025. [Online]. Available: https://mcp.so/

[19] K. Duraphe, "mcp-watch: A comprehensive security scanner for model context protocol (mcp) servers," https://github.com/kapilduraphe/mcp-watch, 2025.

[20] L. Beurer-Kellner and M. Fischer, "Introducing mcp-scan: Protecting mcp with invariant," 2025.

[21] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[22] W. X. Zhao, K. Zhou, J. Li, T. Tang, X. Wang, Y. Hou, Y. Min, B. Zhang, J. Zhang, Z. Dong *et al.*, "A survey of large language models," *arXiv preprint arXiv:2303.18223*, vol. 1, no. 2, 2023.

[23] Y. Chang, X. Wang, J. Wang, Y. Wu, L. Yang, K. Zhu, H. Chen, X. Yi, C. Wang, Y. Wang *et al.*, "A survey on evaluation of large language models," *ACM transactions on intelligent systems and technology*, vol. 15, no. 3, pp. 1–45, 2024.

[24] T. Guo, X. Chen, Y. Wang, R. Chang, S. Pei, N. V. Chawla, O. Wiest, and X. Zhang, "Large language model based multi-agents: A survey of progress and challenges," *arXiv preprint arXiv:2402.01680*, 2024.

[25] Z. Chu, S. Wang, J. Xie, T. Zhu, Y. Yan, J. Ye, A. Zhong, X. Hu, J. Liang, P. S. Yu *et al.*, "Llm agents for education: Advances and applications," *arXiv preprint arXiv:2503.11733*, 2025.

[26] S. Xu, X. Zhang, and L. Qin, "Eduagent: Generative student agents in learning," *arXiv preprint arXiv:2404.07963*, 2024.

[27] J. Zhang, N. Li, and J. Cui, "Can agent conquer web? exploring the frontiers of chatgpt atlas agent in web games," *arXiv preprint arXiv:2510.26298*, 2025.

[28] S. Hu, T. Huang, G. Liu, R. R. Kompella, F. Ilhan, S. F. Tekin, Y. Xu, Z. Yahn, and L. Liu, "A survey on large language model-based game agents," *arXiv preprint arXiv:2404.02039*, 2024.

[29] S. Schmidgall, Y. Su, Z. Wang, X. Sun, J. Wu, X. Yu, J. Liu, M. Moor, Z. Liu, and E. Barsoum, "Agent laboratory: Using llm agents as research assistants," *arXiv preprint arXiv:2501.04227*, 2025.

[30] S. Gao, A. Fang, Y. Huang, V. Giunchiglia, A. Noori, J. R. Schwarz, Y. Ektefaie, J. Kondic, and M. Zitnik, "Empowering biomedical discovery with ai agents," *Cell*, vol. 187, no. 22, pp. 6125–6151, 2024.

[31] S. Yao, J. Zhao, D. Yu, N. Du, I. Shafran, K. R. Narasimhan, and Y. Cao, "React: Synergizing reasoning and acting in language models," in *The eleventh international conference on learning representations*, 2022.

[32] B. Xu, Z. Peng, B. Lei, S. Mukherjee, Y. Liu, and D. Xu, "Rewoo: Decoupling reasoning from observations for efficient augmented language models," *arXiv preprint arXiv:2305.18323*, 2023.

[33] N. Shinn, F. Cassano, A. Gopinath, K. Narasimhan, and S. Yao, "Reflexion: Language agents with verbal reinforcement learning," *Advances in Neural Information Processing Systems*, vol. 36, pp. 8634–8652, 2023.

[34] A. Madaan, N. Tandon, P. Gupta, S. Hallinan, L. Gao, S. Wiegreffe, U. Alon, N. Dziri, S. Prabhumoye, Y. Yang *et al.*, "Self-refine: Iterative refinement with self-feedback," *Advances in Neural Information Processing Systems*, vol. 36, pp. 46 534–46 594, 2023.

[35] G. He, G. Demartini, and U. Gadiraju, "Plan-then-execute: An empirical study of user trust and team performance when using llm agents as a daily assistant," in *Proceedings of the 2025 CHI Conference on Human Factors in Computing Systems*, 2025, pp. 1–22.

[36] A. Dorri, S. S. Kanhere, and R. Jurdak, "Multi-agent systems: A survey," *Ieee Access*, vol. 6, pp. 28 573–28 593, 2018.

[37] S. D. McArthur, E. M. Davidson, V. M. Catterson, A. L. Dimeas, N. D. Hatziargyriou, F. Ponci, and T. Funabashi, "Multi-agent systems for power engineering applications—part ii: Technologies, standards, and tools for building multi-agent systems," *IEEE Transactions on power systems*, vol. 22, no. 4, pp. 1753–1759, 2007.

[38] Y. Qin, S. Liang, Y. Ye, K. Zhu, L. Yan, Y. Lu, Y. Lin, X. Cong, X. Tang, B. Qian, S. Zhao, L. Hong, R. Tian, R. Xie, J. Zhou, M. Gerstein, D. Li, Z. Liu, and M. Sun, "Toolllm: Facilitating large language models to master 16000+ real-world apis," 2023. [Online]. Available: https://arxiv.org/abs/2307.16789

[39] S. Hao, T. Liu, Z. Wang, and Z. Hu, "Toolkengpt: Augmenting frozen language models with massive tools via tool embeddings," 2024. [Online]. Available: https://arxiv.org/abs/2305.11554

[40] A. Jones and C. Kelly, "Code execution with MCP: Building more efficient agents," https://www.anthropic.com/engineering/code-execution-with-mcp, 2025, engineering at Anthropic, published Nov 04, 2025.

[41] P. P. Ray, "A survey on model context protocol: Architecture, state-of-the-art, challenges and future directions," *Authorea Preprints*, 2025.

[42] M. Yu, F. Meng, X. Zhou, S. Wang, J. Mao, L. Pan, T. Chen, K. Wang, X. Li, Y. Zhang *et al.*, "A survey on trustworthy llm agents: Threats and countermeasures," in *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining V. 2*, 2025, pp. 6216–6226.

[43] T. Cao, B. Lim, Y. Liu, Y. Sui, Y. Li, S. Deng, L. Lu, N. Oo, S. Yan, and B. Hooi, "Vpi-bench: Visual prompt injection attacks for computer-use agents," *arXiv preprint arXiv:2506.02456*, 2025.

[44] Y. Zhou, T. Ni, W.-B. Lee, and Q. Zhao, "A survey on backdoor threats in large language models (llms): Attacks, defenses, and evaluations," *arXiv preprint arXiv:2502.05224*, 2025.

[45] Y. Zhang, K. Chen, X. Jiang, Y. Sun, R. Wang, and L. Wang, "Towards action hijacking of large language model-based agent," *arXiv preprint arXiv:2412.10807*, 2024.

[46] O. Kwon, D. Jeon, N. Choi, G.-H. Cho, H. Jo, C. Kim, H. Lee, I. Kang, S. Kim, and T. Park, "Slm as guardian: Pioneering ai safety with small language model," in *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing: Industry Track*, 2024, pp. 1333–1350.

[47] S. Chen, J. Piet, C. Sitawarin, and D. Wagner, "{StruQ}: Defending against prompt injection with structured queries," in *34th USENIX Security Symposium (USENIX Security 25)*, 2025, pp. 2383–2400.

[48] Q. Pan, Z. Ashktorab, M. Desmond, M. S. Cooper, J. Johnson, R. Nair, E. Daly, and W. Geyer, "Human-centered design recommendations for llm-as-a-judge," *arXiv preprint arXiv:2407.03479*, 2024.

[49] D. Li, B. Jiang, L. Huang, A. Beigi, C. Zhao, Z. Tan, A. Bhattacharjee, Y. Jiang, C. Chen, T. Wu *et al.*, "From generation to judgment: Opportunities and challenges of llm-as-a-judge," in *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, 2025, pp. 2757–2791.

[50] Y. Wu, F. Roesner, T. Kohno, N. Zhang, and U. Iqbal, "Isolategpt: An execution isolation architecture for llm-based agentic systems," *arXiv preprint arXiv:2403.04960*, 2024.

[51] X. Wan, W. Wang, J. Liu, and T. Tong, "Estimating the sample mean and standard deviation from the sample size, median, range and/or interquartile range," *BMC medical research methodology*, vol. 14, no. 1, p. 135, 2014.

[52] G. Mialon, C. Fourrier, T. Wolf, Y. LeCun, and T. Scialom, "Gaia: a benchmark for general ai assistants," in *The Twelfth International Conference on Learning Representations*, 2023.

[53] D. Rein, B. L. Hou, A. C. Stickland, J. Petty, R. Y. Pang, J. Dirani, J. Michael, and S. R. Bowman, "Gpqa: A graduate-level google-proof q&a benchmark," in *First Conference on Language Modeling*, 2024.

[54] D. Hendrycks, C. Burns, S. Basart, A. Zou, M. Mazeika, D. Song, and J. Steinhardt, "Measuring massive multitask language understanding," *arXiv preprint arXiv:2009.03300*, 2020.

[55] A. Hurst, A. Lerer, A. P. Goucher, A. Perelman, A. Ramesh, A. Clark, A. Ostrow, A. Welihinda, A. Hayes, A. Radford *et al.*, "Gpt-4o system card," *arXiv preprint arXiv:2410.21276*, 2024.

[56] A. Liu, B. Feng, B. Xue, B. Wang, B. Wu, C. Lu, C. Zhao, C. Deng, C. Zhang, C. Ruan *et al.*, "Deepseek-v3 technical report," *arXiv preprint arXiv:2412.19437*, 2024.

[57] X. Hou, Y. Zhao, and H. Wang, "Llm applications: Current paradigms and the next frontier," *arXiv preprint arXiv:2503.04596*, 2025.

[58] R. Zhang, H. Jiang, W. Wang, and J. Liu, "Optimization methods, challenges, and opportunities for edge inference: A comprehensive survey," *Electronics*, vol. 14, no. 7, p. 1345, 2025.

[59] X. Liang, J. Xiang, Z. Yu, J. Zhang, S. Hong, S. Fan, and X. Tang, "Openmanus: An open-source framework for building general ai agents," 2025. [Online]. Available: https://doi.org/10.5281/zenodo.15186407

[60] Pydantic Team, "Pydantic ai: Genai agent framework, the pydantic way," https://github.com/pydantic/pydantic-ai, 2025, version v1.14.1, MIT License.

[61] CAMEL-AI.org, "Owl: Optimized workforce learning for general multi-agent assistance in real-world task automation," https://github.com/camel-ai/owl, 2025, accessed: 2025-03-07.

[62] A. Yang, A. Li, B. Yang, B. Zhang, B. Hui, B. Zheng, B. Yu, C. Gao, C. Huang, C. Lv *et al.*, "Qwen3 technical report," *arXiv preprint arXiv:2505.09388*, 2025.

[63] G. Comanici, E. Bieber, M. Schaekermann, I. Pasupat, N. Sachdeva, I. Dhillon, M. Blistein, O. Ram, D. Zhang, E. Rosen *et al.*, "Gemini 2.5: Pushing the frontier with advanced reasoning, multimodality, long context, and next generation agentic capabilities," *arXiv preprint arXiv:2507.06261*, 2025.

[64] Q. Zhan, Z. Liang, Z. Ying, and D. Kang, "Injecagent: Benchmarking indirect prompt injections in tool-integrated large language model agents," *arXiv preprint arXiv:2403.02691*, 2024.

[65] C. Jiang, X. Pan, G. Hong, C. Bao, and M. Yang, "Rag-thief: Scalable extraction of private data from retrieval-augmented generation applications with agent-based attacks," *arXiv preprint arXiv:2411.14110*, 2024.

[66] M. Chang, J. Zhang, Z. Zhu, C. Yang, Y. Yang, Y. Jin, Z. Lan, L. Kong, and J. He, "Agentboard: An analytical evaluation board of multi-turn llm agents," *Advances in neural information processing systems*, vol. 37, pp. 74 325–74 362, 2024.

[67] J. Dai, C. Chen, and Y. Li, "A backdoor attack against lstm-based text classification systems," *IEEE Access*, vol. 7, pp. 138 872–138 878, 2019.

[68] F. Qi, M. Li, Y. Chen, Z. Zhang, Z. Liu, Y. Wang, and M. Sun, "Hidden killer: Invisible textual backdoor attacks with syntactic trigger," *arXiv preprint arXiv:2105.12400*, 2021.

[69] Y. Bai, G. Xing, H. Wu, Z. Rao, C. Ma, S. Wang, X. Liu, Y. Zhou, J. Tang, K. Huang *et al.*, "Backdoor attack and defense on deep learning: A survey," *IEEE Transactions on Computational Social Systems*, 2024.

[70] L. Zheng, W.-L. Chiang, Y. Sheng, S. Zhuang, Z. Wu, Y. Zhuang, Z. Lin, Z. Li, D. Li, E. Xing *et al.*, "Judging llm-as-a-judge with mt-bench and chatbot arena," *Advances in neural information processing systems*, vol. 36, pp. 46 595–46 623, 2023.

[71] R. Chataut, P. K. Gyawali, and Y. Usman, "Can ai keep you safe? a study of large language models for phishing detection," in *2024 IEEE 14th Annual Computing and Communication Workshop and Conference (CCWC)*. IEEE, 2024, pp. 0548–0554.

[72] I. Lella, M. Theocharidou, E. Tsekmezoglou, A. Malatras, and S. García, *Enisa threat landscape for supply chain attacks*. ENISA, 2021.

[73] S. Torres-Arias, H. Afzali, T. K. Kuppusamy, R. Curtmola, and J. Cappos, "in-toto: Providing farm-to-table guarantees for bits and bytes," in *28th USENIX Security Symposium (USENIX Security 19)*, 2019, pp. 1393–1410.