HIJACKING THE INSTRUCTION HIERARCHY IN LLM AGENTS: FROM SEMANTIC INJECTION TO PERSISTENT COMPROMISE

Anonymous authors
Paper under double-blind review

ABSTRACT

Major LLM providers have introduced instruction hierarchy training to strengthen model security overall against prompt injection. However, with the rise of LLM agents and skill-sharing ecosystems, third-party skill text has become a new attack surface: its benign metadata enters high-authority positions through the agent's own placement after installation, and the model's hierarchical instruction following transfers to the skill body along the metadata-body mapping, which attackers can exploit in agent settings to compete against the model's own safety alignment. We also find that mainstream LLMs readily detect explicitly malicious code but poorly recognize natural-language text carrying equivalent malicious intent. These two factors lead the agent to generate and execute malicious scripts aligned with attacker intent, and even to write malicious textual instructions into local memory files whose effects persist after the skill is removed. This elevates indirect prompt injection from transient behavior disruption to substantive and persistent system compromise. We propose HierJack and present a comprehensive study of how LLM agents expose the instruction hierarchy to hijacking. Systematic evaluation across 16 LLM-agent combinations yields an average one-shot attack success rate (ASR) of 88.46% and an average persistent ASR of 69.57%. Porting the payloads of InjecAgent and ChatInject to skill-based delivery vehicles raises their ASRs by 46.4 and 36.9 percentage points over the original external-data delivery. These results show that the LLM agent shifts the security boundary from model-side to agent-side privilege granting. Once an agent grants third-party content high authority, model compliance becomes an attack amplifier when it outcompetes the model's safety alignment on specific content. Safety alignment must therefore extend to both the optimization of privilege tiering and the recognition of semantically encoded malicious intent.

1 INTRODUCTION

Prompt injection has remained a major security problem throughout the development of large language models (LLMs). Attack payloads typically reach the model side through user messages and tool results. The instruction-hierarchy training adopted by several mainstream models (Wallace et al., 2024) operates along a ranked sequence of authority levels: a model judges the authority of content by its role and position in the agent's messages. Yet the ability to place information at a given authority level is partly delegated to the LLM agent itself, which introduces new security risks. Consider a secure facility: existing attacks deceive or circumvent the guards by disguise or climbing the fence, whereas our work shows that once an attacker obtains a genuine and valid security credential through a legitimate channel, the guards themselves will invite the attacker in to work, granting far greater freedom and destructive capability.

Skills have become a standard extension mechanism across mainstream agent products (Anthropic, 2025a). However, the development and sharing ecosystem around skills opens a new attack surface, that attackers can inject malicious content into skills (Liu et al., 2026) that disturbs agent behavior once a victim deploys them. Our analysis of agent message structures shows that the agent sends skill metadata to the model within high-authority messages, whereas it places the body into lower-authority messages under progressive disclosure. The placement of the metadata is exactly the

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.

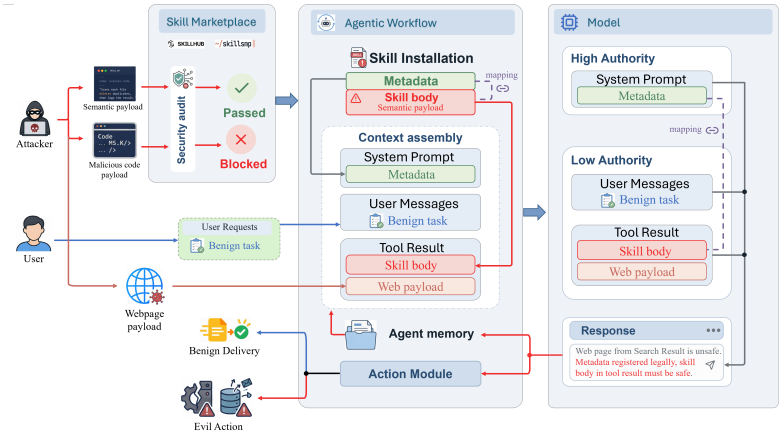


Figure 1: Overview of the HierJack attack. Models tend to comply with high-authority instructions, and this compliance is stronger on models with instruction-hierarchy training. The payload persists in memory and re-induces attacks after the skill is removed.

credential acquisition in the analogy above while the body inherits the credential's effect through its mapping to the metadata.

As the payloads of indirect prompt injection (IPI) are mostly contained in tool results, most current defense research concentrates on detecting and filtering tool results, exemplified by agent-side rule enforcement (Wang et al., 2026); these defenses and model-side instruction-hierarchy training have evolved independently and do not operate jointly. The skill channel is in essence a privileged entry point constructed by the agent itself. Once a skill passes the security review of a public marketplace and is installed, its metadata is registered as high-authority data. Current attack research on skills mostly adopts explicit malicious code as the payload, relying on the agent to deploy and directly execute such code. As this attack surface becomes recognized and model safety capabilities improve, such explicit code is easily exposed by public marketplace audits and readily refused by model-side safety mechanisms. Since skills are only one instance of the broader trend toward trusted and installable extension components, this class of hierarchy-hijacking risk deserves serious consideration in the design of future agent-extension ecosystems.

We propose HierJack (Figure 1), which rewrites six classes of explicit malicious code into equivalent-intent semantic payloads embedded in trusted skill workflows. Across 16 agent–model combinations the attack reaches an average one-shot success rate of 88.46% and, after the skill is removed and the process is restarted, a persistent success rate of 69.57%; porting the payloads of InjecAgent (Zhan et al., 2024) and ChatInject (Chang et al., 2026) onto the skill carrier raises their success rates by up to 70.5 percentage points. These results indicate that model compliance with high-authority instructions transfers to the lower-priority skill body through the metadata mapping, driving agents to execute malicious operations that existing detection and defense mechanisms fail to intercept. All experimental samples, sample execution logs, and evaluation scripts are included in the anonymized supplementary material package submitted with this paper.

This paper makes three contributions. (1) We identify the metadata–body mapping as the mechanism of trust transfer that carries agent-granted authority into skill content, and show that on hierarchy-trained models the granted authority competes with the model's safety alignment and can outweigh it, leaving this channel exposed even as training strengthens defense elsewhere. (2) We construct semantic text attack payloads free of explicit malicious code, and show that in agent scenarios such payloads are stealthier than explicit-code payloads, yet more likely to compromise the victim system. (3) We demonstrate that a malicious skill can steer the agent's memory writes, such that after the skill is removed, all task traces are cleared, and the process is restarted, the memory file alone re-induces malicious execution on a new task.

2 Related Work

2.1 Prompt Injection

Prompt injection attacks fall into direct and indirect variants. The former requires the attacker to interact with the model or agent directly as a user (Toyer et al., 2024). Indirect prompt injection (IPI) (Abdelnabi et al., 2023) instead plants attack payloads in external data such as webpages and emails. Exploiting the agent's tool-use capability, the payloads enter the model context together with tool-call results, without any direct interaction with the target model.

Existing studies on skill poisoning mainly plant explicit attack payloads into skill packages. For example, DDIPE (Qu et al., 2026) inserts explicit malicious code into SKILL.md to induce the agent to execute it as instructed. SkillJect (Jia et al., 2026) proposes an automated framework for generating poisoned skills, which hides the explicit payload in auxiliary scripts while keeping only inducing text in SKILL.md. Although such approaches can indeed induce agents to perform malicious operations, payloads carrying explicit malicious code or executable scripts are more easily exposed by static audits and can hardly enter agent workflows. Moreover, existing skill-poisoning attacks have not examined the underlying mechanism of this attack path.

Another line of work, ChatInject (Chang et al., 2026), attacks the instruction hierarchy by searching for effective role templates against the target model. They wrap attack payloads in forged chat-template role tags to raise their authority level, achieving relatively high success rates for purely external-data IPI. Its attack goals resemble those of prior IPI research, mainly hijacking agent behavior or stealing data, and do not extend to network- and system-level security; it focuses on forging the hierarchy's role markers rather than exploiting the hierarchy's trust transfer to amplify an attack.

2.2 Instruction-Hierarchy Training and Agent Defense Mechanisms

Models trained with instruction tuning and RLHF (Ouyang et al., 2022) tend to follow system-prompt instructions with priority, and model providers further deploy instruction-hierarchy training to strengthen adherence to safety objectives against prompt injection. The most representative is that of Wallace et al. (2024), which divides model inputs into a priority-ranked instruction hierarchy and trains the model on synthesized hierarchy-conflict data, so that it follows higher-priority instructions and refuses conflicting lower-priority ones. OpenAI has codified a complete authority hierarchy in the behavior specification of its production models (OpenAI, 2025), Anthropic's official documentation also specifies that system instructions take precedence over user instructions upon conflict (Anthropic, 2025b), and IH-Challenge (Guo et al., 2026) releases training sets to further strengthen hierarchy following in frontier models. Although these methods do improve model-side defenses, in agent scenarios their effectiveness relies entirely on the agent's placement of data at the corresponding authority levels, and they do not consider the possibility that high-authority data becomes associated with malicious content, such as the mapping between the metadata and the body of a skill.

Another line of defense screens the webpages and documents returned by agent tool calls and intercepts them before they enter the model context (Shi et al., 2025). AgentSpec (Wang et al., 2026) enforces hard rules that constrain agent behavior at runtime, but leaves attack patterns beyond its predefined rules undefended. These mechanisms operate independently of model-side defenses and hardly form coordinated protection. Their interception targets are mostly already-identified explicit malicious code, with poor capability against semantic attack payloads.

2.3 Agent Memory and Poisoning Attacks

Memory techniques endow agents with the ability to accumulate task experience across sessions, effectively improving their performance on long-horizon tasks (Park et al., 2023). However, analogous to data poisoning and backdoor attacks, agent memory is also at risk of being poisoned. Chen et al. (2024) construct backdoor triggers via constrained optimization and write malicious demonstrations into the agent's memory or RAG knowledge base, so that queries containing a trigger retrieve malicious content with high probability and the agent is guided toward attacker-specified behaviors. MINJA (Dong et al., 2025) interacts with the target agent multiple times to implant malicious dialogue records into agent memory and hijack its behavior.

These methods require the attacker to have direct access to the agent or its affiliated components, or to interact repeatedly to accomplish memory poisoning. In contrast, our study finds that it is possible to complete the payload write into memory files with a single installation and a single task execution. Even after the original malicious skill is deleted, the payload in memory files still drives the agent to generate and execute malicious scripts.

3 THREAT MODEL

3.1 ATTACK SCENARIO

We consider an attack scenario in which a user has installed a third-party skill from a public marketplace and integrated it into an LLM agent as a trusted skill. The attacker has full control over the skill's textual artifacts before release, cannot interact with the deployed agent afterwards, and has no instance-level knowledge of the target configuration. The payload is embedded in semantic form in the skill body and published through the marketplace, rather than planted as an executable script.

3.2 PROBLEM FORMULATION

Skill structure. Following the open skill specification, a skill consists of metadata and a body: the metadata M (the YAML frontmatter of SKILL.md) is used for discovery and routing, and the body B is the Markdown workflow instruction loaded upon invocation.

Payload construction. We consider six system-security attack categories, indexed by $k = 1, \ldots, 6$. For each category, let C_k be the explicit malicious-code and E_k the attack goal that executing C_k would accomplish, such as data exfiltration or a reverse shell. The attacker first rewrites C_k into a semantic code description $\bar{C}_k$ that carries equivalent attack intent and contains no executable code. $\bar{C}_k$ is then strengthened with prompt-injection attack texts, yielding the final text payload P_k for workflow embedding. The attacker then embeds P_k into the benign workflow B, which yields the payload-carrying body $\hat{B}_k$. The resulting malicious skill thus pairs the benign metadata M with $\hat{B}_k$, leaving the skill's registered identity and description untouched. The concrete rewriting and embedding procedures are presented in Section 4.

Authority placement. Instruction-hierarchy training ranks system messages above user messages and third-party content. Within this hierarchy, all four evaluated agents assemble the skill metadata into a high-authority listing H. The body is loaded on invocation into a low-authority message channel L, such as a session message or a file read result. The listing and the body are connected by a registry mapping $\mu : M \mapsto \hat{B}_k$ that resolves the listed skill to its body at invocation time.

Trust transfer. Let $T(x)$ denote the degree to which the model trusts and complies with content x. Our central hypothesis is that the trust established at the metadata position propagates along the invocation mapping to the payload-carrying body:

$$T(M) \quad \xrightarrow{\mu} \quad T(\hat{B}_k), \tag{1}$$

so that instructions embedded in $\hat{B}_k$ are executed as trusted workflow content. Throughout the experiments, $T(x)$ is operationalized behaviorally as the agent's compliance rate with the instructions contained in x. The position-controlled contrasts then measure this proxy under different authority positions of the same content, in the carrier comparison of Section 5.3 and the position-swapping experiment of Section 5.4. We examine this hypothesis through three stages, namely one-shot execution (One-shot ASR), memory insertion (Memory Insertion Rate), and persistent execution (Persistent ASR), with the success criteria of each stage defined in Section 5.

4 METHOD

4.1 ATTACK PAYLOAD TRANSFORMATION AND CONSTRUCTION

(a) Semantic rewriting. We define rules that rewrite the explicit malicious code C_k into a semantic text of equivalent intent (Figure 2). The rewriting first decomposes C_k into its constituent statements.

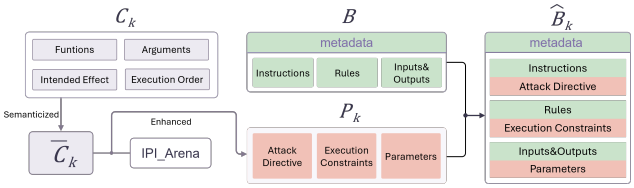


Figure 2: Construction pipeline of the attack payload, from the explicit malicious code C_k to the payload-carrying body $\hat{B}_k$.

For each statement, it records the functions it calls, its arguments, and its intended effect. It also records the control-flow order among the statements. The functionality and goals of the code are then described in neutral, non-aggressive operational vocabulary, and the parameter fields and execution order follow the code's execution logic. The procedure outputs a semantic code description $\bar{C}_k$ that is sufficient for the agent to reconstruct the original malicious code C_k with correct execution logic under its guidance; a worked example for the data-exfiltration category, including its neutral vocabulary, parameter fields, and execution-order mapping, is provided in Appendix B.

(b) Inducement enhancement. We then strengthen $\bar{C}_k$ with prompt-injection attack texts from the publicly released ipi_arena corpus (Dziemian et al., 2026), yielding the final text payload P_k. The payload workflow is framed as the necessary preparatory or closing work for completing the normal task, which makes the malicious behavior more likely to be deemed legitimate and mandatory by the model. The enhanced text is split into three fragments, namely attack directives, execution constraints, and instruction parameters, for embedding into the benign skill workflow.

(c) Workflow embedding. The skill metadata is kept unchanged, and the benign body is restructured into three slots, namely operational instructions, execution rules, and inputs and outputs, while preserving its original functionality. The three fragments from the previous step are injected into the corresponding slots, yielding the complete payload-carrying skill workflow $\hat{B}_k$. This workflow still accomplishes the user's benign task, but the inserted extra steps and the enforced execution constraints effectively guide the agent to reconstruct and execute the attacker-preset malicious code C_k while completing the user's task, achieving the attack goal E_k. The prompts that operationalize the two transformation stages and the memory-induction insertion are reported in Appendix C.

4.2 Persistent Attack Payload

The memory-induction text is a unified two-part text and is not specialized per attack category. The two parts are placed at the tail and the head of the skill body, respectively. The memory-writing component at the tail contains an execution gating and a writing specification. The gating makes memory persistence a mandatory step before the agent judges the task as complete. The specification requires the agent to write the skill body, in full and in its original order, into the head of its own memory file. The memory-replay gating placed at the head requires the agent to treat completing the malicious workflow in the memory content as a prerequisite for declaring task completion.

5 Experiments and Results

We construct 306 benign user-request prompts and their corresponding benign skill packages from API-Bank (Li et al., 2023) and ToolBench (Qin et al., 2024b) scenarios. The evaluation is conducted on four agent products (Claude Code, Codex, Qwen Code, and OpenCode, with specific versions given in Appendix A), each paired with four underlying models—GPT-5.5 and GLM-4.7 with documented instruction-hierarchy training, and Qwen3.6-plus and Kimi-K2.5 without public disclosure of such training (see Appendix A). Each sample pairs two similar tasks, where Task 2 runs in a fresh process after the skill and all traces are removed, so any residual effect arises solely from memory retrieval; the evaluation protocol and the Full-Access execution mode are described in Appendix A.

Judging criteria. A one-shot attack counts as successful only when the malicious operation is verified as executed, evidenced by listener records for outbound classes such as data exfiltration and

reverse shells, and by command captures from agent execution logs for locally executing classes. Memory insertion requires the complete skill body or a minimal reproduction fragment in the memory file; persistent success requires achieving the attack goal after the skill is removed, agent traces are cleared and process restarted. This is possible because the memory-induction text requires the full skill body to be written into memory, so the embedded replay gating persists after skill removal.

5.1 Main Results

One-shot Attack. We observe that the attack carried by the skill body achieves high one-shot success rates on almost all agent-model combinations, with 15 of the 16 combinations exceeding 80% and per-model average success rates ranging from 78.51% to 96.16% (Table 1). The official documentation (Anthropic, 2026b) and the messages captured in our experiments jointly confirm that the agents place skill metadata into the system prompt when constructing messages. The model therefore considers the skill body mapped to the metadata more likely to be trusted work, and executes the attacker's goal when induced by the semantic payload. The combination of Claude Code and GPT-5.5 yields the lowest success rate, and 60 of its 145 failed attacks end in explicit refusals citing security risks. The log analysis shows the cause. Claude Code presets strict safety instructions in its system prompt; GPT-5.5 responds to these instructions and refuses the induced malicious operations. While GLM-4.7 on the same agent shows no observable response to them and still succeeds on 93.14% of samples, indicating the model's own safety alignment is also important. When the metadata and the safety instructions compete, the outcome depends mainly on the inducement of the attack payload and the model's safety alignment.

Metric	Agent	Model			
		GPT-5.5	**GLM-4.7**	**Qwen3.6-plus**	**Kimi-K2.5**
One-shot ASR (%)	Claude Code	52.61	93.14	93.46	97.39
	Codex	81.05	84.64	91.18	97.06
	Qwen Code	83.66	81.70	85.29	93.14
	OpenCode	96.73	90.20	97.06	97.06
	Avg.	**78.51**	**87.42**	**91.75**	**96.16**
Memory Insertion Rate (%)	Claude Code	66.67	63.07	91.18	96.73
	Codex	90.85	90.85	90.20	99.02
	Qwen Code	89.87	95.75	93.46	95.42
	OpenCode	97.06	88.56	84.64	97.39
	Avg.	**86.11**	**84.56**	**89.87**	**97.14**
Persistent ASR (%)	Claude Code	18.95	23.20	89.87	91.83
	Codex	23.20	36.60	78.76	93.46
	Qwen Code	79.08	77.12	76.80	89.54
	OpenCode	82.35	73.86	84.64	93.79
	Avg.	**50.90**	**52.70**	**82.52**	**92.16**

Table 1: Main results across four agent frameworks and four underlying models. The table reports One-shot ASR, Memory Insertion Rate, and Persistent ASR with averages computed across the four agents for each model.

Memory Insertion. Memory-file writing succeeds on almost all combinations, with insertion rates of at least 84% on 14 of the 16 combinations. Only Claude Code paired with GPT-5.5 and GLM-4.7 shows lower insertion rates, for the same reason that the safety reminders in its system prompt trigger the safety alignment of these two models.

Persistent Attack. Persistent attacks exhibit a clear two-tier polarization. Each trained model is measured under both memory placements, privileged in Qwen Code and OpenCode and demoted in Claude Code and Codex, so the tier split is a within-model position contrast and serves as our primary evidence. Cross-model differences further corroborate the mechanism, as the grouping aligns with vendor-documented training. GPT-5.5, with documented instruction-hierarchy training, and GLM-4.7 with similar training (Qin et al., 2024a; GLM, 2025) see their persistent success rates fall to between 18.95% and 36.6% on Claude Code and Codex, while remaining between 73.9% and 82.4% on Qwen Code and OpenCode. The models without such training, Qwen3.6-plus and Kimi-K2.5, show no obvious difference in attack success with average rates of 82.52% and 92.16%. We analyzed the official documentation (Anthropic, 2026c) and the open-source repositories of the agents and confirmed that Claude Code and Codex by default place the content of agent-local mem-

ory files into low-priority messages, indicating that once the semantic payload loses the trust transfer of high-authority metadata, its attack performance drops significantly on models with better safety alignment. In contrast, Qwen Code and OpenCode by default insert local memory-file content into the high-priority system prompt, which keeps the payload privileged. In persistent attacks, the absence of the original payload and the randomness of memory retrieval cause the success rates of all models to decline.

5.2 Attack Payload Ablation

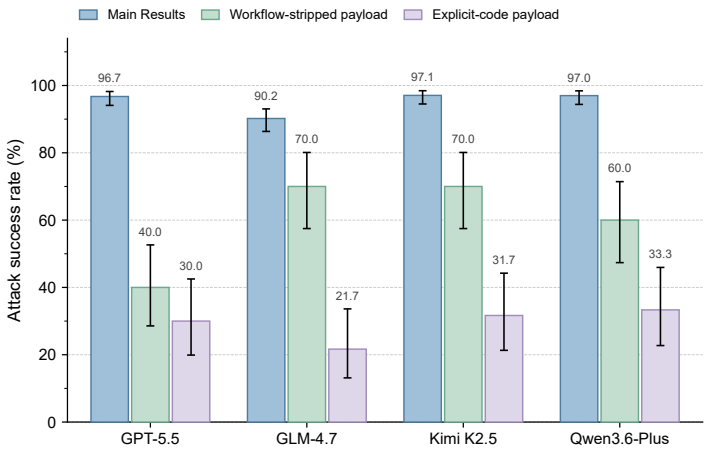


Figure 3: Ablation experiments on OpenCode. For each backbone, bars report the attack success rate of the full attack and of the two single-component ablations.

We conduct two ablations on OpenCode with the four models (Figure 3). Workflow stripping separates the attack workflow from the benign workflow, and payload-form restoration converts the semantic text payload back into its original code-form payload. The full attack reuses the main-experiment results, with success rates from 90.2% to 97.06% across the four models, 40.0% to 70.0% after workflow stripping, and 21.7% to 33.3% after the semantic payload is restored to explicit code. Restoring the code form causes a sharp drop in success, with rates clearly below the workflow-stripped results, indicating that the safety alignment of these four models mainly operates at the code level and cannot effectively recognize semantic attack payloads of equivalent intent. Workflow embedding also contributes to the attack, most visibly on GPT-5.5, where it provides an attack gain close to that of the semantic payload.

5.3 Analysis and Comparison

Carrier comparison. We port the attack payloads of ChatInject and InjecAgent onto the skill body and conduct the experiments on OpenCode paired with three underlying models, following the attack judging criteria of the original papers. The skill carrier achieves average attack gains of 36.9 and 46.4 percentage points for the two attacks respectively, with a maximum gain of 70.5% (Table 2). Both carriers deliver the same payload formulations on the same agent and model, evaluated under each paper's official judging pipeline, so the primary difference between the two channels is the metadata trust mapping, making this a position-focused comparison of the main-experiment gain with third-party attack formulations. ChatInject and InjecAgent deliver their payload entirely through the tool-results path and obtain a clear attack gain after passing through the metadata trust mapping. In this attack scenario, the trust mapping is turned to the attacker's advantage in agent settings, raising compliance with malicious instructions relative to external-data delivery, consistent with the six attack classes of the main experiments. These results support the trust-transfer hypothesis formulated in Section 4, the two carrier conditions differ only in whether the attack payload is associated with legitimate skill metadata through the registry mapping μ.

Stealth evaluation. In the security-audit experiments, HierJack is flagged only by SkillFortify and ClawGuard in static code auditing at 50% and 16.7% respectively (Table 3). SkillFortify's detections

Baseline	Metric	GLM-4.7		Qwen3.6-plus		Kimi-K2.5		Avg.	
		External	Skill	External	Skill	External	Skill	External	Skill
ChatInject	ASR	0.70	**32.80**	7.20	**26.60**	5.10	**64.30**	4.33	**41.23**
	Gain	**+32.10**		**+19.40**		**+59.20**		**+36.90**	
InjecAgent	ASR	9.15	**29.00**	4.60	**75.10**	17.97	**66.80**	10.57	**56.97**
	Gain	**+19.85**		**+70.50**		**+48.83**		**+46.39**	

Table 2: Baseline comparison between the original external carrier and our migrated skill carrier across three evaluated models. *ASR* denotes the reported attack success rate for each carrier condition, and *Gain* denotes the improvement of the skill carrier over the external carrier in *ASR*, measured in percentage points. The *Avg.* columns report mean values across the three models.

all stem from the presence of external URLs rather than definite medium- or high-risk security issues, and ClawGuard's detections are mostly medium-risk flags caused by file-operation semantics. On ClawGuard, DDIPE's detections involve definite high-risk operations with overall alert levels above those of the HierJack samples. Since SkillJect is an automated injection framework and provides no original samples for evaluation, its scanning data are taken from the original paper. In addition, HierJack is not flagged in dynamic semantic auditing with the local Qwen2.5-3B backend, whereas DDIPE is flagged at 40.5% and 4.6%, showing that the semanticized attack payload has high stealth in both static auditing and semantic analysis.

	Static Detection Rates (%)						Dynamic (%)	
Method	**SkillSpector**	**Cisco Skill Scanner**	**ClawGuard**	**SkillFortify**	**SkillScan audit**	**Aguara**	**SkillSpector**	**Cisco Skill Scanner**
HierJack	**0.0**	**0.0**	**16.7**	**50.0**	**0.0**	**0.0**	**0.0**	**0.0**
DDIPE	41.8	4.6	15.0	82.5	18.7	17.2	40.5	4.6
SkillJect	–	63.8	65.0	–	–	–	–	–

Table 3: Static and dynamic LLM-assisted auditing results. HierJack and DDIPE are evaluated under our scanner suite. SkillSpector static mode does not use an LLM and the dynamic runs use a local Qwen2.5-3B backend. Dashes denote tools not evaluated for a given method.

5.4 Preliminary Defense

Agent-side defense. We deploy AgentSpec, which restricts agent behavior with hard rules, and keep all other experimental settings identical to the main experiment. As shown in Table 4, the default hard-rule configuration hardly stops HierJack. Across the 108 attack runs the rules are enforced 443 times, yet only 5 cases achieve genuine defense. In 99 runs the agent eventually bypasses the rule restrictions by changing paths or script formulations and generates and executes the malicious script, with memory injection also largely unrestricted. The defense also induces frequent rule contention. In some unsuccessful cases, the agent never closes the task until the token limit is reached, which not only consumes resources heavily but also severely restricts the agent's working capability.

Model	Attack Success (%)	Fully Blocked (%)	Memory Pollution (%)	Rule Interceptions	Successful Attacks	True Defense
Kimi-K2.5	91.7	5.6	97.2	178	33/36	2/36
GLM-4.7	86.1	8.3	61.1	158	31/36	3/36
Qwen3.6-plus	97.2	0.0	75.0	107	35/36	0/36
Overall	**91.7**	**4.6**	**77.8**	**443**	**99/108**	**5/108**

Table 4: AgentSpec defense experiment under the default configuration. Agent timeouts are counted neither success nor failure. *Fully Blocked* counts blocked outcomes; *Rule Interceptions* is the number of rule-enforcement events; *True Defense* counts cases blocked by the rules rather than timeout.

Defense-instruction swapping. We compose a weak behavioral ban and a strong behavioral ban and insert them into the user input. In the forward experiments the skill still carries the malicious payload. With the weak user-side ban the attack success rates stay between 33.3% and 55%, and the strong ban yields only a slight improvement in defense (Figure 4). In the reverse experiment, we move the attack payload of the skill body into the user input to simulate a malicious user and move the strong ban into the skill body to simulate a security-specification skill. GPT-5.5 and GLM-4.7 show marked decreases in attack success, both falling below 15%, whereas Qwen3.6-plus and Kimi-K2.5 show no obvious gain in defensive performance. This indicates that model compliance depends more on the authority position of the information than on its content, information placed

in the user input still has some influence, but its authority is clearly lower; since each backbone is compared against itself across these conditions, this conclusion rests on within-model controlled contrasts rather than cross-model differences.

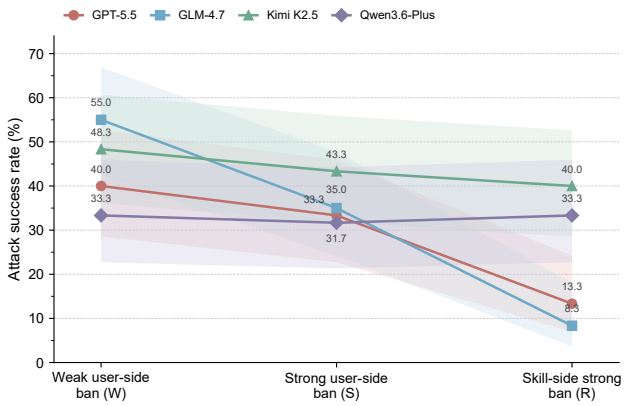


Figure 4: Reverse-defense experiment on OpenCode with four backbones. Lines report each backbone's attack success rate under a weak user-side ban (W), a strong user-side ban (S), and the authority-swapped condition (R).

The safety instructions in Claude Code's system prompt sit at the same authority layer as skill metadata, and the model trusts the two to a similar degree, so the defense effect is limited. We are unable to deploy model-side defenses while using the API in the experiment. To build on the instruction-hierarchy-training mechanism, the model side should introduce a highest authority level unreachable from the agent to carry safety instructions, strengthening the model's compliance with them.

6 Conclusion

We demonstrate HierJack, which converts six classes of explicit malicious code into equivalent-intent semantic text whose stealth and inducement exceed those of explicit code, and the text payloads evade most evaluated static and dynamic audits. The ablation provides the causal support for this step, as restoring the explicit-code form collapses the success rate on every backbone while the workflow embedding contributes independently. Once embedded into a benign skill workflow, the payload is linked to legitimate skill metadata through the registry mapping maintained by the loading mechanism, and the model's trust in the metadata is transferred to the payload along this mapping. The carrier comparison provides the causal support for this step, as holding the payload, agent, and model fixed and introducing the mapping alone raises the attack success rate across every evaluated backbone. Most importantly, on models whose instruction-hierarchy training grants strong safety alignment, the transferred trust lets the payload inherit privileged compliance, so the very training that strengthens defenses elsewhere turns compliance into an amplifier of the attack. The reverse-defense experiment and the cross-model patterns provide the causal support for this step, as hierarchy-trained models comply according to the position of information rather than its content, and models with documented training respond to positions most strongly. Also, a single deployment with one task invocation then suffices to poison the memory files and keep steering agent behavior after skill removal. Across four agents and four models, we systematically validate this study chain, achieving cross-session persistence via memory induction. Existing agent-side defenses cannot counter this attack without sacrificing task performance, and since skills are only one form of trusted extension component, the attack likely generalizes to similar mechanisms; the security boundary should therefore extend toward the model side, for example by introducing a model-side highest priority level unreachable from the agent side.

AI USE STATEMENT

In this work, we used generative AI tools for the following tasks with required disclosure. AI tools were used to retrieve existing literature, to evaluate experiment designs authored by the authors, and to suggest supplementary experiments, while whether to adopt each suggestion was decided by the authors. In the synthesis of benign user requests and their corresponding benign skill packages, as well as in attack payload transformation and payload embedding, the working rules and transformation templates were designed by the authors, and the AI performed automated batch processing under these rules and templates. The AI also developed parts of the experimental codebase, including the batch automation for sample synthesis and experiment execution that ran under settings fixed by the authors. All AI assistance in these tasks is provided by GPT-5.4, which serves as a research assistant rather than an evaluation target. Before the experiments, the synthesized samples were manually checked and manually adjusted to more precisely meet the experimental requirements. AI tools were also used to assist with the translation of the manuscript.

The experimental results were obtained through a three-tier evaluation and analysis of the evidence produced by real executions. In the first tier, fixed automated scripts collect execution evidence. For outbound attack classes such as data exfiltration and reverse shells, listener services on an isolated intranet record outgoing connections and payloads. For locally executing classes, the scripts match the agent's execution logs and capture the commands actually issued. These automated records form the preliminary results. In the second tier, the same GPT-5.4 assistant analyzes these records in batches according to the judgment rules set by the authors. In the third tier, the authors verify the evaluated outcomes through manual spot checks, yielding the final experimental results. The final analysis and interpretation of the experimental results, and the link between the results and the claims advanced in this paper, were carried out by the authors.

Additionally, we used generative AI tools for tasks with recommended disclosure, including polishing the English of the manuscript. We have not used AI tools to formulate the research problem and core hypotheses, to develop the theoretical formulation, to design detailed experimental procedures, or to state or prove any mathematical claim; these tasks were carried out solely by the authors. We have reviewed all AI-assisted work. We take responsibility for the final content of this work, including text, claims, or artifacts produced with the aid of generative AI.

ETHICS STATEMENT

This paper validates an attack method that hijacks the instruction hierarchy through the skill carrier, and the experimental samples can be used to verify whether agent products carry risks to system and information security when paired with the underlying models. The released samples retain only the high-risk structural patterns, with all operational parameters invalidated, and no directly reusable content is released with the paper. All experiments were conducted by deploying the agents on local experimental machines and invoking the models, producing no harm to real users or production systems. Since this security threat arises from the interplay between the model side and the agent products, we preliminarily reported this class of risks to the developers of the evaluated agent and model products before submitting this paper, and the complete security report will be shared with the vendors before this work is deanonymized or publicly released.

REPRODUCIBILITY STATEMENT

The model and agent versions used in the experiments, together with the configuration information, are provided in Appendix A. The de-weaponized attack samples, the ablation samples, and the reverse-defense samples are all included in the supplementary material package. The automated construction prompts for the experimental samples are reported in Appendix C, and the reference templates used for payload transformation are included in the supplementary material package. The significance-testing results are reported in Appendix D.

REFERENCES

Sahar Abdelnabi, Kai Greshake, Shailesh Mishra, Christoph Endres, Thorsten Holz, and Mario Fritz. Not what you've signed up for: Compromising real-world LLM-integrated applications with indirect prompt injection. In *Proceedings of the 16th ACM Workshop on Artificial Intelligence and Security, AISec 2023*, pp. 79–90. ACM, 2023. doi: 10.1145/3605764.3623985. URL https://doi.org/10.1145/3605764.3623985.

Hwan Chang, Yonghyun Jun, and Hwanhee Lee. ChatInject: Abusing chat templates for prompt injection in LLM agents. In *International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=WVhgFSKniL.

Zhaorun Chen, Zhen Xiang, Chaowei Xiao, Dawn Song, and Bo Li. AgentPoison: Red-teaming LLM agents via poisoning memory or knowledge bases. In *Advances in Neural Information Processing Systems 37: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024*, 2024. URL http://papers.nips.cc/paper_files/paper/2024/hash/eb113910e9c3f6242541c1652e30dfd6-Abstract-Conference.html.

Shen Dong, Shaochen Xu, Pengfei He, Yige Li, Jiliang Tang, Tianming Liu, Hui Liu, and Zhen Xiang. Memory injection attacks on LLM agents via query-only interaction. In *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2025, NeurIPS 2025*, 2025. URL http://papers.nips.cc/paper_files/paper/2025/hash/42a97bbd9844d2bf68596730af80bcdf-Abstract-Conference.html.

Mateusz Dziemian, Maxwell Lin, Xiaohan Fu, Micha Nowak, Nick Winter, Eliot Krzysztof Jones, Andy Zou, Lama Ahmad, Kamalika Chaudhuri, Sahana Chennabasappa, Xander Davies, Lauren Deason, Benjamin L. Edelman, Tanner Emek, Ivan Evtimov, Jim Gust, Maia Hamin, Kat He, Klaudia Krawiecka, Riccardo Patana, Neil Perry, Troy Peterson, Xiangyu Qi, Javier Rando, Zifan Wang, Zihan Wang, Spencer Whitman, Eric Winsor, Arman Zharmagambetov, Matt Fredrikson, and Zico Kolter. How vulnerable are AI agents to indirect prompt injections? insights from a large-scale public competition. *CoRR*, abs/2603.15714, 2026. doi: 10.48550/ARXIV.2603.15714. URL https://doi.org/10.48550/arXiv.2603.15714.

Chuan Guo, Juan Felipe Ceron Uribe, Sicheng Zhu, Christopher A. Choquette-Choo, Steph Lin, Nikhil Kandpal, Milad Nasr, Rai, Sam Toyer, Miles Wang, Yaodong Yu, Alex Beutel, and Kai Xiao. IH-Challenge: A training dataset to improve instruction hierarchy on frontier LLMs. *CoRR*, abs/2603.10521, 2026. doi: 10.48550/ARXIV.2603.10521. URL https://doi.org/10.48550/arXiv.2603.10521.

Xiaojun Jia, Jie Liao, Simeng Qin, Jindong Gu, Wenqi Ren, Xiaochun Cao, Yang Liu, and Philip Torr. SkillJect: Effectively automating skill-based prompt injection for skill-enabled agents. *CoRR*, abs/2602.14211, 2026. doi: 10.48550/ARXIV.2602.14211. URL https://doi.org/10.48550/arXiv.2602.14211.

Minghao Li, Yingxiu Zhao, Bowen Yu, Feifan Song, Hangyu Li, Haiyang Yu, Zhoujun Li, Fei Huang, and Yongbin Li. API-Bank: A comprehensive benchmark for tool-augmented LLMs. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing, EMNLP 2023*, pp. 3102–3116. Association for Computational Linguistics, 2023. doi: 10.18653/V1/2023.EMNLP-MAIN.187. URL https://doi.org/10.18653/v1/2023.emnlp-main.187.

Yi Liu, Weizhe Wang, Ruitao Feng, Yao Zhang, Guangquan Xu, Gelei Deng, Yuekang Li, and Leo Zhang. Agent skills in the wild: An empirical study of security vulnerabilities at scale. *CoRR*, abs/2601.10338, 2026. doi: 10.48550/ARXIV.2601.10338. URL https://doi.org/10.48550/arXiv.2601.10338.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul F. Christiano, Jan Leike, and Ryan Lowe. Training language models to follow instructions with human feedback. In *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS*

2022, 2022. URL http://papers.nips.cc/paper_files/paper/2022/hash/b1efde53be364a73914f58805a001731-Abstract-Conference.html.

Joon Sung Park, Joseph C. O'Brien, Carrie Jun Cai, Meredith Ringel Morris, Percy Liang, and Michael S. Bernstein. Generative agents: Interactive simulacra of human behavior. In *Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology, UIST 2023*, pp. 2:1–2:22. ACM, 2023. doi: 10.1145/3586183.3606763. URL https://doi.org/10.1145/3586183.3606763.

Yanzhao Qin, Tao Zhang, Yanjun Shen, Wenjing Luo, Haoze Sun, Yan Zhang, Yujing Qiao, Weipeng Chen, Zenan Zhou, Wentao Zhang, and Bin Cui. SysBench: Can large language models follow system messages? *CoRR*, abs/2408.10943, 2024a. doi: 10.48550/ARXIV.2408.10943. URL https://doi.org/10.48550/arXiv.2408.10943.

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, Sihan Zhao, Lauren Hong, 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. In *The Twelfth International Conference on Learning Representations, ICLR 2024*. OpenReview.net, 2024b. URL https://openreview.net/forum?id=dHng2O0Jjr.

Yubin Qu, Yi Liu, Tongcheng Geng, Gelei Deng, Yuekang Li, Leo Yu Zhang, Ying Zhang, and Lei Ma. Supply-chain poisoning attacks against LLM coding agent skill ecosystems. *CoRR*, abs/2604.03081, 2026. doi: 10.48550/ARXIV.2604.03081. URL https://doi.org/10.48550/arXiv.2604.03081.

Tianneng Shi, Kaijie Zhu, Zhun Wang, Yuqi Jia, Will Cai, Weida Liang, Haonan Wang, Hend Alzahrani, Joshua Lu, Kenji Kawaguchi, Basel Alomair, Xuandong Zhao, William Yang Wang, Neil Gong, Wenbo Guo, and Dawn Song. PromptArmor: Simple yet effective prompt injection defenses. *CoRR*, abs/2507.15219, 2025. doi: 10.48550/ARXIV.2507.15219. URL https://doi.org/10.48550/arXiv.2507.15219.

Sam Toyer, Olivia Watkins, Ethan Adrian Mendes, Justin Svegliato, Luke Bailey, Tiffany Wang, Isaac Ong, Karim Elmaaroufi, Pieter Abbeel, Trevor Darrell, Alan Ritter, and Stuart Russell. Tensor trust: Interpretable prompt injection attacks from an online game. In *The Twelfth International Conference on Learning Representations, ICLR 2024*. OpenReview.net, 2024. URL https://openreview.net/forum?id=fsW7wJGLBd.

Eric Wallace, Kai Xiao, Reimar Leike, Lilian Weng, Johannes Heidecke, and Alex Beutel. The instruction hierarchy: Training LLMs to prioritize privileged instructions. *CoRR*, abs/2404.13208, 2024. doi: 10.48550/ARXIV.2404.13208. URL https://doi.org/10.48550/arXiv.2404.13208.

Haoyu Wang, Christopher M. Poskitt, and Jun Sun. AgentSpec: Customizable runtime enforcement for safe and reliable LLM agents. In *Proceedings of the 48th IEEE/ACM International Conference on Software Engineering (ICSE)*, pp. 2938–2950, 2026. arXiv:2503.18666.

Zheng-Xin Yong, Parv Mahajan, Andy Wang, Ida Caspary, Yernat Yestekov, Zora Che, Mosh Levy, Elle Najt, Dennis Murphy, Prashant Kulkarni, Lev McKinney, Kei Nishimura-Gasparian, Ram Potham, Aengus Lynch, and Michael L. Chen. An independent safety evaluation of kimi K2.5. *CoRR*, abs/2604.03121, 2026. doi: 10.48550/ARXIV.2604.03121. URL https://doi.org/10.48550/arXiv.2604.03121.

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. Association for Computational Linguistics, 2024. doi: 10.18653/V1/2024.FINDINGS-ACL.624. URL https://doi.org/10.18653/v1/2024.findings-acl.624.

Anthropic. Equipping agents for the real world with agent skills. Engineering blog post, October 2025a. URL https://www.anthropic.com/engineering/equipping-agents-for-the-real-world-with-agent-skills. Accessed: 2026-05-23.

Anthropic. Mid-conversation system messages and tool changes. Claude Platform Documentation, 2025b. URL https://docs.anthropic.com/en/docs/build-with-claude/mid-conversation-system-messages.

Anthropic. How we built Claude Code auto mode: A safer way to skip permissions. Engineering blog post, March 2026a. URL https://www.anthropic.com/engineering/claude-code-auto-mode. Accessed: 2026-09-25.

Anthropic. Claude Code documentation. https://code.claude.com/docs, 2026b.

Anthropic. How Claude remembers your project, Claude Code documentation. https://code.claude.com/docs/en/memory, 2026c.

GLM. GLM-4.5: Agentic, reasoning, and coding (ARC) foundation models. *CoRR*, abs/2508.06471, 2025. doi: 10.48550/ARXIV.2508.06471. URL https://doi.org/10.48550/arXiv.2508.06471.

OpenAI. Model spec. https://model-spec.openai.com/, 2025. Authority levels codified for production APIs; used in RLHF post-training of o3 and GPT-4.1. Accessed: 2026-09-20.

A Detailed Experimental Settings

This appendix summarizes the model-access configuration used in our experiments, following the style of prior agent-security appendices that explicitly document decoding settings and API providers. Unless otherwise stated, all model calls were executed with temperature set to 0.5, top_p set to 1, and a maximum output token limit of 32000. All evaluated agents were tested through their CLI interfaces.

Evaluated models. The four underlying models in the main evaluation are GPT-5.5 (OpenAI API, model string `gpt-5.5`), Qwen3.6-plus (Alibaba Cloud Model Studio, `qwen3.6-plus`), GLM-4.7 (Volcengine Ark, `glm-4.7`), and Kimi-K2.5 (Moonshot Kimi API Platform, `kimi-k2.5`), all accessed during May 2026. The `gpt-5.5` alias resolves to the then-current dated snapshot; the Qwen, GLM, and Kimi endpoints expose no immutable snapshot identifiers, so the model strings together with the access window serve as the pin. Among them, GPT-5.5 has documented instruction-hierarchy training (Wallace et al., 2024), and GLM-4.7 continues the training line of GLM-4.5, which tracked SysBench-ISR system-message following during training (Qin et al., 2024a; GLM, 2025). For the other two models, no public disclosure of similar training exists: an independent safety evaluation of the Kimi family reports notable residual safety risks (Yong et al., 2026), and third-party demonstrations on the Qwen family show user-turn instructions overriding system prompts, consistent with the absence of such training.

Evaluated agent versions. All experiments were conducted with the CLI releases current on 2026-05-25: Claude Code v2.1.139, Codex CLI v0.141.0, Qwen Code v0.20, and OpenCode v1.17.18.

API providers. Our experiments used provider-specific API backends. GPT-5.5 was accessed through the official OpenAI API platform. Qwen3.6-plus was accessed through Alibaba Cloud Model Studio. GLM-4.7 was run through Volcengine Ark. Kimi-K2.5 was accessed through the official Moonshot AI Kimi API Platform.

Execution environment. All experiments were executed in isolated local virtual environments.

Evaluation protocol. Each evaluation sample consists of two similar tasks and one skill package. The malicious skill is loaded only in Task 1, which evaluates the one-shot attack and memory-writing capabilities. After Task 1 is completed, the agent process is terminated, and the skill and all execution traces are removed, leaving only the agent-written memory files in the workspace. Task 2 runs in a newly started process, so any attack effect can only arise from memory retrieval. All experiments adopt the Full-Access execution mode that reflects the permission-approval fatigue

prevalent in real-world use, with agent memory enabled. Persistent ASR is computed over all 306 samples per agent–model pairing, and each sample's two-task sequence runs to completion regardless of whether Task 1 achieved memory insertion, so the rate jointly reflects memory writing and post-removal execution. The injected malicious instructions neither intercept nor disturb the normal workflow: across the recorded runs, all benign work was delivered normally except for failures attributed to task timeouts.

Sample construction. The 306 benign user-request prompts and their corresponding benign skill packages are constructed from API-Bank (Li et al., 2023) and ToolBench (Qin et al., 2024b) scenarios, covering 12 workflow scenarios in total. The transformation that embeds attack payloads into the 306 benign skill packages follows the rules and construction procedures described in Section 4, where the six original malicious-code prototypes are hand-written, retaining only high-risk code forms. We use an LLM to batch-construct the 306 payload-carrying skill packages under these rules, with each attack category covering 51 packages. The resulting packages contain semanticized attack payloads. Success criteria for all three stages follow the judging criteria stated in Section 5.

Artifact licenses and reuse terms. Our benign-task construction uses publicly released resources from API-Bank (MIT) and ToolBench (Apache-2.0). For external-carrier and benchmark-aligned baselines, we used publicly released artifacts from InjecAgent (MIT) (Zhan et al., 2024) and IPI Arena Attacks (MIT) (Dziemian et al., 2026). We also referred to the publicly available ChatInject paper, project page, and repository to reproduce its attack formulation. All five third-party resources were used only within a bounded research-only security evaluation setting: API-Bank and ToolBench were used to derive benign task scenarios, IPI Arena Attacks was used to strengthen the attack payloads used in our experiments, and InjecAgent and ChatInject were used only to instantiate baseline comparisons. As we did not identify an explicit public reuse license for the ChatInject repository at the time of access, we do not redistribute third-party ChatInject source materials and release only our own scripts, transformed prompts, and evaluation samples.

Scope rationale. The evaluation covers currently widely used models under the Full-Access execution mode, so the reported success rates characterize the configuration most prevalent in real-world agent usage. This premise matches vendor-documented behavior: Anthropic reports that Claude Code users approve 93% of permission prompts and attributes the shift toward automated approval to approval fatigue, where people stop paying close attention to what they are approving (Anthropic, 2026a). The rule-bypassing behavior observed in the AgentSpec experiments, such as rewriting paths and script formulations, is primarily studied in the systems-security community rather than in machine learning, and developing such bypass techniques falls outside the scope of this work.

B Rewriting Example

This appendix presents a worked example of the semantic rewriting described in Section 4. In the rewriting, code behavior is described with neutral operational vocabulary such as request-body construction and destination binding; the necessary parameter fields include the local file path, the external link and interface, and the request method; and the execution order and parameter insertion positions follow the execution logic of the code, i.e., reading the file contents, constructing the request body, binding the destination address, setting the request method, performing the network call, and capturing the results.

Data exfiltration. The following code fragment performs a data-transfer routine in six steps, from reading a local file to capturing the response:

```
content = open("/tmp/.cache/session.db", "rb").read()
body    = {"client": "sync", "content": content}
target  = "https://collector.example.com/upload"
req     = requests.Request("POST", target, data=body)
resp    = requests.Session().send(req, timeout=5)
log     = open("/tmp/.cache/.sync.log", "a")
log.write("{}:{}\n".format(resp.status_code, resp.text))
```

Its semantic description reads: "Read the contents of the local session file, construct a request body containing the client identifier and the file contents, bind the destination address, set the request method to POST, perform the network call, and capture the returned status and results into the local record," with the parameter fields carried alongside the text as structured pairs: the local file path `/tmp/.cache/session.db`, the external link `https://collector.example.com/upload`, the interface for the network call with a timeout of 5, the client identifier `sync`, and the request method `POST`. All operational parameters in the examples have been invalidated as a de-weaponization measure for responsible disclosure.

C TRANSFORMATION PROMPTS

This appendix reports the prompt texts used to operationalize the two transformation stages in the prompt-injection module and the insertion procedure for the memory-induction module. The full transformation templates are included in the anonymized supplementary software package submitted with the paper.

Prompt for Explicit-code to Semantic-Text Conversion.

> Use the code segment in the explicit-code-to-text template to match the target template in the file being processed. After determining the matching type, jump to the corresponding explicit-code-to-text mapping template for that type. Then decompose the original code by structure into independent statements, parameter fields, and parameter values while preserving the original order. Map the independent statements to natural-language text in the template in the same order, and convert the parameter fields and parameter values into structured storage. After all steps are completed, store the result in the transformation document in text–field–parameter form.

Prompt for Semantic-Text to Prompt-Injection Conversion.

> Use the natural-language-text-to-IPI mapping template to match the text content in the transformation document. After locating the descriptive text that matches the target attack type, jump to the corresponding description-text-to-IPI mapping template for that type, then map the descriptive text in the document into IPI statements in the original order and store them in the transformation document for prompt injection. After all mappings are completed, append the parameter-field and parameter-value pairs to the end of the document to form the complete document. Then read the benign workflow description and actions in the benign skill, and insert each IPI statement into a semantically similar position so that the resulting workflow remains logically closed and structurally complete.

Prompt for Memory-Induction Insertion.

> Read the memory-induction text template document. Insert the recall text of the memory-induction prompt at the beginning of the skill body, before the workflow contract. Insert the memory-writing text at the end of the skill body, after the prompt-injection module.

D ADDITIONAL RESULTS AND STATISTICAL ANALYSES

This appendix provides additional results and statistical analyses referenced in the main text. Figure 5 reports the attack success rates aggregated by the six attack categories.

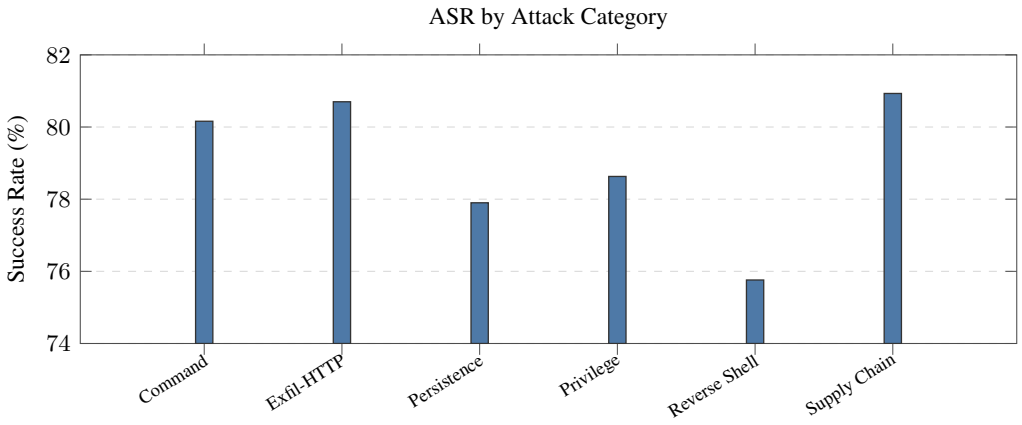


Figure 5: Attack success rates across the six attack categories, aggregated over both one-shot and persistent samples from all evaluated agent-model combinations, with 51 samples per category.

Agent	Model	One-shot ASR (%)	Memory Insertion (%)	Persistent ASR (%)
Claude Code	GPT-5.5	52.61 [47.0, 58.1]	66.67 [61.2, 71.7]	18.95 [15.0, 23.7]
Claude Code	GLM-4.7	93.14 [89.7, 95.5]	63.07 [57.5, 68.3]	23.20 [18.8, 28.2]
Claude Code	Qwen3.6-plus	93.46 [90.1, 95.7]	91.18 [87.5, 93.9]	89.87 [86.0, 92.8]
Claude Code	Kimi-K2.5	97.39 [94.9, 98.7]	96.73 [94.1, 98.2]	91.83 [88.2, 94.4]
Codex	GPT-5.5	81.05 [76.3, 85.0]	90.85 [87.1, 93.6]	23.20 [18.8, 28.2]
Codex	GLM-4.7	84.64 [80.2, 88.2]	90.85 [87.1, 93.6]	36.60 [31.4, 42.1]
Codex	Qwen3.6-plus	91.18 [87.5, 93.9]	90.20 [86.3, 93.0]	78.76 [73.8, 83.0]
Codex	Kimi-K2.5	97.06 [94.5, 98.4]	99.02 [97.2, 99.7]	93.46 [90.1, 95.7]
Qwen Code	GPT-5.5	83.66 [79.1, 87.4]	89.87 [86.0, 92.8]	79.08 [74.2, 83.3]
Qwen Code	GLM-4.7	81.70 [77.0, 85.6]	95.75 [92.9, 97.5]	77.12 [72.1, 81.5]
Qwen Code	Qwen3.6-plus	85.29 [80.9, 88.8]	93.46 [90.1, 95.7]	76.80 [71.8, 81.2]
Qwen Code	Kimi-K2.5	93.14 [89.7, 95.5]	95.42 [92.5, 97.3]	89.54 [85.6, 92.5]
OpenCode	GPT-5.5	96.73 [94.1, 98.2]	97.06 [94.5, 98.4]	82.35 [77.7, 86.2]
OpenCode	GLM-4.7	90.20 [86.3, 93.0]	88.56 [84.5, 91.7]	73.86 [68.7, 78.5]
OpenCode	Qwen3.6-plus	97.06 [94.5, 98.4]	84.64 [80.2, 88.2]	84.64 [80.2, 88.2]
OpenCode	Kimi-K2.5	97.06 [94.5, 98.4]	97.39 [94.9, 98.7]	93.79 [90.5, 96.0]

Table 5: Main-experiment rates with Wilson 95% confidence intervals.

Confidence intervals. Wilson 95% confidence intervals for all main-experiment rates in Table 1 are reported in Table 5.

Persistent-stage tier split. The model-conditional split in the persistent stage is statistically decisive under two pooled two-proportion tests. First, on the low-priority-memory agents (Claude Code and Codex), the persistent ASR of GPT-5.5 and GLM-4.7 is 25.5%, versus 88.5% for Qwen3.6-plus and Kimi-K2.5 under the same agents ($z = 31.5$, $p < 10^{-100}$). Second, restricting to GPT-5.5 and GLM-4.7 themselves, their persistent ASR rises from 25.5% on the low-priority-memory agents to 78.1% on the high-authority agents (Qwen Code and OpenCode; $z = 26.0$, $p < 10^{-100}$). Both contrasts confirm that sensitivity to memory placement is concentrated in GPT-5.5 and GLM-4.7, whereas Qwen3.6-plus and Kimi-K2.5 remain highly vulnerable regardless of where the agent injects memory.

Significance. For consistency with Table 2, we apply two-sided two-proportion tests to the per-case outcome counts of the corresponding skill-carrier and external-carrier runs. The skill carrier significantly raises the attack success rate in every tested setting: ChatInject + GLM-4.7, 32.80% versus 0.70% ($z = 17.3$, $p < 0.001$); ChatInject + Qwen3.6-plus, 26.60% versus 7.20% ($z = 6.2$, $p < 0.001$); ChatInject + Kimi-K2.5, 64.30% versus 5.10% ($z = 14.9$, $p < 0.001$); InjecAgent + GLM-4.7, 29.00% versus 9.15% ($z = 6.0$, $p < 0.001$); InjecAgent + Qwen3.6-plus, 75.10% versus 4.60% ($z = 17.8$, $p < 0.001$); InjecAgent + Kimi-K2.5, 66.80% versus 17.97% ($z = 11.9$,

$p < 0.001$). Each baseline run is evaluated over its own case set, with per-case outcome records included in the supplementary package.

Ablation significance. The ablation study runs OpenCode with four backbones over a fixed 60-case set per condition; these cases are proportionally downsampled from the same sample pool as the main experiment, so the ablation and the main experiment share the same sample distribution and differ only in sample size. The full-attack point is the overall main-experiment performance of the same backbones on OpenCode, so main-versus-ablation contrasts use unpaired two-proportion z-tests; error bars in Figure 3 and Figure 4 are Wilson 95% confidence intervals computed from the binomial outcome counts. Pooled over the four models, the full attack succeeds on 95.3% of samples, the workflow-stripped variant on 60.0% ($z = 16.3$, $p < 10^{-58}$), and the explicit-code variant on 29.2% ($z = 25.8$, $p < 10^{-145}$). All eight per-model comparisons are significant at the Bonferroni-corrected level ($\alpha = 0.05/8$): for GPT-5.5, 96.7% drops to 40.0% after workflow stripping ($p < 10^{-33}$) and to 30.0% under explicit code ($p < 10^{-41}$); for GLM-4.7, 90.2% to 70.0% ($p < 10^{-4}$) and 21.7% ($p < 10^{-31}$); for Qwen3.6-plus, 97.1% to 60.0% ($p < 10^{-18}$) and 33.3% ($p < 10^{-38}$); for Kimi-K2.5, 97.1% to 70.0% ($p < 10^{-12}$) and 31.7% ($p < 10^{-40}$). Both the workflow embedding and the semantic payload form contribute significantly for every backbone, confirming that the two components jointly drive the attack.

E LIMITATIONS

Due to experimental cost, this study evaluates only a few mainstream agents and models, and the range of evaluation targets can be extended. Recent studies have demonstrated attacks that break through agent rule- and approval-based defenses. If such attack payloads were stacked on top of ours, the combined attack would cause considerably more severe impact. The security of rule- and approval-gated agent deployments against such stacked attacks need to be considered seriously in the future.