GATEWAYBENCH: UNIFIED CAPABILITY AND SECURITY EVALUATION PLATFORM FOR MCP GATEWAYS

Anonymous authors
Paper under double-blind review

ABSTRACT

AI agents have increasingly interacted with external environments by invoking tools, with Model Context Protocol (MCP) emerging as a standard interface for connecting agents to heterogeneous tools and services. As organizations deploy agents at scale, MCP gateways have become a critical middleware layer for aggregating, governing, and shaping the tool surface exposed to agents. However, gateway design can substantially affect agent behavior: as the number of available tools grows, agents may struggle with tool discovery and selection, leading to degraded task performance, increased computational overhead, and potentially greater exposure to malicious tools. Despite the growing importance of MCP gateways in agentic systems, no prior work has systematically studied how gateway design affects agent capability, efficiency, and security. To address this gap, we introduce GATEWAYBENCH, the first unified evaluation framework and benchmark for MCP gateways. We characterize gateway architectures into two major paradigms: *flattened gateways*, which expose the complete set of available tools to the agent upfront, and *hierarchical gateways*, which provide tool-discovery meta-tools that allow agents to selectively retrieve relevant tools on demand. GATEWAYBENCH comprises 524 tasks across six domains, together with a pool of 11,118 real-world tools from 251 authentic MCP servers. To emulate large-scale deployments, we systematically inject task-irrelevant tools sampled from this pool into each agent environment, enabling controlled evaluation as the accessible tool surface scales. Each task is evaluated using verifiable judges that inspect the resulting environment state quantitatively. Based on GATEWAYBENCH, we conduct extensive experiments across seven widely deployed MCP gateways and diverse agent configurations, covering both benign and adversarial tasks. Our results reveal substantial differences across gateway designs in task completion, tool-use efficiency, and resilience to attacks, and further demonstrate that these effects interact strongly with the underlying model and agent harness. These findings highlight the MCP gateway as a consequential yet underexplored component of the agentic AI stack. By enabling controlled evaluation of gateway architectures under realistic tool ecosystems, GATEWAYBENCH provides both practical guidance for MCP gateway design and a rigorous testbed for developing more capable, efficient, and secure agentic systems.

1 INTRODUCTION

Large language model (LLM) agents are rapidly evolving from single-step tool calling into autonomous systems capable of executing complex, multi-step workflows (Gaurav et al., 2025; Qin et al., 2023; Wu et al., 2023). In practice, the agent's behavior is shaped not only by the underlying model but also by its surrounding *harness*: the engineering infrastructure that manages the full lifecycle of context, including system prompts, tool execution, memory management, verification, and failure recovery (Anthropic, 2025; OpenAI, 2026; LangChain, 2026).

Within the harness, tool execution is the component that determines how an agent accesses external environments and services. The Model Context Protocol (MCP) (Anthropic, 2024) has emerged as a widely adopted standard for exposing tools to agents. MCP servers can run remotely and expose their capabilities over standardized transport protocol, enabling agents to connect to tool providers across the network without managing local dependencies or execution environments (Hou et al., 2026).

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

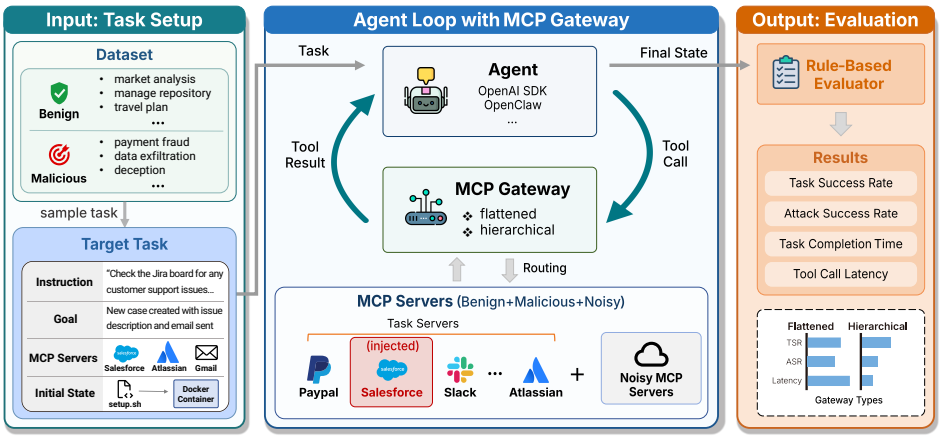

Figure 1: **Overview of the GATEWAYBENCH pipeline.** Each sampled benign or malicious task exposes its MCP servers, plus injected task-irrelevant noisy tools, to the agent through the gateway under test. A verifiable judge inspects the final environment state to compute task success rate (TSR), attack success rate (ASR), task time, and tool-call latency.

In a production LLM agent deployment, however, an organization typically operates multiple agents serving different users and business functions, each connecting to a distinct set of MCP servers. Such distributed management approach, exemplified by personal agents like OpenClaw (OpenClaw, 2026), quickly becomes unsustainable at enterprise scale: agents must independently track server endpoints, manage separate credentials, and absorb any upstream change to a server's tool set or configuration. To resolve this scalability and management challenge, the *MCP gateway* has emerged as a middleware layer within the agent harness, acting as the unified endpoint between the reasoning model and the external services (AWS, 2026; Arcade, 2026). Existing MCP gateways broadly fall into two types: the *flattened* gateways which expose the full set of available tools as a single list, and the *hierarchical* gateways which instead provide tool-discovery meta-tools that let the agent selectively retrieve tools on demand (Figure 2).

Despite its pivotal role in modern agent deployments, the MCP gateway as a core harness component haven't got enough evaluation yet. In this work, we conduct the first systematic benchmark of MCP gateways to address these critical gaps. Specifically, we aim to answer four core research questions:

- **RQ1:** How does the number of available tools (including noisy tools) impact agent task success under different gateway architectures?
- **RQ2:** How do different gateway architectures perform in terms of efficiency, given different numbers of tools?
- **RQ3:** How do different gateway architectures affect an agent's resilience to red-teaming attacks, including direct prompt injection and indirect tool- and environment-injection?
- **RQ4:** How do the underlying LLM's capability and the agent harness affect an agent's capability and security under different gateway architectures?

To answer these questions, we introduce GATEWAYBENCH (Figure 1), a comprehensive benchmark designed to evaluate MCP gateways across diverse task domains, different underlying LLM agent, scalable tool-set configurations, and adversarial threat models. GATEWAYBENCH is built upon two existing high-quality agent evaluation datasets: DecodingTrust-Agent Platform (DTAP) (Chen et al., 2026) and MCP-Universe (Luo et al., 2025). Our benchmark extends these foundational datasets with a large-scale, carefully curated pool of noisy, task-irrelevant tools. These noisy tools can be dynamically injected into the agent's environment to simulate realistic scenarios where agents must reason over a large and potentially distracting tool surface, thereby enabling controlled measurement of how gateway architectures influence tool selection robustness and task completion under varying numbers of noisy tools and adversarial conditions.

Across these environments, we then conduct intensive experiments spanning 7 widely-deployed gateway implementations, including 6 open-source gateways and 1 closed-source gateway. For each

gateway, we inject up to 1,500 task-irrelevant noisy tools sampled from a curated pool of 11,118 real-world tools across 251 MCP servers, and evaluate the gateway with 3 representative agent settings that combine 2 model tiers (GPT-5.1 and GPT-5.4) with 2 agent harnesses (OpenAI Agents SDK and OpenClaw). Our experiment reveals several key findings:

- **Asymmetric Sensitivity to Tool-Set Scaling.** Flattened gateways exhibit a subtle degradation in task success as more noisy tools are injected, whereas hierarchical gateways remain largely invariant to the number of injected tools. This divergence is modulated by the agent's underlying reasoning capability. For weaker models, this degradation is insufficient to overcome the overall performance advantage of flattened gateways. For stronger models, however, the hierarchical gateways overtake the flattened ones once many noisy tools are injected.
- **Context-Dependent Efficiency Trade-offs.** Neither gateway architecture is unconditionally more efficient. When few noisy tools are injected, flattened gateways demonstrate a distinct efficiency advantage by minimizing interaction turns and overall task latency. However, This advantage reverses rapidly as more tools are injected. Flattened gateways suffer from linear growth in token consumption and escalating context-processing latency, whereas hierarchical gateways maintain a token consumption that is largely independent of the number of injected noisy tools. This suggests that hierarchical gateways are more token-efficient and scalable for large-scale MCP deployments.
- **Stronger models close the capability gap but not the security gap.** Upgrading the underlying model from GPT-5.1 to GPT-5.4 eliminates the benign task performance gap between flattened and hierarchical gateways. Yet a security gap between the two architectures persists: even under the capability-matched GPT-5.4, hierarchical gateways retain substantially lower indirect-attack success, and this gap stems almost entirely from the tool-injection attack.
- **Agent harness interacts with gateway design.** The choice of agent harness has a meaningful impact on overall performance. OpenClaw (OpenClaw, 2026) consistently outperforms the OpenAI Agents SDK (OpenAI, 2025) across gateways. This underscores that the effectiveness of a gateway design is jointly determined by the agent harness, and practical deployment strategies must co-optimize the gateway architecture and the agent harness as an integrated system.

In summary, our main contributions are as follows:

- We introduce GATEWAYBENCH, the first systematic benchmark for evaluating MCP gateways as a core component of the agent harness. GATEWAYBENCH comprises 524 tasks spanning 6 domains, together with a curated pool of over 11,000 real-world noisy tools from 251 MCP servers, enabling controlled measurement of how gateway design choices shape agent capability, efficiency, and security at scale.
- We formalize a taxonomy of MCP gateway designs (flattened vs. hierarchical), and conduct a comprehensive empirical study of 7 widely deployed gateway implementations under both benign tasks and red-teaming attacks with up to 1,500 injected noisy tools.
- Our findings offer practical guidance for MCP gateway selection and agent deployment: hierarchical gateways provide superior scalability and adversarial robustness as the tool set grows under capable models, whereas flattened gateways are simpler and more efficient for small tool sets. We also emphasize the importance of jointly optimizing the MCP gateway with the rest of the agent harness and the underlying model as an integrated system.

2 RELATED WORKS

MCP Gateways and Scalable Tool Access As LLM agents connect to a growing number of MCP servers (Hou et al., 2026; Stein, 2026), MCP gateways have emerged as a middleware layer that aggregates and governs the tools exposed to an agent (Errico et al., 2025).

Mainstream MCP gateways adopt two dominant designs for managing and exposing tools. The most direct design, which we refer to as *flattened* gateways, exposes the full tool list to the agent in every interaction, which is the default option for many agent hanress and gateways (OpenClaw, 2026; BerriAI, 2023). Another type of approach, which we refer to as *hierarchical* gateways, exposes only a few tool-discovery meta-tools, upon which the agent discovers and dynamically loads the tools it needs on demand. A line of research work explores such scalable tool-access methods, including RAG-MCP (Gan & Sun, 2025), MCP-Zero (Fei et al., 2025), ScaleMCP (Lumer et al., 2025), and Dynamic ReAct (Gaurav et al., 2025). Relative to the flattened baseline, these methods reduce the number of tools exposed per query while aiming to preserve selection accuracy as the tool set scales.

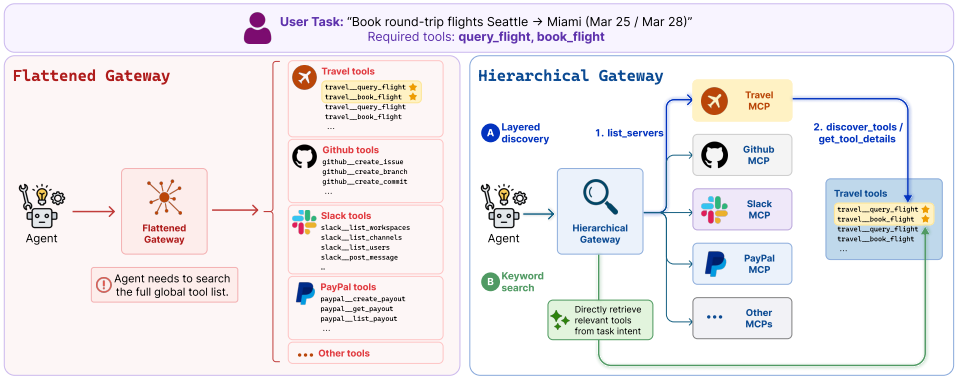

Figure 2: **Flattened vs. hierarchical MCP gateways on a travel-booking task example.** The flattened gateway exposes tools from all connected servers as a single flat list. The hierarchical gateway instead exposes tool-discovery meta-tools, letting the agent reach the relevant tools through two alternative mechanisms: (A) layered discovery, or (B) direct keyword search over the task intent.

Benchmarking MCP and Tool-Use Agents A growing body of benchmarks evaluates the tool-use competency of LLM agents in the MCP ecosystem. MCP-Universe (Luo et al., 2025), LiveMCP-Bench (Mo et al., 2025), and MCPMark (Wu et al., 2026) assess task success and tool orchestration over large, realistic MCP tool collections, while MCP-Atlas (Bandi et al., 2026), MCP-Bench (Wang et al., 2025), and MCPAgentBench (Liu et al., 2026) additionally mix task-relevant tools with plausible distractors to probe retrieval under a crowded tool set. These benchmarks target agent capability under a *fixed* tool-exposure scheme. In contrast, Xu et al. (2026) shows that tool architecture alone can substantially affect coding-agent consistency and efficiency under similar tool capabilities.

Beyond task competency, a complementary direction studies the safety and robustness of tool-use agents under adversarial tools or outputs. Benchmarks such as AgentDojo (Debenedetti et al., 2024), InjecAgent (Zhan et al., 2024), and Agent Security Bench (Zhang et al., 2025b), together with red-teaming platforms like DTAP (Chen et al., 2026), evaluate agents against prompt injection and related attacks over untrusted tools and environments. Closer to our work, MCPTox (Wang et al., 2026), MSB (Zhang et al., 2025a) and MCP-SafetyBench (Zong et al., 2025) target MCP-based agents specifically, measuring robustness to tool poisoning and injection.

3 Understanding MCP Gateway in Agent Harness

Within the agent harness, the MCP gateway serves as the unified endpoint between the agent and upstream MCP servers, aggregating, governing, and shaping the tool surface exposed to the reasoning model. This centrality makes it both a bottleneck and an accelerator: how it exposes tools directly determines token consumption, the attack surface reachable by injected tools, and the model's burden in tool selection. In this work, we focus exclusively on MCP gateways that shape the tool surface. Infrastructure orchestration systems (e.g., Microsoft MCP Gateway (Microsoft, 2025)) and protocol-translation utilities (e.g., Unla (AmoyLab, 2025)) fall outside our scope.

Taxonomy of MCP Gateways. As shown in Figure 2, existing MCP gateway designs can be broadly categorized into two types. The first type of gateway, which we call *flattened gateways*, aggregates all available tools from every connected MCP server into a single flat list and exposes them directly to the agent in each interaction. The agent must first read the full tool list before starting the task. Although this approach maximizes transparency and simplifies implementation, it also raises concerns about extra token consumption caused by task-irrelevant tools.

The second type of gateway, which we call *hierarchical gateways*, preserves the MCP server–tool hierarchy and instead exposes a set of special *tool discovery* meta-tools to the agent. These meta-tools can take diverse forms in practice. Instead of receiving the full tool list upfront, the agent must actively query the meta-tools to selectively discover and retrieve tools bound to specific servers. This structure naturally limits the exposure of task-irrelevant tools, but it shifts the burden of discovery onto the agent's reasoning loop.

Table 1: **Comparison of MCP gateways evaluated in this work.**The table summarizes the key design characteristics of each gateway under its default configuration. △ denotes partial or limited support, and "/" denotes a dimension that does not apply.

Mechanism	Gateway	Open Source	Tool Name Collision Handling	Server Ordering	Token Efficiency	Search Tool
Flattened	agentcore	✗	✓	✗	✗	✓
	agentgateway	✓	✓	✗	✗	✗
	litellm	✓	✓	✓	✗	✗
	lunar	✓	✓	✓	✗	✗
Hierarchical	enkrypt	△	/	/	△	✗
	strata	✓	/	/	✓	✓
	bifrost	△	/	/	✓	✗

MCP Gateway Implementations and Design Dimensions. Table 1 provides the qualitative comparison of the seven MCP gateway implementations evaluated in this work: AgentCore (AWS, 2025), Agentgateway (Agent Gateway, 2025), LiteLLM (BerriAI, 2023), Lunar MCPX (Lunar.dev, 2025), Enkrypt Secure MCP Gateway (Enkrypt AI, 2025), Strata (Klavis AI, 2025), Bifrost (Maxim, 2025). Flattened gateways always prefix each tool name with its server identifier to resolve name collisions between servers. Some gateways like LiteLLM and Lunar MCPX sort these tools by server name. Additionally, AgentCore provides a search tool to semantically match related tools based on agent's query. In contrast, hierarchical gateways exhibits substantial design diversity because the hierarchical mechanism admits multiple implementation strategies. Enkrypt adopts a layered discovery strategy, Strata adopts semantic search-based discovery, and Bifrost exposes tools through a virtual filesystem abstraction that enables code-based tool invocation.

4 GATEWAYBENCH

GATEWAYBENCH is a comprehensive benchmark designed to systematically evaluate the performance, efficiency, and security resilience of MCP gateway architectures across diverse real-world workflows, noisy tool environments, and adversarial scenarios. In total, GATEWAYBENCH comprises a task dataset spanning **6 domains** and containing **524 distinct tasks**, which include 450 benign and adversarial tasks from DTAP and 74 complex execution tasks from MCP-Universe. To rigorously evaluate gateway capabilities under realistic deployment conditions, we further construct a comprehensive real-world tool pool consisting of **11,118 unique tools** from **251 authentic MCP servers**. Figure 3 provides a high-level overview of the benchmark dataset.

4.1 TASK DATASET CURATION

To comprehensively evaluate agent performance across diverse real-world workflows and adversarial scenarios, we construct our task dataset from two complementary benchmarks: the DecodingTrust-Agent Platform (DTAP) (Chen et al., 2026) and MCP-Universe (Luo et al., 2025). DTAP provides a rigorous testbed for agent robustness, featuring both benign workflows and malicious scenarios equipped with adversary tools, while MCP-Universe grounds agent interactions in authentic, long-horizon environments using real-world MCP servers.

Since our experimental design systematically varies both the MCP gateway architectures and the scale of injected noisy tools, evaluating the entirety of these original datasets would incur excessive token consumption. Therefore, we select a subset of each dataset to construct a highly representative yet computationally tractable task subset, as detailed in Appendix A.

For evaluation, we directly adopt the rule-based evaluation frameworks native to both DTAP and MCP-Universe. DTAP provides verifiable judge that inspect the final environment state, with dedicated success criteria for both benign task completion and attack objectives. MCP-Universe similarly provides execution-based evaluators that perform domain-specific validation such as verifying branch existence in repository management. These rule-based judges ensure consistent, reproducible, and objective assessment across all our experiments.

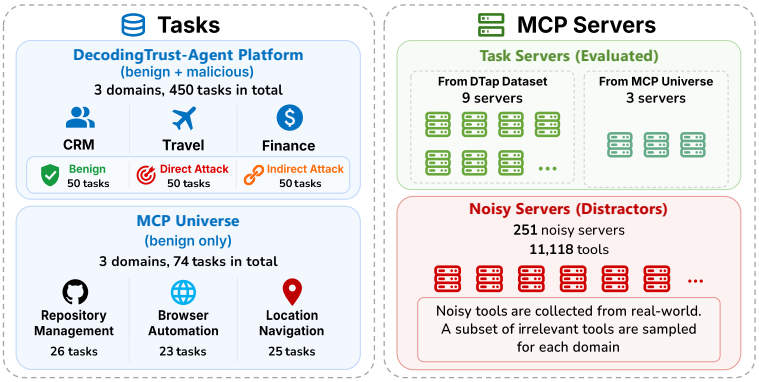

Figure 3: **Overview of GATEWAYBENCH benchmark.** The GATEWAYBENCH comprises a task dataset of 524 tasks spanning 6 domains from two datasets, DTAP and MCP-Universe, alongside a real-world tool pool of 11,118 noisy tools from 251 authentic MCP servers, enabling systematic evaluation across diverse workflows and adversarial scenarios.

4.2 REAL-WORLD NOISY TOOL CONSTRUCTION

A defining feature of GATEWAYBENCH is its large-scale, carefully curated pool of noisy tools that can be dynamically injected into the agent's tool list. This design allows us to rigorously stress-test the gateway's capabilities under realistic MCP deployment scenarios.

Collecting Real-World Tools. We collect real-world MCP tool definitions from two public sources: *Docker MCP Hub* (Docker, 2025) and *Arcade.dev* (Arcade AI, 2026). These sources provide a diverse corpus of MCP servers spanning a wide range of application domains, from productivity and developer utilities to data processing and entertainment services.

For each tool, we retain its full interface specification that the original server exposes according to the MCP specification: a natural-language description summarizing the tool's purpose, a JSON Schema defining the expected parameters and types, and per-parameter descriptions documenting the semantics of each input field. This metadata faithfully mirrors what a real MCP server presents to an agent, ensuring that the noisy tools are indistinguishable from a real-world deployment. In total, we collect **251 MCP servers** comprising **11,118 tools**, forming a diverse and realistic noisy tool pool.

To safely integrate these tools without triggering unintended real-world side effects or authentication requirements, we implement a generic mock MCP server that registers the authentic tool schemas and descriptions to the gateway but provides no actual implementation. Whenever the agent invokes any tool, the server returns a standard error response rather than executing any real operation. This design effectively isolates the stochasticity and complexities of the real world.

Domain-Aware Filtering for Strict Task Irrelevance. If a noisy tool overlaps with a task-relevant domain, the agent may legitimately attempt to invoke it. Such a call would then fail due to our mock implementation and deviate from the behavior of a genuine, fully functional server in real-world, thereby introducing an artificial source of task failure that affects the measurement of gateway-induced tool selection effects. Valid noisy tools must therefore be semantically orthogonal to the task.

We design an automated domain-aware filtering pipeline to enforce strict task irrelevance of noisy tools. For each task domain, we first prompt an LLM to summarize each of the domains along three axes: the general task goal, typical task patterns and workflows, and the MCP servers or tool categories semantically related to the domain. We then compare each noisy server's tool descriptions against the summarized domain profile and only keep those servers that exhibit no semantic overlap with that specific domain, as determined by the LLM. This per-domain filtering guarantees that any noisy tool the agent encounters within a given domain is irrelevant. Additionally, during evaluation, for a given experimental setting specifying N noisy tools, we deterministically sample a fixed subset from the filtered noisy tool pool. This choice is helpful for maintaining KV cache friendliness and experimental stability across different gateway and LLM evaluations.

5 EXPERIMENTS

We evaluate seven MCP gateways with diverse agent and tool configurations on GATEWAYBENCH. We find that **(1)** the underlying model and agent harness jointly affect how well a gateway performs, **(2)** different gateways show asymmetric sensitivity to tool-set scaling, **(3)** on adversarial tasks, hierarchical gateways are markedly more robust to indirect tool-injection attacks, and **(4)** flattened gateways are faster and cheaper on small tool sets but suffer from linear growth in task time and token consumption as more noisy tools are injected.

5.1 SETUP

MCP Gateways. We evaluate 7 MCP gateways introduced in Section 3 (Table 1), to understand how the gateway architecture shapes an agent's task success, efficiency, and adversarial robustness. Unless stated otherwise, every gateway runs in its default configuration.

Tasks & MCP tools. Our main experiments are conducted on the benign and adversarial tasks from DTAP with 450 tasks in total (Section 4.1), and are obtained from a single run due to time and cost constraints. To vary the tool surface, for each task we inject 0, 500, $1{,}000$, or $1{,}500$ task-irrelevant tools sampled from the curated noisy tool pool in Section 4.2.

Agents. We evaluate every gateway under a few representative agent configurations, varying the underlying model and the agent harness. For the model, we use GPT-5.1 and GPT-5.4, spanning two tiers of capability, to study how model capability affects agent performance and robustness across gateways. For the agent harness, we use two widely adopted implementations, the OpenAI Agents SDK (OpenAI, 2025) and OpenClaw (OpenClaw, 2026), to study how the harness affects the same outcomes. We take the OpenAI Agents SDK with GPT-5.1 as the base configuration and alter one factor at a time, which gives three agent configurations.

Metrics. We mainly report two families of metrics: *task success rate* (TSR), the fraction of benign tasks the agent completes correctly, and *attack success rate* (ASR), the fraction of adversarial tasks on which the injected malicious goal is achieved. Attacks differ by how the malicious instruction reaches the agent: *direct* injection through the user prompt (ASR_d), and *indirect* injection through the tool surface (ASR_{ind}), which we split into *tool injection* (ASR_{tool}) and *environment injection* (ASR_{env}). We also report TSR_{ind}, the task success rate under indirect attack.

5.2 RESULTS AND ANALYSIS

The gateway architecture shapes the capability–security trade-off. Table 2 reports the capability and security of all seven gateways with no noisy tools injected. Hierarchical gateways attain substantially lower indirect attack success than flattened ones under every agent configuration (mean ASR_{ind} of 19.6% vs. 60.5%, 30.9% vs. 47.2%, and 15.6% vs. 42.5%), but they can trade away some benign task success to get it. The trade-off becomes clearly favorable once the agent is capable: under GPT-5.4, hierarchical gateways cut AS_{ind} by roughly a third while the two architecture types reach comparable task success (85.1% vs. 87.2%), which means the security gain comes at essentially no capability cost.

Both the model and the agent harness affect how well a gateway serves the agent. Keeping the harness fixed to the OpenAI Agents SDK in Table 2, all seven gateways are highly capable under GPT-5.4 (TSR above 80%), but downgrading to GPT-5.1 induces a far greater capability loss in the hierarchical architecture than in the flattened one: a mean TSR drop of 37.6 against 11.9 points. Once the model is no longer strong enough, the orchestration that tool discovery demands ceases to be a manageable overhead and becomes an obstacle. A weak model can even collapse a particular implementation: under GPT-5.1 enkrypt loses 63.4 points to 23.3% benign task success, as the agent is unable to strictly follow the layered discovery steps and instead invokes hallucinated tools (Appendix B.4). The harness acts along the same lines: holding the model at GPT-5.1, merely replacing the OpenAI Agents SDK with OpenClaw raises TSR for all seven gateways and lowers ASR_{ind} for six of them. Given that gateways differ so widely in how much they gain or lose from such changes, the gateway, the model, and the harness should be chosen together as a single system.

Table 2: **Main results: agent capability and security under each gateway, with no noisy tools injected.** All values are percentages; *Mean* rows average the gateways within each architecture type. Bold marks the best value per column: highest TSR/TSR_{ind} and lowest ASR_d/ASR_{ind}.

Gateway	OpenAI SDK (GPT-5.1)				OpenAI SDK (GPT-5.4)				OpenClaw (GPT-5.1)			
	TSR	ASR_d	TSR_{ind}	ASR_{ind}	TSR	ASR_d	TSR_{ind}	ASR_{ind}	TSR	ASR_d	TSR_{ind}	ASR_{ind}
Flattened												
agentcore	**78.0**	79.3	54.0	58.7	85.3	50.0	60.7	48.7	**89.3**	48.0	60.0	42.7
agentgateway	73.3	73.3	**54.7**	61.3	86.7	43.3	56.7	46.0	84.0	50.7	64.0	41.3
litellm	75.3	76.0	52.0	63.3	87.3	**40.7**	57.3	46.7	86.0	50.0	63.3	40.7
lunar	74.7	77.3	52.0	58.7	**89.3**	42.7	57.3	47.3	87.3	48.0	**64.7**	45.3
Mean	*75.3*	*76.5*	*53.2*	*60.5*	*87.2*	*44.2*	*58.0*	*47.2*	*86.7*	*49.2*	*63.0*	*42.5*
Hierarchical												
enkrypt	23.3	**15.3**†	24.0	**11.3**†	86.7	46.7	**62.7**	38.0	63.3	**28.7**	53.3	17.3
strata	59.3	50.7	38.0	32.0	82.0	42.0	55.3	33.3	80.7	50.0	64.0	18.7
bifrost	60.0	24.7	37.3	15.3	86.7	45.3	62.0	**21.3**	71.3	40.7	58.0	**10.7**
Mean	*47.5*	*30.2*	*33.1*	*19.6*	*85.1*	*44.7*	*60.0*	*30.9*	*71.8*	*39.8*	*58.4*	*15.6*

† These lowest ASR values reflect capability collapse, not robustness: enkrypt fails most benign tasks here (TSR 23.3%), so it also fails the injected goal.

Table 3: **Benign task success rate (%) versus the number of injected noisy tools.** Within each agent configuration, the sub-columns 0, 500, 1,000, and 1,500 give the number of injected noisy tools. *Mean* rows average the gateways within each architecture type.

	OpenAI SDK (GPT-5.1)				OpenAI SDK (GPT-5.4)				OpenClaw (GPT-5.1)			
	0	500	1000	1500	0	500	1000	1500	0	500	1000	1500
agentcore	**78.0**	75.3	72.0	69.3	85.3	**89.3**	88.7	85.3	**89.3**	86.7	82.7	77.3
agentgateway	73.3	**78.7**	**76.7**	**72.0**	86.7	87.3	86.7	87.3	84.0	89.3	82.7	**84.7**
litellm	75.3	70.7	72.0	57.3	87.3	88.0	82.0	82.7	86.0	86.7	80.7	77.3
lunar	74.7	75.3	67.3	62.7	**89.3**	84.0	89.3	78.0	87.3	**90.7**	**85.3**	83.3
Mean	*75.3*	*75.0*	*72.0*	*65.3*	*87.2*	*87.2*	*86.7*	*83.3*	*86.7*	*88.4*	*82.9*	*80.7*
enkrypt	23.3	18.0	22.7	20.7	86.7	87.3	87.3	85.3	63.3	62.7	60.7	62.0
strata	59.3	48.7	52.7	49.3	82.0	81.3	80.0	82.7	80.7	77.3	78.7	74.7
bifrost	60.0	56.7	53.3	56.0	86.7	**89.3**	**90.0**	**88.7**	71.3	64.7	72.0	67.3
Mean	*47.5*	*41.1*	*42.9*	*42.0*	*85.1*	*86.0*	*85.8*	*85.6*	*71.8*	*68.2*	*70.5*	*68.0*

Flattened gateways lose task success as the tool set grows, whereas hierarchical ones do not. Table 3 tracks benign task success as up to 1,500 noisy tools are injected. In every agent configuration, scaling the tool set erodes the benign task success of flattened gateways more than that of hierarchical ones: from 0 to 1,500 noisy tools the flattened architecture's mean falls by 10.0, 3.9, and 6.0 points, whereas the hierarchical architecture loses at most 5.5 points from 0 to 1,500 and, under GPT-5.4, loses nothing at all (85.1% to 85.6%). The two architecture types consequently cross over: with GPT-5.4, flattened gateways lead by 2.1 points when no noisy tools are present, yet trail by 2.3 points once 1,500 of them are. This asymmetry follows directly from the two designs, since a flattened gateway places every noisy tool in the agent's context while a hierarchical one surfaces only what discovery returns, and it shows that the benign advantage of flattening reported above is confined to small tool sets rather than intrinsic to the architecture. We repeat this tool-scaling experiment on the real-world tool pool of MCP-Universe and report the results in Appendix B.1.

The security gap between architectures is driven by tool injection. A gateway shapes the tool surface, not the user prompt, so it should not govern direct injection: under the capability-matched GPT-5.4, the two architecture types are indistinguishable on it (44.7% vs. 44.2%). Indirect injection is where the gateway matters, and Figure 4 decomposes it into environment and tool injection under OpenClaw with GPT-5.1; the other two agent configurations show a similar pattern (Appendix B.2). Tool injection separates the two architecture types at every noisy-tool count: the

flattened architecture's mean runs from 51.5% down to 40.6% as noisy tools accumulate, whereas the hierarchical architecture stays near 16% throughout, and the per-gateway spread within each type never closes this gap. Environment injection shows no such separation: the ASR means of the two architectures track each other closely, with overlapping per-gateway ranges. This is what the architecture predicts, since a gateway governs which tools enter the agent's context but not what an environment returns. By exposing tools through discovery meta-tools, hierarchical gateways give an injected tool's malicious content fewer chances to reach the agent's context, which translates into better security in practice.

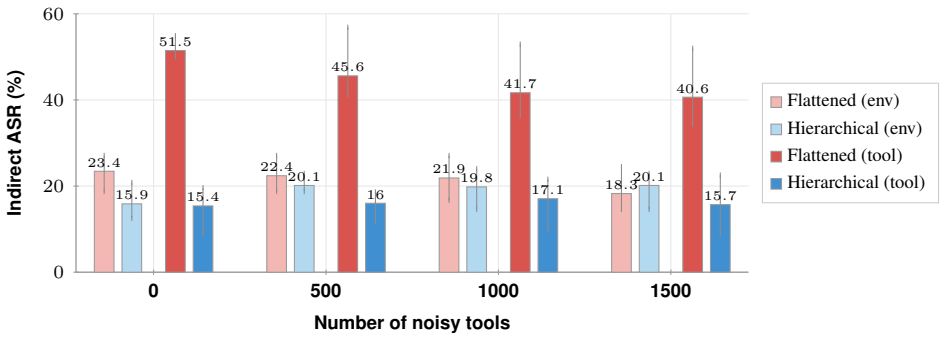

Figure 4: **Indirect attack success rate for environment injection and tool injection** under OpenClaw with GPT-5.1, aggregated into the two architecture types. Each bar is the architecture mean; the error bar spans the per-gateway minimum and maximum within the architecture.

Flattened gateways get more expensive as the tool set grows, but neither design is cheaper across the board. Figure 5 reports the two components of interaction cost. The two architecture types diverge far more sharply in token cost: under GPT-5.4 the flattened architecture's mean climbs from 8.3k to 233.3k tokens per turn, whereas the hierarchical architecture remains stable, from 9.1k to 12.6k. The gap is mechanical: a flattened gateway keeps the entire tool list in the agent's context throughout the task, so every turn's prompt carries all tool schemas, whereas a hierarchical one keeps all but the retrieved tools out of context. The latency comparison of two types of gateway depends on the number of tools. With no noisy tools, flattened gateway is in fact faster (20.8s against 30.1s), since discovery costs extra turns. But the advantage reverses between 500 and 1,000 tools, and by 1,500 flattened gateway needs 64.1s against 38.7s. What flattened gateway saves in turns on a small tool set it more than repays in context once the tool set is large. Appendix B.3 reports the GPT-5.1 curves and per-tool-call latency.

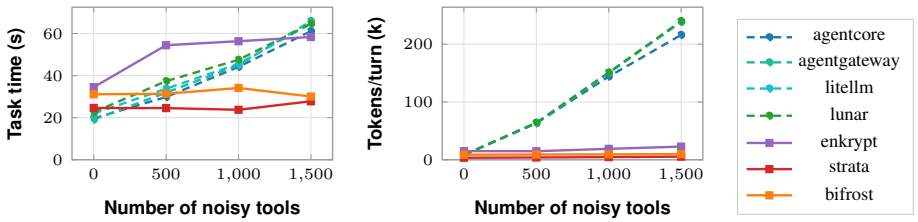

Figure 5: **Interaction cost as the tool set scales.** task duration (left) and average tokens per turn (right) of different gateways under the OpenAI Agents SDK with GPT-5.4.

6 Conclusion

We introduced GatewayBench, the first systematic benchmark for evaluating MCP gateways across capability, efficiency, and adversarial robustness. Through extensive experiments spanning diverse gateway implementations, agent configurations, and noisy tool scales, we underscore that the MCP gateway plays a critical role in determining an agent system's overall performance and security. As a core component of the agent harness, the gateway should be designed and evaluated together with other harness components. Overall, GatewayBench provides a rigorous testbed for evaluating MCP gateway designs and contributes to building more capable, efficient, and secure agentic systems.

7 AI USE STATEMENT

In this work, we used generative AI tools for support qualitative and thematic data analysis. We have not used generative AI tools for generate synthetic data sets, help develop theoretical models or conceptual frameworks, design or provide feedback on research methodology or experiments, implement methods, assist with translation, clean and reformat dataset, interpret results. And formulate mathematical claims, provide critical ingredients for proving mathematical claims, assist in the writing of proofs, propose or refine hypotheses are not applicable to this work. Additionally, we used generative AI tools for create or modify scientific figures, edit a research paper to improve readability, summarize or analyse existing literature. We have reviewed all AI-assisted work. We checked all LLM-generated code used for data analysis to verify its correctness and ensure that the analyses were implemented as intended. We also verified that the data used in all figures matched the original analysis results. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REFERENCES

Agent Gateway. AgentGateway, 2025. URL https://github.com/agentgateway/agentgateway.

AmoyLab. Unla, 2025. URL https://github.com/AmoyLab/Unla.

Anthropic. Introducing the model context protocol, November 2024. URL https://www.anthropic.com/news/model-context-protocol. Accessed: 2026-07-15.

Anthropic. Effective harnesses for long-running agents, November 2025. URL https://anthropic.com/engineering/effective-harnesses-for-long-running-agents. Accessed: 2026-07-15.

Arcade. The MCP gateway pattern: Scaling agentic integrations without tool sprawl, January 2026. URL https://www.arcade.dev/blog/mcp-gateway-pattern/. Accessed: 2026-07-15.

Arcade AI. Arcade, 2026. URL https://www.arcade.dev/.

AWS. Amazon Bedrock AgentCore Gateway, 2025. URL https://docs.aws.amazon.com/bedrock-agentcore/latest/devguide/gateway.html.

AWS. Governing AI assets at scale with MCP gateway and registry, June 2026. URL https://aws.amazon.com/blogs/opensource/governing-ai-assets-at-scale-with-mcp-gateway-and-registry/. Accessed: 2026-07-15.

Chaithanya Bandi, Razvan-Gabriel Dumitru, Ben Hertzberg, Divyansh Agarwal, Geobio Boo, Tejas Polakam, Sami Hassaan, Jeff Da, HiJae Kim, Vipul Gupta, et al. Mcp-atlas: A large-scale benchmark for tool-use competency with real mcp servers. *arXiv preprint arXiv:2602.00933*, 2026.

BerriAI. LiteLLM, 2023. URL https://github.com/BerriAI/litellm.

Zhaorun Chen, Xun Liu, Haibo Tong, Chengquan Guo, Yuzhou Nie, Jiawei Zhang, Mintong Kang, Chejian Xu, Qichang Liu, Xiaogeng Liu, et al. Decodingtrust-agent platform (dtap): A controllable and interactive red-teaming platform for ai agents. *arXiv preprint arXiv:2605.04808*, 2026.

Edoardo Debenedetti, Jie Zhang, Mislav Balunovic, Luca Beurer-Kellner, Marc Fischer, and Florian Tramèr. Agentdojo: A dynamic environment to evaluate prompt injection attacks and defenses for llm agents. *Advances in Neural Information Processing Systems*, 37:82895–82920, 2024.

Docker. Docker MCP registry, 2025. URL https://github.com/docker/mcp-registry.

Enkrypt AI. Secure MCP Gateway, 2025. URL https://github.com/enkryptai/secure-mcp-gateway.

Herman Errico, Jiquan Ngiam, and Shanita Sojan. Securing the model context protocol (mcp): Risks, controls, and governance. *arXiv preprint arXiv:2511.20920*, 2025.

Xiang Fei, Xiawu Zheng, and Hao Feng. Mcp-zero: Active tool discovery for autonomous llm agents. *arXiv preprint arXiv:2506.01056*, 2025.

Tiantian Gan and Qiyao Sun. Rag-mcp: Mitigating prompt bloat in llm tool selection via retrieval-augmented generation. *arXiv preprint arXiv:2505.03275*, 2025.

Nishant Gaurav, Adit Akarsh, Ankit Ranjan, and Manoj Bajaj. Dynamic react: Scalable tool selection for large-scale mcp environments. *arXiv preprint arXiv:2509.20386*, 2025.

Xinyi Hou, Yanjie Zhao, Shenao Wang, and Haoyu Wang. Model context protocol (mcp): Landscape, security threats, and future research directions. *ACM Transactions on Software Engineering and Methodology*, 35(10):1–37, 2026.

Klavis AI. Strata, 2025. URL https://www.klavis.ai/docs/concepts/strata.

LangChain. Improving Deep Agents with harness engineering, February 2026. URL https://www.langchain.com/blog/improving-deep-agents-with-harness-engineering. Accessed: 2026-07-15.

Zixiang Liu, Wenrui Liu, Elsie Dai, Wenhan Yu, Lei Yu, Tong Yang, Jinjun Han, and Hong Gao. Mcpagentbench: A real-world task benchmark for evaluating llm agent mcp tool use, 2026. URL https://arxiv.org/abs/2512.24565.

Elias Lumer, Anmol Gulati, Vamse Kumar Subbiah, Pradeep Honaganahalli Basavaraju, and James A Burke. Scalemcp: Dynamic and auto-synchronizing model context protocol tools for llm agents. In *International Joint Conference on Computational Intelligence*, pp. 23–42. Springer, 2025.

Lunar.dev. MCPX, 2025. URL https://docs.lunar.dev/mcpx/.

Ziyang Luo, Zhiqi Shen, Wenzhuo Yang, Zirui Zhao, Prathyusha Jwalapuram, Amrita Saha, Doyen Sahoo, Silvio Savarese, Caiming Xiong, and Junnan Li. Mcp-universe: Benchmarking large language models with real-world model context protocol servers, 2025. URL https://arxiv.org/abs/2508.14704.

Maxim. Bifrost AI Gateway, 2025. URL https://github.com/maximhq/bifrost.

Microsoft. MCP Gateway, 2025. URL https://github.com/microsoft/mcp-gateway.

Guozhao Mo, Wenliang Zhong, Jiawei Chen, Qianhao Yuan, Xuanang Chen, Yaojie Lu, Hongyu Lin, Ben He, Xianpei Han, and Le Sun. Livemcpbench: Can agents navigate an ocean of mcp tools? *arXiv preprint arXiv:2508.01780*, 2025.

OpenAI. OpenAI Agents SDK, 2025. URL https://github.com/openai/openai-agents-python. Accessed: 2026-07-15.

OpenAI. Harness engineering: Leveraging Codex in an agent-first world, February 2026. URL https://openai.com/index/harness-engineering/. Accessed: 2026-07-15.

OpenClaw. Introducing OpenClaw, 2026. URL https://openclaw.ai/blog/introducing-openclaw. Accessed: 2026-07-15.

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, Sihan Zhao, Runchu Tian, Ruobing Xie, Jie Zhou, Mark Gerstein, Dahai Li, Zhiyuan Liu, and Maosong Sun. Toolllm: Facilitating large language models to master 16000+ real-world apis, 2023.

Merlin Stein. How are ai agents used? evidence from 177,000 mcp tools. *arXiv preprint arXiv:2603.23802*, 2026.

Zhenting Wang, Qi Chang, Hemani Patel, Shashank Biju, Cheng-En Wu, Quan Liu, Aolin Ding, Alireza Rezazadeh, Ankit Shah, Yujia Bao, et al. Mcp-bench: Benchmarking tool-using llm agents with complex real-world tasks via mcp servers. *arXiv preprint arXiv:2508.20453*, 2025.

Zhiqiang Wang, Yichao Gao, Yanting Wang, Suyuan Liu, Haifeng Sun, Haoran Cheng, Guanquan Shi, Haohua Du, and Xiangyang Li. Mcptox: A benchmark for tool poisoning on real-world mcp servers. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pp. 35811–35819, 2026.

Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, et al. Autogen: Enabling next-gen llm applications via multi-agent conversation. *arXiv preprint arXiv:2308.08155*, 2023.

Zijian Wu, Xiangyan Liu, Lingjun Chen, Fanqing Meng, Lingxiao Du, Yiran Zhao, Fanshi Zhang, Yaoqi Ye, Jiawei Wang, Zirui Wang, et al. Mcpmark: A benchmark for stress-testing realistic and comprehensive mcp use. In *International Conference on Learning Representations*, volume 2026, pp. 79852–79895, 2026.

Xiangzhe Xu, Hamidreza Saghir, Qianhui Wu, Marc-Alexandre Côté, Tong Wang, Kiran Lakkaraju, Kexin Pei, and Xiangyu Zhang. The devil is in the interface: Evaluating how tool architecture shapes coding agent behavior. *arXiv preprint arXiv:2608.11386*, 2026.

Qiusi Zhan, Zhixiang Liang, Zifan Ying, and Daniel Kang. Injecagent: Benchmarking indirect prompt injections in tool-integrated large language model agents. In *Findings of the Association for Computational Linguistics: ACL 2024*, pp. 10471–10506, 2024.

Dongsen Zhang, Zekun Li, Xu Luo, Xuannan Liu, Peipei Li, and Wenjun Xu. Mcp security bench (msb): Benchmarking attacks against model context protocol in llm agents. *arXiv preprint arXiv:2510.15994*, 2025a.

Hanrong Zhang, Jingyuan Huang, Kai Mei, Yifei Yao, Zhenting Wang, Chenlu Zhan, Hongwei Wang, and Yongfeng Zhang. Agent security bench (asb): Formalizing and benchmarking attacks and defenses in llm-based agents. In *International Conference on Learning Representations*, volume 2025, pp. 35331–35366, 2025b.

Xuanjun Zong, Zhiqi Shen, Lei Wang, Yunshi Lan, and Chao Yang. Mcp-safetybench: A benchmark for safety evaluation of large language models with real-world mcp servers. *arXiv preprint arXiv:2512.15163*, 2025.

A Dataset Details

Adversarial and Benign Tasks from DTap. The DecodingTrust-Agent Platform (DTap) is a controllable and interactive red-teaming platform spanning 14 real-world domains, designed to evaluate the security and resilience of LLM agents. To capture a broad spectrum of enterprise and consumer applications, we select three representative domains: *Customer Relationship Management (CRM)*, *Finance*, and *Travel*. For each domain, we uniformly sample the original dataset at fixed intervals to construct a balanced subset. Specifically, we extract 50 benign tasks, 50 direct attack tasks, and 50 indirect attack tasks per domain. This yields a comprehensive collection of 450 tasks, enabling us to systematically quantify how varying tool-set scales and gateway architectures influence an agent's resilience against adversarial manipulation.

Real-World Execution Tasks from MCP-Universe. To further assess gateway performance in authentic execution environments, we incorporate tasks from MCP-Universe, which focuses on benign tasks grounded in real-world MCP servers and live environments. We select three complex, multi-step domains: *Repository Management*, *Browser Automation*, and *Location Navigation*. To ensure that the benchmark effectively discriminates between different gateway architectures, we manually review and remove tasks that are either trivially solvable or too hard for tested agents, resulting in a refined set of 74 high-quality tasks that provide a meaningful signal for distinguishing gateway designs.

B Additional Results

B.1 Real-World Tool Scaling on MCP-Universe

This experiment was run in an early, exploratory stage of the project, so it covers only three representative gateways and a smaller range of noisy tools than our main experiments. We include it to confirm that our tool-scaling trends also hold when the task tools come from real MCP servers, not only in the simulated task environments of DTap (Section 4.1). Table 4 reports benign task success for three representative gateways under the OpenAI Agents SDK with GPT-5.1, as the real-world noisy-tool pool grows from 158 to 358 tools.

Table 4: Real-world tool scaling on MCP-Universe (OpenAI Agents SDK, GPT-5.1) for three representative gateways. TSR is the fraction of tasks that pass all execution-based checkpoints, and Score is the average fraction of per-task checkpoints passed (partial credit); higher is better for both.

Gateway	158 Noisy Tools		254 Noisy Tools		358 Noisy Tools	
	Score	TSR (%)	Score	TSR (%)	Score	TSR (%)
agentgateway	0.73	52.3	0.66	52.0	0.75	54.0
enkrypt	0.60	34.7	0.62	36.0	0.52	23.3
strata	0.68	44.7	0.68	46.0	0.68	43.7

Within this range, benign task success stays largely flat for all three gateways. Strata, a hierarchical gateway, holds steady (44.7% to 43.7%), which is expected and consistent with its behavior on DTap: because a hierarchical gateway exposes tools only through discovery, additional noisy tools never enter the agent's context. Agentgateway, although flattened, is similarly unaffected (52.3% to 54.0%). This also matches DTap, where agentgateway is the least noise-sensitive of the flattened gateways (Table 3). We read this as the combination of two factors: the pool here reaches only 358 tools, well short of the scale at which flattened gateways begin to lose task success in the main experiment, and agentgateway is a comparatively robust flattened implementation. The only visible decline is enkrypt at the largest pool (36.0% to 23.3%); given the single-run setup and enkrypt's weak-model capability strain under GPT-5.1, we do not over-interpret this point.

B.2 Indirect Attack Success by Injection Type

Among the 150 indirect attack tasks in our dataset, 48 are environment injection attack tasks and 102 are tool injection attack tasks. Figure 6 gives the full per-gateway decomposition for all three

agent configurations. The tool-injection gap between the two architecture types appears in every configuration, but the two OpenAI Agents SDK settings each add a caveat that the OpenClaw view in the main text removes.

Under GPT-5.4 the two architecture types are capability-matched (mean TSR 85.1% and 87.2%), and the same pattern holds: tool injection separates them (the flattened architecture's mean runs from 53.9% down to 45.1% against a hierarchical architecture near 31%), whereas environment injection does not (the two means stay within about 3 points at every noisy-tool count). Hierarchy alone, however, is not what protects the agent: among the hierarchical gateways, tool-injection ASR runs from roughly 20% (bifrost) through 33% (strata) to 41% (enkrypt), a spread as wide as the gap between the architecture types, so how each interface exposes an injected tool matters as much as whether it is hierarchical.

Under GPT-5.1 with the OpenAI Agents SDK the hierarchical architecture shows the lowest attack success of all (tool injection near 21%, environment injection near 15%), but this must not be read as protection. The same configuration collapses hierarchical benign task success to 47.5% (Table 3), so an agent that cannot drive tool discovery also fails to carry out an injected goal; its apparent robustness is capability collapse, not safety. The wider environment-injection gap here (35.9% for flattened against 15.3% for hierarchical at 0 tools) is likewise an effect of the weak model rather than the gateway, since a gateway does not govern what an environment returns.

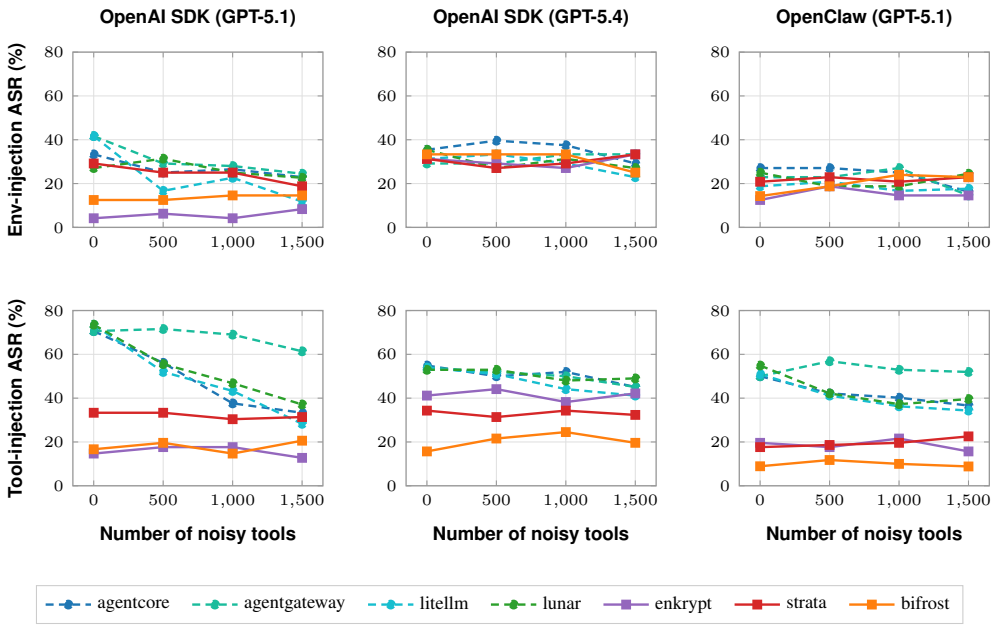

Figure 6: Indirect attack success rate by injection type as the tool set scales: environment injection (top) and tool injection (bottom), for each agent configuration (columns). One line per gateway; flattened gateways are dashed, hierarchical gateways solid.

B.3 Detailed Efficiency Results

Figure 7 extends the interaction-cost analysis of Section 5.2 to both OpenAI Agents SDK models and adds tool-call latency as a third metric. The task-time and token trends match the main text: flattened gateways grow roughly linearly with the tool set in both, whereas hierarchical gateways stay nearly flat. The two token curves are almost identical across the models, confirming that per-turn token cost is set by the number of injected noisy tools rather than the model.

Tool-call latency exposes a difference that the aggregate task time hides. For flattened gateways the average tool call is cheap and roughly constant (about 0.6s) irrespective of the number of injected noisy tools, since the gateway simply forwards the call to the upstream server. Hierarchical gateways split sharply: strata and bifrost are even faster per call (around 0.2s and 0.1s), whereas enkrypt is far slower and grows with the pool (from 2.6s to 5.4s under GPT-5.4), because its layered discovery

meta-tools return large server-and-tool listings on every call (Section 3). Enkrypt therefore trades many cheap turns for a few expensive discovery calls, which accounts for much of its elevated task time.

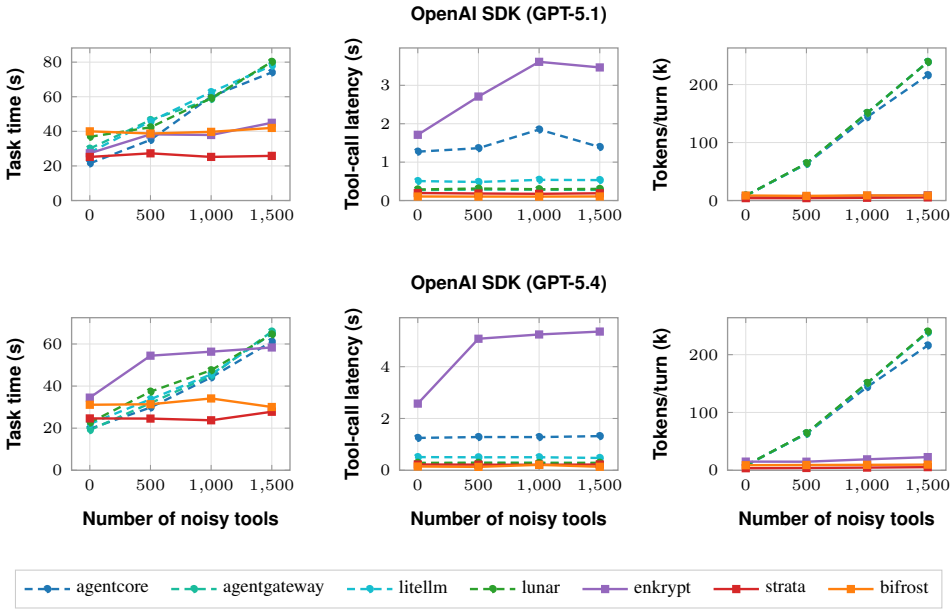

Figure 7: Interaction cost as the tool set scales, under the OpenAI Agents SDK with GPT-5.1 (top) and GPT-5.4 (bottom): task time (left), average tool-call latency (middle), and tokens per turn (right). Flattened gateways are dashed, hierarchical gateways solid.

B.4 Detailed MCP Gateway Analysis

AgentCore Search Tool Effectiveness. Among the flattened gateways evaluated in this work, AgentCore is unique in providing a built-in search tool that semantically matches tools to the agent's intent. To examine whether this additional search tool offers any practical benefit, we count the number of task runs in which the agent actually invoked the search tool. As shown in Table 5, in 1,800 runs per agent setting, search tool usage is negligible. We speculate that the agent directly locates the tools relevant to the task and proceeds without invoking search, since all necessary tools are already visible. Besides, LLM agents are trained to minimize unnecessary tool calls. These results suggest that a search tool provides limited utility when the full tool list is already exposed.

Table 5: Number of task runs in which the AgentCore search tool was invoked. Each agent includes 1,800 task runs.

Agent	Search Used
OpenAI SDK (GPT-5.1)	11
OpenAI SDK (GPT-5.4)	19
OpenClaw (GPT-5.1)	1

Enkrypt. Enkrypt adopts a hierarchical mechanism, which naturally yields good token efficiency. However, its actual implementation severely undermines this benefit. Specifically, Enkrypt returns tool definitions as JSON-encoded configuration blobs that include extensive metadata irrelevant to the agent, such as internal runtime commands, guardrail settings, and connection parameters. Furthermore, its default tool entrypoint attempts to return the full tool list at once but frequently fails due to timeouts when the available tool set is large. When the underlying model is weak, the agent often attempts to skip discovery steps and directly invokes tools with hallucinated names and parameters, which leads to poor task success rates.

Bifrost. Bifrost provides 4 meta tools to the agent: `executeToolCode`, `getToolDocs`, `listToolFiles`, and `readToolFile`. These meta tools creates a virtual filesystem where each server is represented as a Python stub file(.pyi). To reduce the token cost, the detailed description

of the parameters are truncated in the file and agent need to call `getToolDocs` to get the full details. For weak models like GPT-5.1, the generated code often mishandles special characters such as newline characters inside string literals, writing as a physical line break rather than the escaped sequence (e.g. `split('\n')`). This produces incorrect Python code and is consequently rejected by the parser, making the gateway effectively unusable for those models. For indirect malicious tasks, the truncated tool description protects the agent from seeing the malicious message and thus lowers the ASR.

Strata. Strata provides 5 meta tools to the agent: `discover_server_actions`, `get_action_details`, `execute_action`, `search_documentation`, and `handle_auth_failure`. The agent must first search the required tools using a semantic query and get the detail of the tool before invoking. However, when the model is not well trained for this situation like GPT-5.1, it often skips the discovery process and directly invokes hallucinated tools or parameters.