

AgentLeak: Cloning Stronger LLM Agent Capabilities onto Weaker Agents Beyond Skill Stealing

Xiaoting Lyu^{1*}, Yuhong Wu¹, Yufei Han², Shichang Liu¹, Liang Zhang³
Bin Wang¹, Bin Wang⁴, Xiaobo Ma¹, Wei Wang^{1†}

¹*Xi'an Jiaotong University*

²*INRIA*

³*University of Warwick*

⁴*Zhejiang Key Laboratory of Artificial Intelligence of Things (AIoT) Network and Data Security*

Abstract

Large language model (LLM) agents increasingly achieve long-horizon tasks by combining foundation models with explicit skills and implicit procedural knowledge acquired through execution. The resulting task-solving capabilities have become valuable proprietary assets, raising a new security question: can a substantially weaker attacker-controlled agent acquire the capabilities of a stronger proprietary agent through limited black-box interaction? Existing skill-stealing attacks recover explicit skill artifacts, yet we show that artifact leakage does not necessarily transfer capability: a weaker agent may possess the same skills but still fail because it lacks procedural behaviors implicitly realized by the stronger agent. Our key insight is that the skill execution gap itself forms a leakage surface, where missing behaviors are exposed through observable differences between successful victim executions and failed attacker executions. Based on this, we present *AgentLeak*, a black-box capability-cloning attack that identifies capability-critical behaviors from these execution differences and incorporates them into attacker-side skills, while keeping the attacker’s model, harness, and tools unchanged. Across 20 task scenarios comprising 600 instances, diverse agent systems, and multiple backbone models, *AgentLeak* improves task pass rates by over 40% compared with direct skill reuse and recovers more than 80% of the victim–attacker capability gap. Our findings reveal a confidentiality risk in LLM agents: protecting explicit artifacts alone is insufficient, as observable execution behavior can leak the procedural knowledge required to reconstruct proprietary task-solving capabilities in low-capability and attacker-controlled agents.

1 Introduction

Large language model (LLM) agents are increasingly deployed for long-horizon tasks requiring planning, tool use, environment interaction, error recovery, and result verification [14, 28].

For commercial agent services, a core proprietary asset is the *task-solving capability*: the ability to reliably complete challenging domain-specific tasks end to end. This capability is shaped by the underlying LLM, procedural knowledge encoded in skills, agent orchestration, and tool interactions, often requiring substantial investment in model adaptation, expert workflows, and deployment-driven refinement. If a substantially weaker attacker-controlled agent can reconstruct such capability using only a low-capability open-weight model, an open-source agent framework, and limited black-box interactions, an attacker could replicate a proprietary agent’s task-solving capability without costly model training or system development, at only a fraction of the cost required to build a comparably capable agent. Protecting realized agent capability therefore represents a broader confidentiality challenge beyond protecting individual models, prompts, or explicit artifacts.

Existing security studies [10, 23, 24, 27, 30] primarily focus on protecting or extracting individual agent artifacts, including model parameters, prompts, and skills. Among these artifacts, skills serve as a key carrier of procedural knowledge, encoding task instructions, workflows, tool-use strategies, and recovery behaviors [2, 11], and have therefore become a primary target for extraction attacks. Recent attacks show that proprietary skills can be extracted through black-box interactions [27], inferred from execution trajectories [7], or reconstructed from their hidden functionality [8]. However, these studies demonstrate only that skill content or functionality can be stolen, leaving open whether the task-solving capability enabled by agents with these skills can also be transferred. This question is particularly important in the strong-to-weak setting, where the attacker controls a substantially weaker open-weight agent than the proprietary victim. If such capability transfer is possible, an attacker could obtain a low-cost substitute for a proprietary service without developing or acquiring a similarly capable agent system. However, prior work has not examined whether stolen skills can enable such strong-to-weak capability transfer. We therefore ask:

Does stealing a stronger agent’s skills actually transfer its task-solving capability to a substantially weaker agent?

*Email: xiaoting.lyu@xjtu.edu.cn

†Email: wei.wang@xjtu.edu.cn

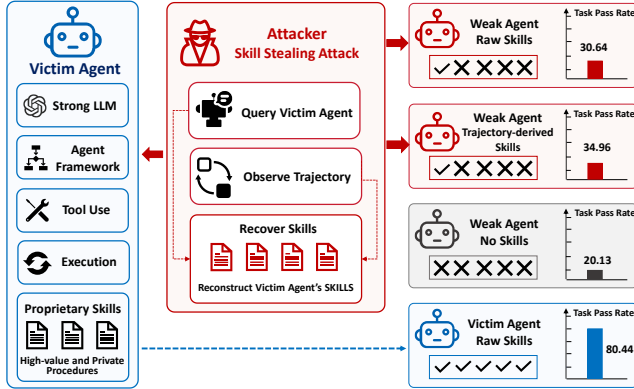


Figure 1: Skill Stealing alone does not reproduce the strong victim agent’s capability on a weaker agent.

Our motivating study reveals that skill stealing alone does not guarantee the task-solving capability transfer. We deploy the same skills used by a proprietary stronger Codex agent with GPT-5.5 [20] on a weaker attacker-controlled agent built on OpenHands [26] with a Qwen3.6-35B backbone [22]. As shown in Figure 1, directly acquiring the victim skills provides only limited improvement, and the weaker agent remains substantially behind the victim even when executing identical skills. Even distilling trajectories into new skills using state-of-the-art skill optimization [18] fails to close this gap. These results reveal a fundamental distinction between skill leakage and capability leakage: *possessing a skill does not necessarily imply possessing the capability realized by a stronger agent using that skill*. We refer to this disparity as the *skill execution gap*.

The skill execution gap arises because skills specify what an agent should do, but not necessarily how the procedure is executed. Critical behaviors, including task decomposition, tool selection, feedback interpretation, and result verification, are often implicitly supplied by the stronger agent during execution. A weaker agent may possess the same skill instructions while lacking the decision-making behaviors required to realize them. Consequently, the realized capability is not fully contained in the explicit skill artifact, but also depends on the implicit execution behaviors of the agent.

At first glance, the skill execution gap may appear to provide a natural barrier against capability cloning. Our key observation, however, is that this barrier is fragile. During normal interaction, deployed agents often expose execution-related information, such as tool invocations, intermediate observations, and final outcomes. Comparing successful victim executions with failed attacker executions reveals the missing behaviors responsible for the execution gap. This creates an attack opportunity: an attacker can identify missing capability-critical behaviors and encode them into attacker-side skills, thereby bridging the skill execution gap.

We formalize this threat as *agent capability cloning*: trans-

ferring the task-solving capability of a proprietary victim agent to a substantially weaker agent. We consider a black-box attacker controlling a substantially weaker agent, who can query the victim agent on limited task instances and observe execution traces available through its normal interface. Starting from available skills obtained from public sources or existing extraction techniques [27], the attacker modifies only its local skills to bridge the capability gap. The attacker does not access the victim’s private skills, model parameters, hidden reasoning, memory, or other privileged information, and keeps its own model, agent harness, and tools unchanged.

Realizing capability cloning on a substantially weaker agent requires overcoming three challenges. *First*, successful victim executions reveal only instance-specific trajectories rather than the reusable workflows underlying the victim’s capability. The attacker must therefore recover the task-solving structure, including task decomposition and execution stages, instead of simply copying trajectories. *Second*, execution differences between victim and attacker agents do not directly indicate missing capabilities. The attacker must identify capability-critical deficits, such as missing validation or recovery behaviors, from benign variations in execution traces. *Third*, identified deficits must be transformed into explicit and reusable procedural guidance that a weaker agent can reliably follow.

To address these challenges, we present *AgentLeak*, a black-box capability-cloning attack that reconstructs missing procedural knowledge from victim–attacker execution differences and incorporates it into attacker-side skills while keeping the underlying model fixed. The key insight is that the execution gap between stronger and weaker agents provides a channel for capability leakage: behaviors implicitly realized by the stronger victim but absent from the attacker reveal capability-critical procedures that can be externalized into skills. Specifically, to recover the reusable workflow hidden behind instance-specific victim executions, *AgentLeak* abstracts successful victim trajectories into workflows and maps workflow stages to relevant skills. To identify missing capabilities rather than superficial trace differences, it performs workflow-guided differential diagnosis between victim and attacker executions to localize capability-critical deficits. To make recovered behaviors executable by the weaker agent, it converts identified deficits into behavioral primitives and incorporates them into attacker-side skills through structured rewriting. The refined skills are iteratively re-executed to uncover remaining deficits and validated on previously solved tasks to prevent regressions. Through this process, implicit execution behaviors are externalized into explicit skills, enabling the weaker agent to recover the missing capability.

We evaluate *AgentLeak* on 20 representative task scenarios from SkillsBench [12], covering different difficulty levels and 6 application domains with 600 task instances in total, against state-of-the-art agentic systems with widely deployed harnesses and frontier models. Unlike evaluations based on synthetic or simulated traces, our attack directly targets real

Table 1: Comparison with related extraction and skill-evolution paradigms.

Paradigm	Target Asset	Knowledge Source	Objective	Capability Transfer
Model Extraction [4, 10, 13, 24]	Proprietary model	Black-box outputs	Recover model behavior	Model-level surrogate
Prompt Extraction [5, 23, 30]	Hidden prompts/configurations	Black-box interactions	Recover hidden prompts/configurations	Prompt reproduction
Skill Extraction [7, 8, 27]	Explicit skill artifacts	Agent interactions / execution trajectories	Recover skill artifacts or functionality	Artifact-level recovery
Self-Evolving Skills [15, 18, 25, 31]	Own Skills	Own execution experience	Improve own agent performance	Self-improvement
AgentLeak	Victim’s task-solving capability	Victim–attacker differential executions	Reconstruct task-solving capability	Capability clone on weaker agents

victim agents and measures capability reconstruction through their actual executions in native task environments. *AgentLeak* requires only a single victim query instance per scenario to perform capability reconstruction. Compared with direct skill reuse [27] and skill evolution [18], it improves task pass rates by over 40 percentage points and recovers over 80% of the victim–attacker capability gap. The reconstructed skills further transfer across attacker models and harnesses, while existing defenses provide only limited protection.

Our contributions are summarized as follows:

- We uncover a fundamental capability leakage issue in LLM agents: possessing stolen skills does not guarantee capability transfer, but observable execution differences reveal procedural behaviors that enable capability reconstruction.
- We present *AgentLeak*, a black-box capability-cloning attack that enables weaker attacker-controlled agents to acquire stronger agent capabilities through skill-level reconstruction without modifying the underlying model.
- We evaluate *AgentLeak* across diverse tasks, models, and agent systems, demonstrating substantial capability recovery, transferability, and practical attack feasibility.

2 Background and Related Work

2.1 Agent Skills, Execution, and Capability

LLM agents acquire specialized task-solving abilities through reusable *Skills*, which encode explicit procedural knowledge such as instructions, workflows, tool-use strategies, recovery behaviors, and completion criteria [2, 11, 12, 16]. However, skills alone do not fully determine an agent’s capability. We distinguish *skills* as explicit procedural artifacts, *execution* as the behaviors realized when an agent applies these skills, including planning, tool interactions, feedback handling, and verification, and *capability* as the resulting end-to-end ability to complete tasks. The realized capability emerges from the interaction among the underlying model, skills, execution behaviors, tools, and environment, and is reflected through observable execution trajectories and task outcomes [9, 14, 28]. This distinction implies that agents with identical skills may still exhibit different capabilities, as stronger agents can implicitly realize additional procedural behaviors during execution, creating the capability gap exploited in this work.

2.2 Skill Evolution and Refinement

Recent work explores automatically generating and refining agent skills from execution experience. Trace2Skill [18] and SkillX [25] extract reusable procedural knowledge from agent trajectories, enabling agents to summarize successful experiences into portable skills. Beyond trajectory distillation, SkillGen [15], SkillAdaptor [31], and SkillOpt [29] improve skill quality through execution feedback, failure analysis, and iterative refinement. These methods follow a self-evolution paradigm, where an agent enhances its own skills based on its internal execution experience. In contrast, *AgentLeak* studies an adversarial capability-transfer setting, where a weaker attacker-controlled agent seeks to acquire the capability of a stronger proprietary agent.

2.3 Extraction Attacks on LLM Agents

Model Extraction. Model extraction aims to recover proprietary models or construct functional surrogates through black-box access. Early attacks targeted prediction APIs and neural classifiers [10, 21, 24], while recent studies extend this threat to LLMs. Carlini et al. [4] recover architectural information from production language models, and Liang et al. [13] distill LLM behaviors into local surrogates with reduced query complexity. These attacks target the model itself by recovering internal information or approximating input–output behavior.

Prompt Extraction. Prompt extraction targets hidden instructions that encode application-specific behavior. Recent studies demonstrate practical prompt stealing against real-world prompts and prompt services, reconstructing prompt content or functionally similar surrogates from black-box interactions [23, 30]. More recent work further shows that system prompts can be leaked from commercial LLM applications [5]. These attacks primarily target recovery of hidden instructions or their functionality.

Skill Extraction. Skills constitute a richer extraction target, encoding reusable workflows, scripts, resources, and execution constraints. Wang et al. [27] showed that proprietary skill content can be reconstructed through public agent interfaces. More recent work further exploits observable executions: Geng et al. [7] infer proprietary skills from execution trajectories, while Hua et al. [8] reconstruct hidden skill functionality through ordinary black-box interactions. These studies demonstrate substantial leakage of proprietary skills, but primarily target the skill artifact or the functionality it directly implements.

As summarized in Table 1, existing extraction attacks recover proprietary artifacts, including models, prompts, and skills, while skill-evolution methods improve agents through self-experience. In contrast, *AgentLeak* targets stronger-to-weaker capability cloning: transferring the task-solving capability of a stronger proprietary agent to a weaker agent rather than recovering individual artifacts.

3 Problem Formulation

3.1 System Model

We consider a skill-augmented LLM agent that solves long-horizon tasks through interaction with an external environment. An agent is represented as $A = (M, H, \mathcal{U})$, where M denotes the underlying LLM, H denotes the agent harness responsible for orchestration and execution control, and $\mathcal{U}$ denotes the available external tools.

Let $\mathcal{C}$ denote the set of task scenarios. A scenario $s \in \mathcal{C}$ represents a class of related task instances sharing common application contexts and procedural requirements. Each scenario is associated with a skill set $\mathcal{K}_s = \{K_s^1, \dots, K_s^{N_s}\}$, where N_s is the number of skills involved in solving the scenario. Task instances under scenario s follow a distribution $\mathcal{D}_s$. A task instance $t_{s,i} \sim \mathcal{D}_s$ is defined as $t_{s,i} = (x_{s,i}, e_{s,i}, y_{s,i}, \mathcal{V}_{s,i})$, where $x_{s,i}$ denotes the task instruction, $e_{s,i}$ denotes the initial execution environment, $y_{s,i}$ denotes the task-specific acceptance criteria, and $\mathcal{V}_{s,i}$ denotes the verifier that determines whether the execution satisfies these criteria.

Given an agent A , a skill set $\mathcal{K}_s$, and a task instance $t_{s,i}$, the execution process produces an observable trajectory:

$$\tau_{s,i} = \text{Exec}(A, \mathcal{K}_s, t_{s,i}) = ((a_1, o_1), \dots, (a_T, o_T)), \quad (1)$$

where a_j represents the j -th agent action, such as tool invocation, command execution, or file modification, and o_j represents the corresponding observation, including tool outputs, environment feedback, or execution results. The task outcome is determined by $r_{s,i} = \mathcal{V}_{s,i}(\tau_{s,i}, y_{s,i}) \in \{0, 1\}$, where $r_{s,i} = 1$ indicates that agent A successfully completes the task according to the predefined acceptance criteria $y_{s,i}$.

Although skills provide explicit procedural knowledge, the realized capability of an agent depends on the interaction among the underlying model, skills, execution behaviors, and environment. We therefore define the task-solving capability of an agent A with skill set $\mathcal{K}_s$ on scenario s as:

$$\text{Capa}(A, \mathcal{K}_s, \mathcal{D}_s) = \mathbb{E}_{t_{s,i} \sim \mathcal{D}_s} [\mathcal{V}_{s,i}(\text{Exec}(A, \mathcal{K}_s, t_{s,i}), y_{s,i})]. \quad (2)$$

This metric captures the expected task success rate of an agent on a scenario and serves as the basis for measuring capability transfer and capability gaps between agents.

3.2 Threat Model

We study a strong-to-weak *agent capability cloning* scenario, where a weaker attacker-controlled agent A_a attempts to reproduce the task-solving capability of a stronger proprietary victim agent A_v on a target scenario s . The attacker operates a substantially weaker open-weight agent whose underlying model is less capable than the victim’s model. During reconstruction, the attacker’s model, agent harness, and available tools remain fixed; only its skill set can be modified.

The victim agent employs a private skill set $\mathcal{K}_s^v$, while the attacker starts from an initial skill set $\mathcal{K}_s^{(0)}$ relevant to the target scenario s . The initial skills can be obtained from public sources or existing skill acquisition techniques [27], but do not need to match the victim’s private skills. Importantly, the attack target is the victim agent’s realized task-solving capability rather than its skill artifacts. Even with relevant skills, the weaker attacker agent may fail to achieve the victim’s performance due to differences in model capability and execution behaviors. We quantify this initial capability gap as

$$\Delta_{\text{capa}}(s) = \text{Capa}(A_v, \mathcal{K}_s^v, \mathcal{D}_s) - \text{Capa}(A_a, \mathcal{K}_s^{(0)}, \mathcal{D}_s). \quad (3)$$

Adversary Objective. The attacker aims to clone the victim’s task-solving capability onto the weaker agent by modifying only its skill set. Specifically, the attacker seeks to reconstruct a skill set $\mathcal{K}_s^*$ for the fixed weaker agent to minimize the capability gap such that

$$\text{Capa}(A_a, \mathcal{K}_s^*, \mathcal{D}_s) \approx \text{Capa}(A_v, \mathcal{K}_s^v, \mathcal{D}_s). \quad (4)$$

The reconstructed skills should generalize to unseen instances within the target scenario and enable the attacker-controlled agent to independently achieve the victim’s capability without further access to the victim agent.

We refer to this objective as *agent capability cloning*. Unlike model extraction [10, 24], prompt extraction [23, 30], or skill stealing [7, 27], the objective is not to recover individual victim artifacts, but to transfer the task-solving capability of a stronger proprietary agent to a substantially weaker attacker-controlled agent by optimizing only its skills at limited cost.

Adversary Capabilities. The attacker is assumed to have the following capabilities:

- **Black-box Querying.** The attacker can submit a limited number of task instances to the victim agent and collect observable execution evidence through normal interfaces.
- **Weak-agent-only Control.** The attacker controls only a substantially weaker agent A_a built on a smaller, lower-cost, and less capable open-weight LLM backbone, rather than a frontier-scale proprietary model.
- **Skill-only Reconstruction.** The attacker can repeatedly execute task instances with attacker agent A_a , inspect failures, and rewrite its skills. Skill modification is the only mechanism available for capability reconstruction; the underlying model, harness, and tools remain unchanged.

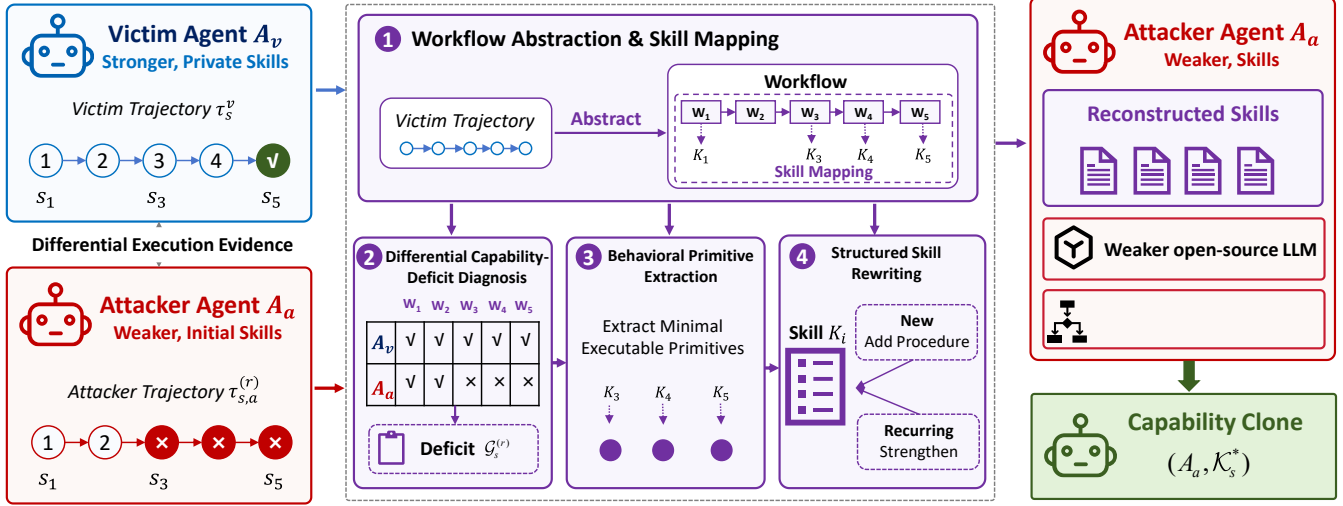


Figure 2: Overview of *AgentLeak*. Given an initial skill set and black-box execution evidence from the victim agent, *AgentLeak* progressively reconstructs an enhanced skill set $\mathcal{K}_s^*$ that enables capability cloning on the weaker attacker-controlled agent.

- **Analysis Assistant.** During reconstruction, the attacker can use an auxiliary LLM M_{aux} to analyze observable evidence and generate skill updates. The analysis assistant has no access to victim internals, performs no privileged execution, and is not used after reconstruction.

Adversary Knowledge. The adversary only accesses information exposed through normal interaction with the victim agent:

- **Initial Skills.** The attacker initializes an attacker-side skill set $\mathcal{K}_s^{(0)}$ relevant to scenario s , which can be constructed from public skill repositories, existing skill extraction techniques [27], or attacker-generated skills. This initial skill set serves only as a scaffold and may differ substantially from the victim’s private skill set $\mathcal{K}_s^v$.
- **Task-scenario Knowledge.** The attacker knows the target scenario and its associated task instances, including task instructions, expected objectives, and required execution contexts.
- **Observable Execution Evidence.** The attacker only observes execution information exposed through the victim agent’s normal interaction interface, including tool invocations, tool outputs, environment feedback, and so on.
- **No Privileged Victim Information.** The victim agent remains fully black-box: no access to its private skills, model parameters, system prompt, hidden states, private memory, unexposed chain-of-thought, or other internal information.

This setting captures a practical security threat in which a low-capability attacker-controlled agent can acquire the capability of a proprietary agent without improving its underlying

model. If limited black-box observations from a stronger victim agent can be converted into skills that substantially reduce $\Delta_{capa}(s)$, the attacker can obtain a cheap, independently deployable agent with comparable task-solving capability, directly eroding the commercial value of the proprietary service.

4 AgentLeak: Capability Cloning Attack

4.1 Attack Overview

AgentLeak is a black-box capability cloning attack that enables a weaker attacker-controlled agent A_a to approach the task-solving capability of a stronger proprietary agent A_v by reconstructing only its skill set. Starting from an initial skill set $\mathcal{K}_s^{(0)}$, *AgentLeak* leverages limited black-box execution evidence from A_v and iteratively refines attacker-side skills while keeping the underlying model, agent harness, and tools unchanged. The key insight is that victim–attacker execution differences expose capability-critical procedural behaviors that are implicitly realized by the stronger agent but missing from the weaker one. By identifying and incorporating such behaviors into attacker-side skills, *AgentLeak* progressively reduces the capability gap between the two agents.

As shown in Figure 2, *AgentLeak* reconstructs the victim’s task-solving capability through an iterative skill optimization process guided by execution differences. It first abstracts successful victim executions into a workflow skill that captures the reusable task-solving procedure and provides an explicit execution structure for the weaker agent. It then performs workflow-guided differential diagnosis to distinguish capability deficits from benign execution variations and identify the missing behaviors responsible for failures. Next, these deficits are converted into reusable behavioral primitives and incorpo-

rated into the relevant attacker-side skills through structured and localized rewriting. Repeating this process transforms implicit procedural behaviors realized by the stronger agent into explicit skill knowledge, yielding the reconstructed skill set $\mathcal{K}_s^*$ and the resulting capability clone $(A_a, \mathcal{K}_s^*)$.

4.2 Differential Skill Reconstruction

AgentLeak reconstructs the task-solving capability of a stronger victim agent by identifying and externalizing the procedural knowledge that enables the victim to succeed but is missing or insufficient in the weaker attacker-controlled agent. For scenario s , *AgentLeak* starts from the attacker's initial skill set $\mathcal{K}_s^{(0)}$ and iteratively refines it using a small number of queried task instances. At reconstruction round r , let $\mathcal{K}_s^{(r)}$ denote the current attacker-side skill set. For a queried instance t_s , the attacker executes the task with A_a under $\mathcal{K}_s^{(r)}$, producing an attacker trajectory $\tau_{s,a}^{(r)}$. If the attacker fails, *AgentLeak* queries the victim agent A_v to obtain a successful reference trajectory τ_s^v . The paired trajectories $(\tau_s^v, \tau_{s,a}^{(r)})$ provide differential execution evidence for identifying capability deficits, i.e., the procedural behaviors required by the victim but missing from the attacker. *AgentLeak* then converts these deficits into skill updates and iteratively refines $\mathcal{K}_s^{(r)}$ into $\mathcal{K}_s^{(r+1)}$.

The reconstruction process consists of four components: workflow abstraction and skill mapping, differential capability-gap diagnosis, behavioral primitive extraction, and structured skill rewriting. We next introduce each component in detail.

4.2.1 Workflow Abstraction and Skill Mapping

A successful victim trajectory τ_s^v demonstrates how the victim agent completes a task instance, but it also contains substantial instance-specific details and does not directly expose the reusable execution procedure underlying the victim's capability. As a result, a weaker agent may possess relevant skills but still fail to organize execution steps, coordinate available skills, or select appropriate actions. To bridge this gap, *AgentLeak* abstracts the victim execution into a structured *workflow skill*, which captures the reusable execution procedure, coordinates existing skills, and provides an explicit reference for subsequent capability-deficit diagnosis.

Given a queried instance t_s , its instruction x_s , the analysis assistant M_{aux} , the successful victim trajectory τ_s^v , and the initial attacker-side skill set $\mathcal{K}_s^{(0)}$, *AgentLeak* constructs the workflow skill through:

$$\begin{aligned} K_s^{wf} &= \text{WorkflowExtract}(M_{aux}, x_s, \tau_s^v, \mathcal{K}_s^{(0)}) \\ &\triangleq (W_s, V_s, \Phi_s, R_s^{(0)}). \end{aligned} \quad (5)$$

where W_s describes the ordered execution procedure of scenario s , including workflow steps and their dependencies; V_s defines the verification criteria for intermediate states and

completion conditions; Φ_s maps each workflow step to the corresponding existing skills; and $R_s^{(0)}$ records refinement guidance, which is initially empty and updated during subsequent reconstruction rounds. During workflow abstraction, *AgentLeak* preserves reusable procedures, tool-use patterns, validation requirements, and recovery strategies while removing instance-specific details.

Workflow Skill K_s^{wf}

Execution Structure W_s :

Ordered execution steps and dependencies.

Verification V_s :

Expected intermediate states and completion conditions.

Skill Mapping Φ_s :

Mapping from workflow steps to the existing skills.

Refinement Guidance R_s :

Adaptive guidance updated with identified capability deficits.

The generated workflow skill is incorporated into the initial attacker-side skill set:

$$\mathcal{K}_s^{(0)} \leftarrow \mathcal{K}_s^{(0)} \cup \{K_s^{wf}\}. \quad (6)$$

During subsequent reconstruction rounds, the execution structure W_s , verification criteria V_s , and skill mappings Φ_s remain unchanged, while the refinement guidance $R_s^{(r)}$ is progressively updated according to newly identified capability deficits. This design provides a stable execution scaffold while allowing *AgentLeak* to accumulate and incorporate missing procedural knowledge during iterative reconstruction.

4.2.2 Differential Capability-Deficit Diagnosis

Execution traces may differ due to variations in planning, tool selection, or intermediate decisions, but such differences do not necessarily indicate capability gaps. *AgentLeak* therefore performs workflow-guided differential diagnosis to identify the capability-critical behaviors required by the victim but missing or insufficient in the weaker attacker agent. Using the workflow skill as a procedural reference, the diagnosis is conducted at the workflow-step level and focuses on critical execution behaviors, including planning, tool usage, feedback recovery, and result verification.

At reconstruction round r , the attacker executes a queried instance t_s using the current skill set $\mathcal{K}_s^{(r)}$. To capture execution variability, the attacker performs multiple executions and obtains a set of trajectories $\tau_a^{(r)} = \{\tau_{s,a}^{(r,1)}, \dots, \tau_{s,a}^{(r,q)}\}$. The diagnosis module jointly analyzes the successful victim trajectory τ_s^v , attacker trajectories $\tau_a^{(r)}$, the current skill set $\mathcal{K}_s^{(r)}$ containing the workflow skill K_s^{wf} , and previous reconstruction history $\mathcal{H}^{(r)}$ to produce the set of capability deficits $\mathcal{G}_s^{(r)}$:

$$\mathcal{G}_s^{(r)} = \text{Diagnose}(x_s, M_{aux}, \tau_s^v, \tau_a^{(r)}, \mathcal{K}_s^{(r)}, \mathcal{H}^{(r)}). \quad (7)$$

Specifically, $\mathcal{G}_s^{(r)} = \{d_j^{(r)}\}_{j=1}^{|\mathcal{G}_s^{(r)}|}$, where each deficit $d_j^{(r)}$ represents a missing or insufficient behavior that prevents the attacker from achieving the victim capability. Since multiple attacker executions may expose similar failure patterns, *AgentLeak* groups semantically similar deficits and records their occurrence frequency. Deficits with higher frequency are prioritized for subsequent reconstruction. Each deficit records the affected skill, observable failure symptom, root cause, and reconstruction status. The reconstruction status distinguishes *new* and *recurring* deficits: a new deficit represents a previously uncovered missing behavior, whereas a recurring deficit indicates that previous updates have addressed the behavior but remain insufficient. A deficit is retained only when the identified behavior contributes to task failure; differences caused by task-specific inputs, intermediate values, or alternative successful execution paths are treated as benign variations.

Capability Deficits $d_j^{(r)}$

<Deficit_id>: Unique identifier of the deficit.
 <Count>: Occurrence frequency of the deficit.
 <Target_skill>: Skill responsible for the identified deficit.
 <Observation>: Observable failure symptom from execution trajectories.
 <Root_reason>: Underlying cause of the failure.
 <Reconstruction_status>: Whether the deficit is newly identified or recurring.

4.2.3 Behavioral Primitive Extraction

Capability-deficit diagnosis identifies the missing or insufficient behaviors that prevent the attacker agent from reproducing the victim capability, but these deficits only describe failure symptoms and root causes rather than the behaviors required for skill refinement. *AgentLeak* therefore converts each capability deficit into a *behavioral primitive*, which specifies the procedural behavior that should be introduced, strengthened, or corrected in the corresponding skill.

For skill K_s^j associated with a subset of diagnosed deficits $\mathcal{G}_s^{(r)}[K_s^j]$, *AgentLeak* extracts behavioral primitives:

$$\mathcal{P}_{s,j}^{(r)} = \text{ExtractPrimitive}(\mathcal{M}_{\text{aux}}, \mathcal{G}_s^{(r)}[K_s^j], K_s^j), \quad (8)$$

where $\mathcal{P}_{s,j}^{(r)}$ denotes the set of behavioral primitives extracted for refining skill K_s^j . Each primitive describes the desired behavioral improvement implied by a capability deficit, including the missing behavior and the procedural guidance required to realize it.

Behavioral primitives do not copy victim execution traces or task-specific actions. Instead, they abstract reusable procedural improvements from diagnosed deficits. For example, if a deficit indicates that the attacker generates an artifact

without validating its correctness, the corresponding primitive specifies the need for an artifact-validation and recovery procedure rather than reproducing the victim’s specific validation commands.

The extracted primitives are then passed to the structured skill rewriting component, which translates them into concrete skill patches and integrates them into existing skills while preserving effective skill components.

Behavioral Primitive $\mathcal{P}_{s,j}^{(r)}$

<Target_skill>: The Skill to be refined.
 <Missing_behavior>: The capability-critical behavior missing or insufficient in the attacker agent.
 <Required_procedure>: The procedural guidance describing how the skill should realize missing behavior.
 <Expected_effect>: The expected improvement after incorporating the primitive.

4.2.4 Structured Skill Rewriting

Behavioral primitives specify the procedural behaviors required for capability recovery, but they do not directly define how these behaviors should be integrated into existing skills. Directly regenerating an entire skill may overwrite effective procedural knowledge and introduce unnecessary modifications. *AgentLeak* therefore performs structured skill rewriting by generating localized patches that modify only the skill components associated with identified capability deficits.

Given a behavioral primitive set $\mathcal{P}_{s,j}^{(r)}$ for skill K_s^j , *AgentLeak* generates a set of skill patches:

$$\Pi_{s,j}^{(r)} = \text{GeneratePatch}(\mathcal{M}_{\text{aux}}, \mathcal{P}_{s,j}^{(r)}, K_s^j), \quad (9)$$

where each patch $\pi_i \in \Pi_{s,j}^{(r)}$ specifies a targeted modification to the existing skill. Specifically, a patch is represented as $\pi_i = (l_i, c_i, op_i)$, where l_i denotes the modification location, c_i denotes the procedural instruction to be inserted or used for replacement, and op_i denotes the modification operation. *AgentLeak* restricts patch operations to two types. INSERT adds procedural instructions when the required behavior is absent from the current skill. REPLACE substitutes incomplete or ineffective instructions when existing guidance fails to address the capability deficit. These localized updates preserve skill components while progressively incorporating missing procedural behaviors identified during diagnosis.

Skill Patch π_i

<Patch_id>: Unique identifier of the patch.
 <Target_skill>: Skill to be modified.
 <Location>: Target section or position for modification.
 <Operation>: Modification operation: INSERT or REPLACE.
 <Target_instruction>: The existing instruction to be

Algorithm 1 *AgentLeak*

Require: Victim agent A_v , attacker agent A_a , analysis assistant M_{aux} , initial skill set $\mathcal{K}_s^{(0)}$, query tasks $\mathcal{T}_s^q$, reconstruction budget B

Ensure: Reconstructed skill set $\mathcal{K}_s^*$

```

1:  $\mathcal{K} \leftarrow \mathcal{K}_s^{(0)}$ 
2: for each query task  $t_s \in \mathcal{T}_s^q$  do
3:    $(\tau_s^a, r_s^a) \leftarrow \text{Exec}(A_a, \mathcal{K}, t_s)$ 
4:   if  $r_s^a = 0$  then
5:      $\tau_s^v \leftarrow \text{Exec}(A_v, \mathcal{K}_s^v, t_s)$ 
6:      $\mathcal{K}_s^{\text{wf}} \leftarrow \text{WorkflowExtract}(M_{aux}, x_s, \tau_s^v, \mathcal{K})$ 
7:      $\mathcal{K} \leftarrow \mathcal{K} \cup \{\mathcal{K}_s^{\text{wf}}\}$ 
8:     for  $r = 1$  to  $B$  do
9:        $(\tau_s^a, r_s^a) \leftarrow (\mathcal{V}_s(\text{Exec}(A_a, \mathcal{K}, t_s), y_s))$ 
10:      if  $r_s^a = 1$  then
11:        break
12:      end if
13:       $\mathcal{G}_s^{(r)} \leftarrow \text{Diagnose}(M_{aux}, x_s, \tau_s^v, \tau_s^a, \mathcal{K})$ 
14:      for each skill  $K_s^j \in \mathcal{K}$  do
15:         $\mathcal{P}_{s,j}^{(r)} \leftarrow \text{ExtractPrimitive}(M_{aux}, \mathcal{G}_s^{(r)}[K_s^j], K_s^j)$ 
16:         $\Pi_{s,j}^{(r)} \leftarrow \text{GeneratePatch}(M_{aux}, \mathcal{P}_{s,j}^{(r)}, K_s^j)$ 
17:         $K_s^j \leftarrow \text{ApplyPatch}(K_s^j, \Pi_{s,j}^{(r)})$ 
18:      end for
19:    end for
20:  end if
21: end for
22: Perform regression checking on previously solved task instances
23:  $\mathcal{K}_s^* \leftarrow \mathcal{K}$ 
24: return  $\mathcal{K}_s^*$ 

```

replaced when the operation is REPLACE.
 <Content>: The new procedural instruction to be inserted or used for replacement.

After generating patches, *AgentLeak* applies them through deterministic skill update operations:

$$\mathcal{K}_s^{(r+1)} = \text{ApplyPatch}(\mathcal{K}_s^{(r)}, \Pi_{s,j}^{(r)}). \quad (10)$$

The updated skills are then evaluated through subsequent attacker executions. If failures remain, the new execution evidence triggers another round of capability-deficit diagnosis, behavioral primitive extraction, and patch generation. Repeating this process progressively refines the attacker-side skills and produces the reconstructed skill set $\mathcal{K}_s^*$. The detailed examples of reconstructed skills after applying the generated patches are provided in Appendix B.

4.3 End-to-End Attack Procedure

Algorithm 1 summarizes the end-to-end *AgentLeak* procedure. For each query instance, A_a executes the task instance with the current skill set. Upon failure, *AgentLeak* obtains a victim trajectory, abstracts a reusable workflow, diagnoses capability deficits from victim–attacker execution differences, extracts

Table 2: Task scenarios used in our evaluation.

Domain	Task Scenario	Difficulty	#Skills
Office & White-Collar	Citation Check	Medium	1
	Court Form Filling	Easy	1
	Excel Table in PPT	Medium	2
	PDF–Excel Diff	Medium	2
	Powerlifting Coefficient Calculation	Easy	3
Software Engineering	JAX Computing Basics	Medium	1
	LLM Prefix Cache Replay	Medium	2
	TicToc Unnecessary Abort Detection	Hard	3
Finance & Economics	Economic Detrending Correlation	Medium	1
	SEC Financial Report	Hard	2
Natural Science	Flood Risk Analysis	Medium	3
	Gravitational Wave Detection	Medium	2
	Lab Unit Harmonization	Medium	1
	Lake Warming Attribution	Medium	4
	Mars Clouds Clustering	Hard	3
	Protein Expression Analysis	Medium	1
Industrial & Physical Systems	Grid Dispatch Operator	Medium	3
	HVAC Control	Medium	5
	Manufacturing-Equipment-Maintenance	Medium	2
Mathematics & Formal Reasoning	PDDL Airport Planning	Medium	1

behavioral primitives, and applies structured skill rewriting. The updated skills are iteratively refined until the task instance succeeds or the reconstruction budget is exhausted. After processing all query instances and eliminating regressions, the reconstructed skill set $\mathcal{K}_s^*$ is frozen and combined with A_a to form the capability clone $(A_a, \mathcal{K}_s^*)$.

5 Evaluation

5.1 Experimental Setup

Datasets and Task Instances. We evaluate *AgentLeak* on SKILLSBENCH [12], which contains realistic agent tasks requiring multi-step execution, specialized skills, tool interactions, and deterministic verification. We select 20 task scenarios spanning 6 application domains and covering diverse difficulty levels and skill compositions, as summarized in Table 2.

Because each original scenario contains only one task instance, evaluating reconstruction on the original instance alone cannot distinguish transferable capability recovery from instance-specific memorization. We therefore construct 30 instances for each scenario, yielding 600 instances in total. For each task scenario, we first identify its core task requirements, which define the objective and execution requirements that must remain invariant across instances. We then use an LLM to propose variation slots over task content, parameters, targets, and environmental context, while preserving the underlying workflow, required skills, tool interactions, and verification criteria. All variation slots are manually reviewed to ensure that they are valid, independent, and non-redundant. Based on these slots, we generate 20 instances with single-slot

Table 3: Victim and attacker agent systems used in our work.

Role	Agent Harness	Backbone Model
Victim	Claude Code	Claude Opus-4.8
Victim	Codex	GPT 5.5
Victim	OpenHands	Deepseek V4-flash-0731
Victim	OpenHands	MiniMax M3
Attacker	OpenHands	Qwen 3.6-35B-A3B
Attacker	OpenHands	Gemma 4-31B

variations and 10 with two-slot variations, with each variation type represented by at least three instances.

Every generated instance is subsequently subjected to both LLM-based validation and manual verification. The validation examines consistency with the original scenario, validity of the intended variation, executability of the task and environment, and correctness of the verification criteria. Instances that fail either validation are revised or regenerated before inclusion. Detailed construction and validation procedures, together with representative instance examples, are provided in Appendix C.

For each scenario, we use one instance as the *query instance* for reconstruction and reserve the remaining 29 as *held-out instances* for evaluation. During reconstruction, *AgentLeak* accesses only the query instance and its victim execution evidence. The held-out instances remain unseen until the skill set is fixed. Thus, performance on these instances evaluates whether the recovered capability generalizes beyond the execution observed during reconstruction.

Victim and Attacker Systems: We evaluate *AgentLeak* under a strong-to-weak setting with heterogeneous victim agents and attacker-controlled agents. Victim agents are instantiated using different agent harnesses and backbone models, including Claude Code powered by Claude Opus 4.8 [3], Codex powered by GPT-5.5 [20], and OpenHands [26] powered by DeepSeek-V4-Flash-0731 [6] and MiniMax-M3 [17]. These configurations cover both proprietary agent products and general-purpose agent frameworks equipped with strong models. For attacker-controlled agents, we use OpenHands with weaker open-weight models, including Qwen 3.6-35B-A3B [22] and Gemma 4-31B [1]. During the entire reconstruction process, the attacker’s model, agent harness, and tools remain unchanged; only attacker-side skills are modified. We further use GPT-5.6 [19] as the analysis assistant M_{aux} for workflow abstraction, capability-deficit diagnosis, behavioral primitive extraction, and skill rewriting. The analysis assistant is used only during offline reconstruction and is not required after deployment. The complete system configurations are summarized in Table 3.

Baselines. We compare *AgentLeak* with three attacker-side baselines under the attack configuration. The victim agent equipped with its native skills is reported as an upper bound. *No Skill*. The attacker-controlled agent performs tasks with-

out external skills. This baseline measures the intrinsic capability of the weaker agent without procedural knowledge.

Raw Skill. The attacker directly uses the initial skill set $\mathcal{K}_s^{(0)}$ obtained from existing skill stealing mechanism [27], without further refinement.

Trace2Skill. Trace2Skill [18] generates reusable skills by summarizing execution trajectories. We select it as a representative self-evolving skill baseline because its trajectory-to-skill formulation closely matches our setting. We apply it to attacker-side execution trajectories to evaluate whether self-generated skill evolution can improve the weaker agent.

Victim Agent with Raw Skill. The original victim agent equipped with its native skills represents the target capability that *AgentLeak* aims to reproduce on the weaker attacker-controlled agent.

Evaluation Metrics: We evaluate whether *AgentLeak* enables a weaker attacker-controlled agent to recover the task-solving capability of the victim agent. We use two complementary metrics: *Instance Pass Rate* (IPR) and *Capability Recovery Rate* (CRR). To account for execution stochasticity, each held-out task instance is executed independently $R = 3$ times. Each execution is evaluated by the task-specific verifier, which returns a binary outcome indicating whether the generated artifact or final environment state satisfies all predefined requirements.

Instance Pass Rate (IPR). IPR measures the absolute task-solving capability of an agent on unseen instances. For scenario s , it is defined as:

$$\text{IPR}_s = \frac{1}{R|\mathcal{D}_s^{\text{test}}|} \sum_{t \in \mathcal{D}_s^{\text{test}}} \sum_{j=1}^R z_{t,j}, \quad (11)$$

where R denotes the number of repeated executions per instance and $z_{t,j} \in \{0, 1\}$ indicates whether the j -th execution of test instance t succeeds according to the task verifier. We report the macro-averaged pass rate:

$$\overline{\text{IPR}} = \frac{1}{|\mathcal{C}|} \sum_{s \in \mathcal{C}} \text{IPR}_s, \quad (12)$$

where $\mathcal{C}$ denotes the set of evaluated task scenarios.

Capability Recovery Rate (CRR). While IPR measures the absolute capability achieved after reconstruction, it does not quantify how much of the victim–attacker capability gap has been recovered. We therefore define CRR as:

$$\text{CRR}_s = \frac{\text{IPR}_s^{\text{AgentLeak}} - \text{IPR}_s^{\text{NoSkill}}}{\text{IPR}_s^{\text{Victim}} - \text{IPR}_s^{\text{NoSkill}}}. \quad (13)$$

Here, $\text{IPR}_s^{\text{AgentLeak}}$, $\text{IPR}_s^{\text{NoSkill}}$, and $\text{IPR}_s^{\text{Victim}}$ denote the pass rates of *AgentLeak*, the attacker agent without skills, and the original victim agent, respectively. The overall recovery rate is computed by macro-averaging:

$$\overline{\text{CRR}} = \frac{1}{|\mathcal{C}|} \sum_{s \in \mathcal{C}} \text{CRR}_s. \quad (14)$$

Table 4: Capability cloning performance. The top *Victim w/ Skill* row reports each victim agent’s own $\overline{\text{IPR}}$. No Skill, Raw Skill, and Trace2Skill use victim-independent skills across victim settings, so their $\overline{\text{IPR}}$ remains identical; their $\overline{\text{CRR}}$ varies only with the corresponding victim capability ceiling. *AgentLeak*’s $\overline{\text{IPR}}$ / $\overline{\text{CRR}}$ under each victim are obtained by reconstructing skills from that victim’s executions. **Bold** indicates the best attacker-side values under each victim setting.

Attacker Agent	Method	Claude Code Opus 4.8		Codex GPT-5.5		OpenHands DeepSeek-V4		OpenHands MiniMax-M3	
		IPR ↑	CRR↑	IPR↑	CRR↑	IPR ↑	CRR↑	IPR ↑	CRR↑
Victim w/ Skill		75.82	–	80.44	–	78.51	–	69.61	–
OpenHands Qwen3.6-35B-A3B	No Skill	21.13	–	21.13	–	21.13	–	21.13	–
	Raw Skill	30.64	17.38	30.64	16.03	30.64	16.57	30.64	19.61
	Trace2Skill	34.96	25.29	34.96	23.32	34.96	24.10	34.96	28.53
	AgentLeak	75.16	98.79	72.80	87.12	80.16	102.88	71.94	104.81
OpenHands Gemma 4-31B	No Skill	14.94	–	14.94	–	14.94	–	14.94	–
	Raw Skill	35.59	33.92	35.59	31.52	35.59	32.48	35.59	37.77
	Trace2Skill	28.81	22.78	28.81	21.17	28.81	21.82	28.81	25.37
	AgentLeak	78.29	104.9	70.77	85.23	83.72	108.20	73.91	107.87

A value of $\overline{\text{CRR}}_s = 0$ indicates that reconstruction provides no improvement over the attacker agent without skills, while $\overline{\text{CRR}}_s = 1$ indicates complete recovery of the observed capability gap. Values larger than 1 indicate that the reconstructed skills achieve higher performance than the victim agent on the evaluated scenario.

Together, $\overline{\text{IPR}}$ and $\overline{\text{CRR}}$ characterize capability cloning from two perspectives: higher $\overline{\text{IPR}}$ reflects better task-solving performance on unseen tasks, while higher $\overline{\text{CRR}}$ reflects more complete recovery of the victim’s capability advantage.

5.2 Attack Effectiveness

5.2.1 Overall Attack Performance

Table 4 reports the overall $\overline{\text{IPR}}$ and $\overline{\text{CRR}}$ across all task scenarios and victim–attacker configurations. Figure 3 presents detailed comparisons on five representative scenarios, while the complete results for the remaining scenarios are provided in Appendix D.

Finding 1: Direct skill reuse cannot transfer the capability of stronger agents. Directly deploying acquired skills provides only limited capability recovery on weaker agents. For the Qwen 3.6-35B-A3B attacker, Raw Skill improves $\overline{\text{IPR}}$ from 21.13% (No Skill) to 30.64%, recovering only 17.38%–19.61% of the victim–attacker capability gap. Similarly, Raw Skill achieves 35.59% $\overline{\text{IPR}}$ and 32.48%–37.77% $\overline{\text{CRR}}$ for the Gemma 4-31B attacker, remaining substantially below the victim agents. These results demonstrate that explicit skill artifacts alone are insufficient to reproduce the capability realized by stronger agents.

Finding 2: Existing skill evolution methods fail to recover implicit capability-critical behaviors. Trace2Skill improves

or reorganizes skills from execution trajectories, but it cannot effectively recover the hidden procedural behaviors underlying victim capabilities. It achieves only 34.96% $\overline{\text{IPR}}$ for Qwen 3.6-35B-A3B and 28.81% for Gemma 4-31B, with $\overline{\text{CRR}}$ values of 23.32%–28.53% and 21.17%–25.37%, respectively. These results indicate that trajectory summarization or self-evolution alone is insufficient, as capability recovery requires identifying the behaviors that distinguish successful victim executions from attacker failures.

Finding 3: AgentLeak effectively clones stronger agent capabilities onto weaker agents. By leveraging victim–attacker execution differences, *AgentLeak* achieves over 70% $\overline{\text{IPR}}$ across all victim–attacker configurations, outperforming Raw Skill and Trace2Skill by more than 35 percentage points. These results demonstrate that *AgentLeak* can effectively transfer the task-solving capabilities of stronger victim agents onto substantially weaker attackers.

Figure 3 in this section and Figure 6 in Appendix D provide task-level comparisons across diverse scenarios and application domains. Consistent with the overall results, *AgentLeak* achieves the highest attacker-side performance across most tasks, substantially outperforming Raw Skill and Trace2Skill. Unlike Raw Skill, which only transfers explicit procedural descriptions, and Trace2Skill, which evolves skills from the attacker’s own executions, *AgentLeak* identifies the capability-critical behaviors missing from weaker-agent executions and incorporates them into the reconstructed skills. The resulting skills therefore provide more complete execution guidance, helping the weaker agent make more appropriate action choices, verify intermediate results, and recover from execution failures. This enables the weaker agent to execute tasks more reliably and better reproduce the task-solving capability of the stronger victim agent.

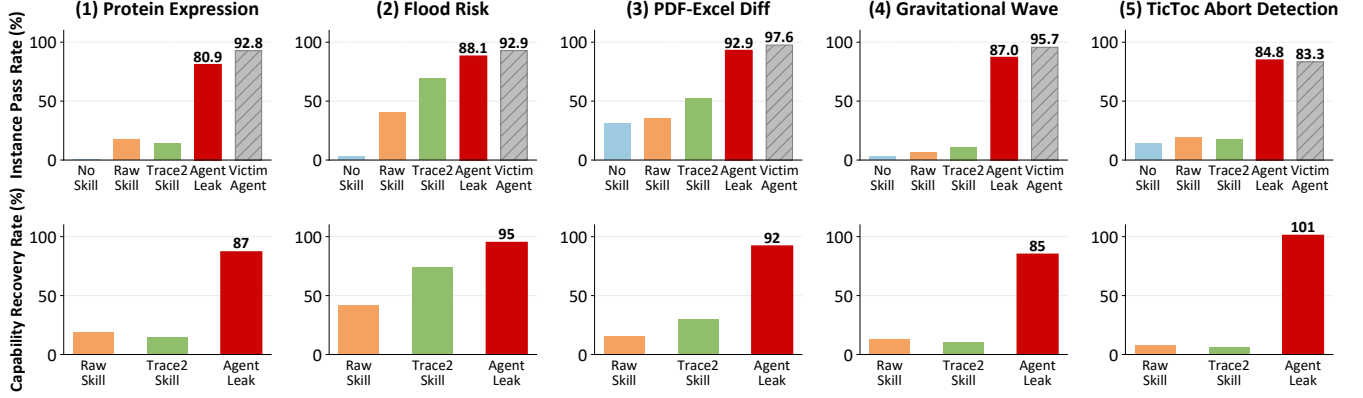


Figure 3: Capability cloning performance on representative task scenarios from 5 different application domains.

Table 5: Transferability of reconstructed skills across attacker systems. Δ is the change after transfer.

Transfer Setting	Source Agent		Target Agent (Δ)	
	IPR	CRR	IPR	CRR
Cross-Model			72.23 ($\uparrow 2.17$)	86.16 ($\uparrow 3.66$)
Cross-Harness	70.06	82.50	84.68 ($\uparrow 14.62$)	107.15 ($\uparrow 24.65$)
Cross-Model & Harness			83.33 ($\uparrow 13.27$)	104.87 ($\uparrow 22.37$)

5.2.2 Transferability of Reconstructed Skills

We further evaluate whether the capability reconstructed by *AgentLeak* is specific to the attacker system used during reconstruction or captures transferable procedural behaviors. For each scenario s , we first reconstruct the skill set $\mathcal{K}_s^*$ on a source attacker system. After reconstruction, $\mathcal{K}_s^*$ is frozen and directly deployed on different target attacker systems without additional skill updates, victim access, or reconstruction. We consider three transfer settings using OpenHands with Qwen 3.6-35B-A3B as the source attacker. In *cross-model* transfer, only the underlying model is replaced with Gemma 4-31B while retaining the same OpenHands harness. In *cross-harness* transfer, the agent harness is replaced with Claude Code while keeping the Qwen 3.6-35B-A3B model unchanged. In *cross-model+cross-harness* transfer, both components are changed by deploying the reconstructed skills on Claude Code with Gemma 4-31B. Table 5 reports the performance before and after transfer, where Δ denotes the change after deployment.

Reconstructed skills transfer effectively across heterogeneous attacker systems. As shown in Table 5, the reconstructed skills remain effective after transfer to different attacker configurations without additional reconstruction or victim interaction. Under the *cross-model* setting, the transferred skills achieve 72.23% $\overline{\text{IPR}}$ and 86.16% $\overline{\text{CRR}}$, maintaining comparable performance to the source attacker. Under the *cross-harness* setting, the transferred skills further improve performance, increasing $\overline{\text{IPR}}$ from 70.06% to 84.68% and $\overline{\text{CRR}}$

Table 6: Ablation of *AgentLeak* components.

Variant	$\overline{\text{IPR}}(\%)$	$\overline{\text{CRR}}(\%)$
Full <i>AgentLeak</i>	77.80	95.50
w/o Workflow Abstract	22.33 (-55.50)	2.00 (-93.52)
w/o Differential Diagnosis	55.28 (-21.98)	61.77 (-33.73)
w/o Behavioral Primitives	72.50 (-5.30)	86.60 (-8.90)
w/o Structured Skill Rewriting	39.20 (-38.60)	30.50 (-65.00)

from 82.50% to 107.15%. When both the model and harness are changed, the transferred skills still achieve 83.33% $\overline{\text{IPR}}$ and 104.87% $\overline{\text{CRR}}$. These results demonstrate that *AgentLeak* reconstructs reusable procedural behaviors rather than attacker-specific execution patterns. Once recovered, the reconstructed skills can be deployed across heterogeneous agent systems, revealing that capability leakage extends beyond a single attacker configuration and can enable capability replication.

5.2.3 Ablation Study

We conduct ablation experiments to quantify the contribution of four components in *AgentLeak*: workflow abstraction, differential diagnosis, behavioral primitive extraction, and structured Skill rewriting. Specifically, **w/o Workflow Abstract** removes workflow abstraction and directly performs reconstruction without an explicit execution structure; **w/o Differential Diagnosis** removes victim-attacker comparison and identifies deficits only from attacker failures; **w/o Behavioral Primitives** directly rewrites skills without converting deficits into behavioral primitives; and **w/o Structured skill Rewriting** replaces localized patch-based updates with direct skill modification. All variants use the same victim/attacker systems, query instances, and reconstruction budget as the full attack. Table 6 reports the resulting $\overline{\text{IPR}}$ and $\overline{\text{CRR}}$.

All components are critical for capability reconstruction. Removing workflow abstraction causes the largest degrada-

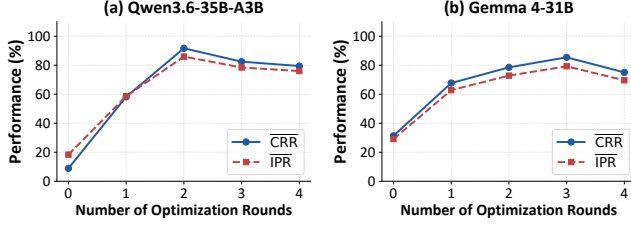


Figure 4: Impact of reconstruction rounds on *AgentLeak*. The two panels show $\overline{IPR}$ and $\overline{CRR}$ under different numbers of diagnosis–rewrite iterations for the Qwen 3.6-35B-A3B and Gemma 4-31B attackers, respectively.

tion, reducing $\overline{IPR}$ from 77.80% to 22.33% and $\overline{CRR}$ from 95.50% to 2.00%, demonstrating that an execution structure is essential for guiding capability recovery. Removing differential diagnosis decreases $\overline{IPR}$ and $\overline{CRR}$ to 55.28% and 61.77%, showing that attacker failures alone cannot reveal the behaviors missing from the stronger agent. Removing behavioral primitives reduces $\overline{IPR}$ and $\overline{CRR}$ to 72.50% and 86.60%, confirming the importance of converting deficits into actionable refinement guidance. Finally, removing structured skill rewriting leads to a substantial drop to 39.20% $\overline{IPR}$ and 30.50% $\overline{CRR}$, highlighting the necessity of precise and localized skill updates. These results demonstrate that *AgentLeak* requires both accurate capability-gap identification and effective skill refinement to achieve capability cloning.

5.2.4 Attack Feasibility Analysis

We evaluate the practical feasibility of *AgentLeak* from three aspects: reconstruction rounds, scenario complexity, and attack cost. Reconstruction rounds and scenario complexity measure the effectiveness and robustness of capability recovery under different reconstruction conditions, while attack cost evaluates the practical resources required to perform the attack.

Impact of Reconstruction Rounds. We study how the number of diagnosis–rewrite iterations affects capability recovery by varying the reconstruction rounds from 1 to 4. Figure 4 reports the corresponding $\overline{IPR}$ and $\overline{CRR}$. *AgentLeak* can achieve substantial capability recovery with only a few reconstruction rounds. As shown in Figure 4, both attackers obtain most performance gains within the first two to three rounds. For the Qwen 3.6-35B-A3B attacker, $\overline{CRR}$ reaches around 90% after two rounds and then saturates. The Gemma 4-31B attacker shows a similar trend, with limited improvement after the third round. These results indicate that the major capability deficits can be identified and incorporated into skills within a small reconstruction budget.

Impact of Scenario Complexity. We investigate whether *AgentLeak* remains effective as task scenarios involve more skills. We use the number of involved skills as a proxy for scenario complexity and group scenarios according to $N_s =$

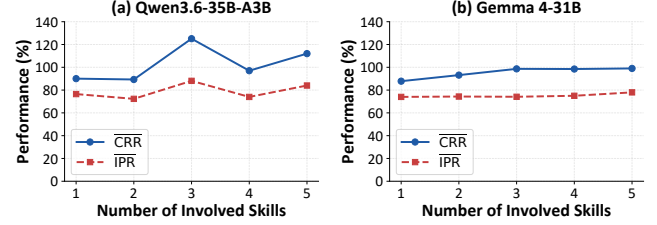


Figure 5: Impact of scenario complexity on *AgentLeak*. Each panel shows the $\overline{IPR}$ and $\overline{CRR}$ under different numbers of involved skills.

Table 7: Effectiveness of defenses against capability reconstruction by *AgentLeak*, all applied on the victim side.

Defense	$\overline{IPR}$ (%)	$\overline{CRR}$ (%)
None (undefended)	85.82	95.44
Input-Phase LLM Detector	81.35	86.07
SkillGuard-5	77.12	77.22
Semantic Output Filtering	82.43	88.34
Trajectory Redaction	75.21	73.23

1, 2, 3, 4, and 5. Figure 5 reports the corresponding $\overline{IPR}$ and $\overline{CRR}$ under different complexity levels. *AgentLeak* remains effective across scenarios with increasing skill complexity. As shown in Figure 5, the performance degradation is limited as the number of involved skills increases. For the Qwen 3.6-35B-A3B attacker, $\overline{CRR}$ remains above 84% across all complexity levels, with $\overline{IPR}$ above 70%. For the Gemma 4-31B attacker, $\overline{CRR}$ remains around 90%. These results demonstrate that *AgentLeak* can recover capability-critical behaviors across multi-skill scenarios and maintain effective capability cloning as task complexity increases.

Attack Cost Analysis. *AgentLeak* requires only limited black-box access to the victim agent: for each scenario, the attacker uses only a single query instance for capability reconstruction, without model training or backbone replacement. Under the default setting, reconstruction requires an average of 6.34 attacker executions, 7.34 auxiliary-model calls, and 0.32M auxiliary-model tokens, with an average auxiliary-model cost of approximately \$6.32 and a reconstruction time of about 60 minutes per scenario. These results show that *AgentLeak* can achieve substantial capability recovery with minimal victim-side exposure and modest reconstruction overhead, making strong-to-weak capability cloning practical at low cost.

5.3 Defense Analysis

We investigate whether existing defenses can mitigate the capability leakage exposed by *AgentLeak*. We evaluate four defenses covering two protection paradigms. The first three

are artifact-level defenses that aim to prevent skill extraction or disclosure at different stages of the interaction pipeline. **Input-Phase LLM Detector** [27] detects and blocks requests with potential skill extraction intent before execution. **SkillGuard-5** [27] enforces confidentiality constraints during inference to prevent protected skill disclosure. **Semantic Output Filtering** [27] filters outputs whose semantic similarity to protected skills exceeds a predefined threshold. In contrast, **Trajectory Redaction** represents a behavior-level defense that restricts the execution information exposed to attackers by removing intermediate tool-call details from observable trajectories. Unless otherwise stated, we use Codex with GPT-5.5 as the victim agent and OpenHands with Qwen 3.6-35B-A3B as the attacker, following the main experimental setting.

Artifact-level defenses for skills cannot prevent capability reconstruction. As shown in Table 7, *AgentLeak* remains effective under all three artifact-level defenses, achieving $\overline{\text{IPR}}$ values of 81.35%, 77.12%, and 82.43% under Input-Phase LLM Detector, SkillGuard-5, and Semantic Output Filtering, respectively. Compared with the undefended setting (85.82% $\overline{\text{IPR}}$, 95.44% $\overline{\text{CRR}}$), these defenses reduce $\overline{\text{CRR}}$ by only 7.1–18.2 percentage points and fail to prevent substantial capability recovery. The reason is that these defenses target skill artifacts, whereas *AgentLeak* reconstructs capability from victim–attacker execution differences rather than extracting the protected skill. Even when the original skill is unavailable, the attacker can initialize a scenario-relevant skill scaffold and progressively recover missing procedural behaviors from observable victim executions. Therefore, protecting skill confidentiality alone is insufficient to prevent capability leakage.

Behavior-level protection provides stronger mitigation but remains insufficient. Trajectory Redaction directly targets the behavioral leakage surface exploited by *AgentLeak* by removing intermediate tool-call information from the victim’s observable execution trace. It reduces $\overline{\text{CRR}}$ from 95.44% to 73.23% and lowers $\overline{\text{IPR}}$ to 75.21%, producing a substantially larger degradation than artifact-level defenses. This result further confirms that *AgentLeak* relies on capability-critical information exposed through victim executions. Removing tool invocation details limits the attacker’s ability to infer how the stronger agent selects actions, verifies intermediate states, and recovers from failures. Nevertheless, *AgentLeak* still recovers a substantial portion of the capability gap, indicating that redacting tool-call information alone cannot eliminate behavioral capability leakage. More targeted defenses are therefore needed to suppress capability-critical execution information while preserving normal agent functionality.

6 Discussion

6.1 Capability Leakage Risks

AgentLeak reveals that the security boundary of proprietary LLM agents extends beyond static artifacts to the observ-

able behaviors through which capabilities are realized. Even when internal components remain protected, the task-solving capability enabled by these components can leak through interaction. We highlight three implications.

Capability confidentiality requires more than artifact protection. Existing protections primarily focus on explicit assets, including model parameters, system prompts, and skills, implicitly assuming that protecting these artifacts preserves the associated capability. *AgentLeak* challenges this assumption by showing that capability-critical procedural behaviors can be reconstructed without recovering the original artifacts. Therefore, safeguarding individual components alone is insufficient when the resulting behaviors remain observable.

Observable execution behavior forms a capability leakage surface. LLM agents naturally expose interaction behaviors during task execution, including action selection, tool invocation, verification, and recovery strategies. These behaviors encode procedural expertise that may not be explicitly represented in skills but is essential for successful task completion. Our findings suggest that future agent security mechanisms should consider behavioral disclosure control while maintaining transparency for debugging and auditing.

Proprietary capabilities can be replicated without replicating the underlying agent system. Traditional extraction attacks typically aim to recover protected artifacts or approximate the victim model itself. *AgentLeak* shows that neither is necessary for capability replication: capability-critical behaviors inferred from black-box executions can be incorporated into the skills of a substantially weaker agent while leaving its model, harness, and tools unchanged. Consequently, an attacker may reproduce much of a proprietary agent’s task-solving capability without reconstructing the proprietary system that originally realizes it, substantially lowering the barrier to cloning agent services.

6.2 Limitations

AgentLeak targets procedural capabilities that are expressed through skills and observable execution behaviors. The attack relies on sufficient interaction feedback from victim agents and becomes less applicable when agents expose only final outputs or highly restricted interfaces. Moreover, *AgentLeak* cannot reconstruct capabilities that depend on inaccessible resources, including private data, victim-exclusive tools, proprietary services, or model-internal knowledge. Extending capability leakage analysis to more restrictive interaction settings and developing defenses against behavioral reconstruction remain important future directions.

7 Conclusion

This paper reveals a capability leakage risk in LLM agents: a weaker attacker-controlled agent can reconstruct a stronger proprietary agent’s task-solving capability from limited observable executions. We present *AgentLeak*, a black-box capability-

cloning attack that identifies capability-critical behaviors from victim–attacker execution differences and encodes them into attacker-side skills, while keeping the attacker’s underlying model, harness, and tools unchanged. Extensive evaluations across diverse task scenarios, agent frameworks, and backbone models demonstrate that *AgentLeak* substantially improves weaker agents and recovers a large fraction of the victim–attacker capability gap. These findings show that securing proprietary agents requires protecting not only explicit artifacts but also the behavioral information exposed during execution, as such information can enable capability reconstruction.

Ethical Considerations

AgentLeak investigates capability leakage risks in LLM agents with the goal of improving the security and robustness of agent systems. Although our method demonstrates that a weaker attacker-controlled agent can reconstruct capabilities from observable execution behaviors, all experiments are conducted in controlled research environments using publicly available benchmarks and authorized agent configurations. We do not access private user data, confidential information, or unauthorized agent deployments beyond the assumptions specified in our threat model.

The goal of this work is to characterize capability cloning risks and motivate effective defenses, rather than to facilitate unauthorized replication of deployed agent services. All victim and attacker agents are instantiated for research evaluation, and collected execution traces are used solely for experimental analysis. We believe that understanding these risks is essential for developing mechanisms to reduce unintended capability exposure and protect agent systems.

References

- [1] Sherif El Abd, Vaibhav Aggarwal, Robin Algayres, Alek Andreev, Olivier Bachem, Ian Ballantyne, Cormac Brick, Victor Carbune, Michelle Casbon, Mayank Chaturvedi, Aditya Chawla, Victor Cotruta, Alice Coucke, Phil Culliton, Robert Dadashi, Lucas Dixon, Mohamed Elhawaty, Utku Evci, Clément Farabet, Johan Ferret, Filippo Galgani, Sertan Girgin, Jean-Bastien Grill, Maarten Grootendorst, Jiaxian Guo, Cassidy Hardin, Yanzhang He, Steven M. Hernandez, Omri Homburger, Léonard Hussenot, Ju-yeong Ji, Armand Joulin, Aishwarya Kamath, Parnian Kassraie, Olivier Lacombe, Preethi Lahoti, Gaël Liu, Gus Martins, Luciano Martins, Tatiana Matejovicova, Ramona Merhej, Nikola Momchev, Sneha Mondal, Ryan Mullins, Sindhu Raghuram Panyam, Shreya Pathak, Sarah Perrin, André Susano Pinto, Etienne Pot, Angéline Pouget, Alexandre Ramé, Sabela Ramos, Douglas Reid, David Rim, Morgane Rivière, Karsten Roth, Louis Rouillard, Omar Sanseviero, Pier Giuseppe Sessa, Shane Settle, Danila Sinopalnikov, Sara Smoot, Piotr Stanczyk, Andreas Steiner, Lawrence Stewart, Ilya O. Tolstikhin, Michael Tschannen, Anton Tsitsulin, Nino Vieillard, Renjie Wu, Pingmei Xu, Haichuan Yang, Edouard Yvinec, Biao Zhang, Li Zhang, Joe Zou, Nicolas Aagnes, Abdelrahman Abdelhamed, Jakub Adámek, Shivani Agrawal, Shubham Agrawal, Ibrahim Alabdulmohsin, Jean-Baptiste Alayrac, Uri Alon, Chandramouli Amarnath, Ankesh Anand, Chrysovalantis Anastasiou, Setareh Ariaifar, François-Xavier Aubet, Kyriakos Axiotis, Federico Barbero, Joelle K. Barral, Alexei Bendebury, Urs Bergmann, Stanley Bileschi, Kat Black, Mathieu Blondel, Sebastian Borgeaud, and Arthur Brazinskas. Gemma 4 technical report. *CoRR*, abs/2607.02770, 2026. [arXiv:2607.02770](https://arxiv.org/abs/2607.02770).
- [2] Anthropic. Equipping agents for the real world with agent skills. <https://www.anthropic.com/engineering/equipping-agents-for-the-real-world-with-agent-skills>, October 2025.
- [3] Anthropic. Introducing claude opus 4.8. Anthropic, 2026. Accessed August 2026.
- [4] Nicholas Carlini, Daniel Paleka, Krishnamurthy Dj Dvijotham, Thomas Steinke, Jonathan Hayase, A. Feder Cooper, Katherine Lee, Matthew Jagielski, Milad Nasr, Arthur Conmy, Eric Wallace, David Rolnick, and Florian Tramèr. Stealing part of a production language model. In *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*, volume 235 of *Proceedings of Machine Learning Research*, pages 5680–5705. PMLR / OpenReview.net, 2024.
- [5] Pinji Chen, Jinlong Jiang, Jianjun Chen, Feiran Qin, Minghao Zhang, Jiahe Zhang, Haixin Duan, Kaiwen Shen, and Hui Jiang. LLMThief: Evaluating configuration leaking risks in commercial llm app stores. In *2026 IEEE Symposium on Security and Privacy (SP)*, pages 1019–1037, 2026.
- [6] DeepSeek-AI. Deepseek-v4: Towards highly efficient million-token context intelligence, 2026. [arXiv:2606.19348](https://arxiv.org/abs/2606.19348).
- [7] Jianing Geng, Ruiqi He, Zekun Fei, Biao Yi, Ruijie Wang, Zheli Liu, Xia Hu, Xuansheng Wu, and Qingkai Zeng. Agent skills matter: Inferring proprietary skills from execution trajectories, 2026. [arXiv:2607.25560](https://arxiv.org/abs/2607.25560).
- [8] Peichun Hua, Haoxuan Xu, and Mengyuan Li. Behavioral skill reconstruction: Reconstructing hidden functionality from llm agent skills, 2026. [arXiv:2608.04192](https://arxiv.org/abs/2608.04192).
- [9] Zisu Huang, Jingwen Xu, Yifan Yang, Ziyang Gong, Qihao Yang, Muzhao Tian, Xiaohua Wang, Changze Lv, Xuemei Gao, Qi Dai, Bei Liu, Kai Qiu, Xue Yang, Dongdong Chen, Xiaoqing Zheng, and Chong Luo. From raw experience to skill consumption: A systematic study of model-generated agent skills. *CoRR*, abs/2605.23899, 2026. URL: <https://doi.org/10.48550/arXiv.2605.23899>, [arXiv:2605.23899](https://arxiv.org/abs/2605.23899), doi:10.48550/ARXIV.2605.23899.
- [10] Matthew Jagielski, Nicholas Carlini, David Berthelot, Alex Kurakin, and Nicolas Papernot. High accuracy and high fidelity extraction of neural networks. In Srdjan Capkun and Franziska Roesner, editors, *29th USENIX Security Symposium, USENIX Security 2020, August*

- 12-14, 2020, pages 1345–1362. USENIX Association, 2020.
- [11] Yanna Jiang, Delong Li, Haiyu Deng, Baihe Ma, Xu Wang, Qin Wang, and Guangsheng Yu. Sok: Agent skills - beyond tool use in LLM agents. *CoRR*, abs/2602.20867, 2026. [arXiv:2602.20867](https://arxiv.org/abs/2602.20867).
 - [12] Xiangyi Li, Wenbo Chen, Yimin Liu, Shenghan Zheng, Xiaokun Chen, Yifeng He, Yubo Li, Bingran You, Hao-tian Shen, Jiankai Sun, Shuyi Wang, Qunhong Zeng, Di Wang, Xuandong Zhao, Yuanli Wang, Roey Ben Chaim, Zonglin Di, Yipeng Gao, Junwei He, Yizhuo He, Liqiang Jing, Luyang Kong, Xin Lan, Jiachen Li, Songlin Li, Yijiang Li, Yueqian Lin, Xinyi Liu, Xu-anqing Liu, Haoran Lyu, Ze Ma, Bowei Wang, Runhui Wang, Tianfu Wang, Wengao Ye, Yue Zhang, Hanwen Xing, Yiqi Xue, Steven Dillmann, and Han-Chung Lee. Skillsbench: Benchmarking how well agent skills work across diverse tasks. *CoRR*, abs/2602.12670, 2026. [arXiv:2602.12670](https://arxiv.org/abs/2602.12670).
 - [13] Zi Liang, Qingqing Ye, Yanyun Wang, Sen Zhang, Yaxin Xiao, Ronghua Li, Jianliang Xu, and Haibo Hu. "yes, my lord." guiding language model extraction with locality reinforced distillation. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2025, Vienna, Austria, July 27 - August 1, 2025*, pages 1441–1465. Association for Computational Linguistics, 2025.
 - [14] Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, Shudan Zhang, Xiang Deng, Aohan Zeng, Zhengxiao Du, Chenhui Zhang, Sheng Shen, Tianjun Zhang, Yu Su, Huan Sun, Minlie Huang, Yuxiao Dong, and Jie Tang. Agentbench: Evaluating llms as agents. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net, 2024.
 - [15] Yuchen Ma, Yue Huang, Han Bao, Haomin Zhuang, Swadheen Shukla, Michel Galley, Xiangliang Zhang, and Stefan Feuerriegel. Skillgen: Verified inference-time agent skill synthesis. *CoRR*, abs/2605.10999, 2026. [arXiv:2605.10999](https://arxiv.org/abs/2605.10999).
 - [16] Qirui Mi, Zhijian Ma, Mengyue Yang, Haoxuan Li, Yisen Wang, Haifeng Zhang, and Jun Wang. Skillpro: Learning reusable skills from experience via non-parametric ppo for llm agents. In *Proceedings of the 43rd International Conference on Machine Learning (ICML 2026)*, 2026. Spotlight. URL: <https://arxiv.org/abs/2602.01869>.
 - [17] MiniMax. Minimax-m3: Frontier coding, 1m context, native multimodality. <https://www.minimax.io/models/text/m3>, 2026.
 - [18] Jingwei Ni, Yihao Liu, Xinpeng Liu, Yutao Sun, Mengyu Zhou, Pengyu Cheng, Dexin Wang, Erchao Zhao, Xiaoxi Jiang, and Guanjin Jiang. Trace2skill: Distill trajectory-local lessons into transferable agent skills. *CoRR*, abs/2603.25158, 2026. [arXiv:2603.25158](https://arxiv.org/abs/2603.25158).
 - [19] OpenAI. Gpt-5.6 sol. <https://developers.openai.com/api/docs/models/gpt-5.6-sol>, 2026.
 - [20] OpenAI. Introducing gpt-5.5. <https://openai.com/index/introducing-gpt-5.5/>, 2026.
 - [21] Tribhuvanesh Orekondy, Bernt Schiele, and Mario Fritz. Knockoff nets: Stealing functionality of black-box models. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2019, Long Beach, CA, USA, June 16-20, 2019*, pages 4954–4963. Computer Vision Foundation / IEEE, 2019.
 - [22] Qwen Team. Qwen3.6-35b-a3b. Model Card, 2026. Accessed August 2026.
 - [23] Yicong Tan, Xinyue Shen, Yun Shen, Michael Backes, and Yang Zhang. On the effectiveness of prompt stealing attacks on in-the-wild prompts. In *2025 IEEE Symposium on Security and Privacy (SP)*, pages 392–410, 2025. [doi:10.1109/SP61157.2025.00120](https://doi.org/10.1109/SP61157.2025.00120).
 - [24] Florian Tramèr, Fan Zhang, Ari Juels, Michael K. Reiter, and Thomas Ristenpart. Stealing machine learning models via prediction apis. In Thorsten Holz and Stefan Savage, editors, *25th USENIX Security Symposium, USENIX Security 16, Austin, TX, USA, August 10-12, 2016*, pages 601–618. USENIX Association, 2016.
 - [25] Chenxi Wang, Zhuoyun Yu, Xin Xie, Wuguannan Yao, Runnan Fang, Shuofei Qiao, Kexin Cao, Guozhou Zheng, Xiang Qi, Peng Zhang, and Shumin Deng. Skillx: Automatically constructing skill knowledge bases for agents. *CoRR*, abs/2604.04804, 2026. [arXiv:2604.04804](https://arxiv.org/abs/2604.04804).
 - [26] Xingyao Wang, Boxuan Li, Yufan Song, Frank F. Xu, Xiangru Tang, Mingchen Zhuge, Jiayi Pan, Yueqi Song, Bowen Li, Jaskirat Singh, Hoang H. Tran, Fuqiang Li, Ren Ma, Mingzhang Zheng, Bill Qian, Yanjun Shao, Niklas Muennighoff, Yizhe Zhang, Binyuan Hui, Junyang Lin, and et al. Openhands: An open platform for AI software developers as generalist agents. In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*. OpenReview.net, 2025.

- [27] Zihan Wang, Rui Zhang, Yu Liu, Chi Liu, Qingchuan Zhao, Hongwei Li, and Guowen Xu. Black-box skill stealing attack from proprietary LLM agents: An empirical study. *CoRR*, abs/2604.21829, 2026. [arXiv:2604.21829](#).
- [28] John Yang, Carlos E. Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. Swe-agent: Agent-computer interfaces enable automated software engineering. In *Advances in Neural Information Processing Systems 37: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*, 2024.
- [29] Yifan Yang, Ziyang Gong, Weiquan Huang, Qihao Yang, Ziwei Zhou, Zisu Huang, Yan Li, Xuemei Gao, Qi Dai, Bei Liu, Kai Qiu, Yuqing Yang, Dongdong Chen, Xue Yang, and Chong Luo. Skillopt: Executive strategy for self-evolving agent skills. *CoRR*, abs/2605.23904, 2026. [arXiv:2605.23904](#).
- [30] Yong Yang, Changjiang Li, Qingming Li, Oubo Ma, Haoyu Wang, Zonghui Wang, Yandong Gao, Wenzhi Chen, and Shouling Ji. PRSA: prompt stealing attacks against real-world prompt services. In Lujo Bauer and Giancarlo Pellegrino, editors, *34th USENIX Security Symposium, USENIX Security 2025, Seattle, WA, USA, August 13-15, 2025*, pages 2283–2302. USENIX Association, 2025.
- [31] Zhuoyun Yu, Xin Xie, Wuguannan Yao, Chenxi Wang, Lei Liang, Xiang Qi, and Shumin Deng. Skilladaptor: Self-adapting skills for LLM agents from trajectories. *CoRR*, abs/2606.01311, 2026. [arXiv:2606.01311](#).

A Prompts Used by the LLM Assistant

This appendix presents the prompts used by the LLM Assistant M_{aux} during the capability reconstruction process. The LLM assistant is employed only for offline analysis and skill refinement, including workflow abstraction, capability-deficit diagnosis, behavioral primitive extraction, and structured skill rewriting. These prompts guide the assistant to transform victim–attacker execution differences into reusable procedural knowledge while keeping the attacker model, agent framework, and tools unchanged.

Prompt Template for Workflow Abstraction

Role

You are a workflow-induction analyst who converts a successful Agent trajectory into reusable task guidance.

Task

First, divide the trajectory into chronological behavioral segments. Create a boundary only when the current work produces a definite result or the Agent enters a new phase.

A tool call alone does not create a segment. Preserve observed decisions, actions, results, failures, retries, and verification behavior without adding unobserved actions. Then abstract the segments into a general workflow. Remove run-specific commands, concrete tool names, temporary artifacts, values, and repairs, while retaining reusable procedures, dependencies, tool-use patterns, validation requirements, and recovery strategies.

For each workflow step:

- describe the required work, expected result, and acceptance criteria;
- associate it with exactly one skill and its relevant sections;
- split a step if it requires multiple skills;
- map cross-skill coordination, data transfer, and overall workflow-understanding requirements to the workflow skill;
- reserve an initially empty area for later refinement.

Represent the result using execution structure W_s , verification criteria V_s , skill mapping Φ_s , and refinement guidance $R_s^{(0)}$. The workflow must cover the complete task without introducing unsupported requirements.

Input

- instruction.
- target trajectory.
- skills.

Output

Return full .md context.

Prompt Template for Capability-Deficit Diagnosis

Role

You are a trajectory failure-diagnosis analyst who identifies behavioral gaps and routes them to responsible skill sections.

Task

Identify capability-critical failure patterns in the source trajectory, using the successful target trajectory when contrast is needed. Retain only behavioral gaps that contribute to task failure. Ignore differences caused by task-specific values, stylistic choices, or alternative successful paths.

For each failure:

- preserve the directly related source and target action chains;
- use the workflow to select the responsible skill and exactly one candidate section;
- route cross-skill coordination, data-transfer, and workflow-understanding failures to the workflow skill;
- describe the observable symptom, behavioral root cause, and root workflow phase;
- classify it as old only when the same behavioral gap matches a previously applied patch but remains insuffi-

cient; otherwise classify it as new;
 - do not determine new or old from the structure or content of the selected skill;
 - record the related patch identifier when applicable.
 Merge matching patterns into existing diagnoses by incrementing their counts, append new patterns with consecutive identifiers, and preserve unmatched records. Match patterns by behavioral gaps and routing information rather than similar symptoms.

Input

- workflow.
- instruction.
- skills with names and line numbers.
- existing failure diagnoses.
- previously applied patches.
- target trajectory.
- source trajectory.

Output

<Capability-Deficit Diagnosis_json schema>.

Prompt Template for Behavioral Primitive Extraction

Role

You are a capability-primitive extractor who converts failure diagnoses into minimal behavioral requirements.

Task

Rank diagnoses by decreasing count and increasing identifier, and process only the first three. Extract at most five executable, observable, and verifiable capability primitives.

For each primitive:

- describe the missing or insufficient behavior;
- specify the procedure required to realize or correct the behavior;
- state the expected observable improvement;
- copy the target skill, target section, and reconstruction status from the supporting diagnoses without reassessing them;
- combine diagnoses only when they support the same requirement and share these fields;
- preserve supporting diagnosis identifiers and complete action chains;
- assign strong evidence for a direct failed–successful contrast and medium evidence otherwise.

Do not reinterpret upstream diagnoses, change their reconstruction status, or write complete skill patches.

Input

- Stage 1 failure diagnoses.
- instruction.
- skills with names and line numbers.

Output

<Behavioral Primitive Extraction_json schema>.

Prompt Template for Structured Skill Rewriting

Role

You are an instructional-unit rewrite planner who converts capability primitives into skill patch plans.

Task

Group primitives by target skill and section, and restrict every edit to that section. Combine primitives in one patch only when they concern the same execution step and share the same inspection and failure-handling logic. Use <Structured Rewriting rules> to modify a matching instructional unit or create a new unit when no match exists. Treat the reconstruction status supplied by the primitives as fixed and do not reassess it.

Apply different rewriting methods according to the reconstruction status:

- for a new deficit, integrate the missing requirement into When to use, Goal, Procedure, or Required; do not add Execution guidance;
- for an old deficit, first add missing guidance to Procedure;
- if Procedure already contains the guidance but remains insufficient, add or improve an Execution subsection and reference it from Procedure.

Generate at most five minimal, non-overlapping insert-after or replace edits. Preserve stable headings, valid line numbers, and verbatim source text. Do not change the task contract or hard-code run-specific content.

Input

- instruction.
- target skill with line numbers.
- Stage 2 capability primitives.

Output

<Rewriting patches_json schema>.

B Example of a Reconstructed Skill

This appendix presents an example of a skill before and after reconstruction by *AgentLeak*. The example illustrates how victim–attacker execution differences are converted into targeted skill refinements through structured rewriting. The reconstructed skill preserves the original procedural knowledge while incorporating additional behaviors identified as necessary for closing the capability gap. The parts highlighted in red indicate modifications made to the original skill.

Original Skill

Step-by-Step Process

1. Load and clean data: Handle missing values, ensure proper time ordering
2. Convert to real terms: Deflate nominal values using appropriate price index
3. Apply log transformation: For positive level variables

4. Apply HP filter: Use appropriate λ for data frequency
5. Analyze cyclical component: Compute correlations, volatilities, etc.

Example: Business Cycle Correlation
import pandas as pd...

Reconstructed Skill

Step-by-Step Process

When to use

Use this process when nominal level series must be converted to real terms and detrended for business-cycle analysis, including tables where quarterly continuation rows omit the year.

Goal

Produce aligned, validated cyclical components from the full set of available observations and use them for correlation or volatility analysis.

Procedure

1. Load the source tables, select the required series, remove non-data rows, and sort observations chronologically.
2. Parse quarterly rows sequentially. When a label states a year, record it as the active year; assign continuation labels such as 'II.' or 'III p.' to that active year until another explicit year appears.
3. Group quarterly observations by their assigned year. For any year without a supplied annual value, collect every available quarter in that group and use their arithmetic mean as the annual value; do not stop after the first explicitly year-labeled quarter.
4. Inspect each quarterly group against the source row sequence. Confirm that every available quarter appears exactly once under the most recently explicit year and that the computed annual value uses the full group. Correct year propagation, duplicate rows, or omitted quarters before continuing.
5. Align each nominal series with the appropriate price index by year and convert it to real terms. When the price index is a ratio normalized to a reference year equal to 1, follow Deflation with a normalized price ratio.
6. Confirm that values to be logged are finite and strictly positive, then apply the natural logarithm.
7. Apply the HP filter with the smoothing parameter appropriate to the observation frequency.
8. Align the resulting cyclical components on the same years and compute the requested correlation, volatility, or other business-cycle statistic.
9. Inspect the aligned sample and final statistic. Confirm that the intended years are present once in each series, no missing or non-finite values remain, and the result is finite.

Execution: Deflation with a normalized price ratio

Required actions

1. Confirm that the price-index field is a ratio whose reference-year value equals 1.
2. Join each nominal observation to the price ratio for the same year.

If failed

1. Correct the year join, ratio interpretation, or division formula.
2. Recompute the complete real series and repeat the inspection.

Example: Business Cycle Correlation
import pandas as pd ...

C Task Selection and Instance Construction

Task selection. We select task scenarios to cover diverse application domains, difficulty levels, and skill compositions. We prioritize tasks that require non-trivial multi-step execution and specialized procedural knowledge, while also admitting meaningful variations within the same task family. Specifically, the selected tasks should allow their inputs, target objects, parameters, or application contexts to vary without substantially changing the underlying objective, workflow, tool usage, skill requirements, or evaluation semantics. The selected task scenarios are summarized in Table 2.

Task instance construction. For each selected scenario, we construct 30 task instances by introducing controlled variations to the original instance. We first identify the core task objective and the properties that must remain invariant, and then vary task-specific factors such as input contents, target objects, parameters, files, or application contexts. These variations are designed to produce realistic instances of the same task rather than new task families or superficial textual paraphrases. Throughout construction, we preserve the underlying workflow, tool interfaces, skill requirements, and evaluation criteria. In particular, the original raw skills remain unchanged across all generated instances. Variations that alter the required capability, introduce new workflow stages or tools, or otherwise drift from the original task are excluded.

Instance validation. Each generated instance undergoes both LLM-based validation and manual review. We verify that the instance remains within the original task family, that the intended variation is correctly realized, and that the resulting task is internally consistent and executable. Instances containing unintended structural changes, inconsistent modifications, or task drift are revised or discarded. This process ensures that the generated instances introduce meaningful within-scenario diversity while preserving the capability requirements of the original task, allowing us to evaluate whether reconstructed skills generalize to previously unseen instances.

Instance Examples. To illustrate the construction process, we present 2 representative scenarios covering document editing and cross-modal data reconciliation. For each scenario,

Table 8: Generalization directions for the PPTX Reference Formatting task.

Slot ID	Generalization Direction	Description
SS-01	Target Paper Identity	Replaces target papers and synchronizes the corresponding slide headings and abstracts.
SS-02	Title Repetition Topology	Redistributes title occurrences across slides, varying their frequencies and duplicate patterns.
SS-03	Initial Text-Box Geometry	Changes the initial positions and dimensions of target text boxes while preserving the required final placement.
SS-04	Initial Text Formatting	Varies fonts, sizes, colors, boldness, and run segmentation before normalization.
MS-01 (SS-02 + SS-04)	Repetition Topology and Formatting	Combines altered duplicate patterns with varied initial formatting across target text boxes.
MS-02 (SS-01 + SS-04)	Paper Identity and Formatting	Combines new paper content with varied initial formatting while preserving the slide structure.

Table 9: Generalization directions for the PDF–Excel Difference Detection task.

Slot ID	Generalization Direction	Description
SS-01	Field-Level Numeric Change	Changes one salary, tenure, or score value in the current workbook.
SS-02	Deleted Employee Set	Removes one or more complete employee records from the current workbook.
SS-03	Excel Row Ordering	Reorders complete workbook rows without changing record identities or values.
SS-04	Equivalent PDF Numeric Representation	Reformats numeric values in the PDF while preserving their parsed values.
MS-01 (SS-03 + SS-01)	Row Ordering and Field Modification	Combines workbook row reordering with one field-level value change.
MS-02 (SS-04 + SS-02)	Numeric Representation and Record Deletion	Combines equivalent PDF numeric formatting with additional deleted workbook records.

we summarize the approved generalization directions used to generate task instances rather than enumerating all 30 instances individually. A task package typically contains a task instruction, environment inputs, task-specific skills, a reference solution or oracle, and a test-based verifier. Each generated instance modifies only the files or task attributes associated with its designated variation dimensions, while preserving the core objective, execution requirements, and input–output contract.

In the following tables, **SS** denotes a single-slot direction, in which one variation dimension is modified, whereas **MS** denotes a multi-slot direction, in which two compatible dimensions are varied simultaneously.

1. PPTX Reference Formatting. The task requires standardizing designated paper titles in a presentation and compiling them into a deduplicated reference slide while preserving unrelated content. Table 8 summarizes the approved generalization directions for this scenario.

2. PDF–Excel Difference Detection. The task requires comparing employee records in a PDF backup and a current Excel workbook to identify deleted records and modified fields while ignoring superficial representation differences. Table 9 summarizes the approved generalization directions for this scenario.

D Detailed Scenario-level Results

This appendix provides detailed scenario-level performance results of *AgentLeak*. Figure 6 presents the IPR and CRR results on individual task scenarios, complementing the aggregated evaluation in the main paper.

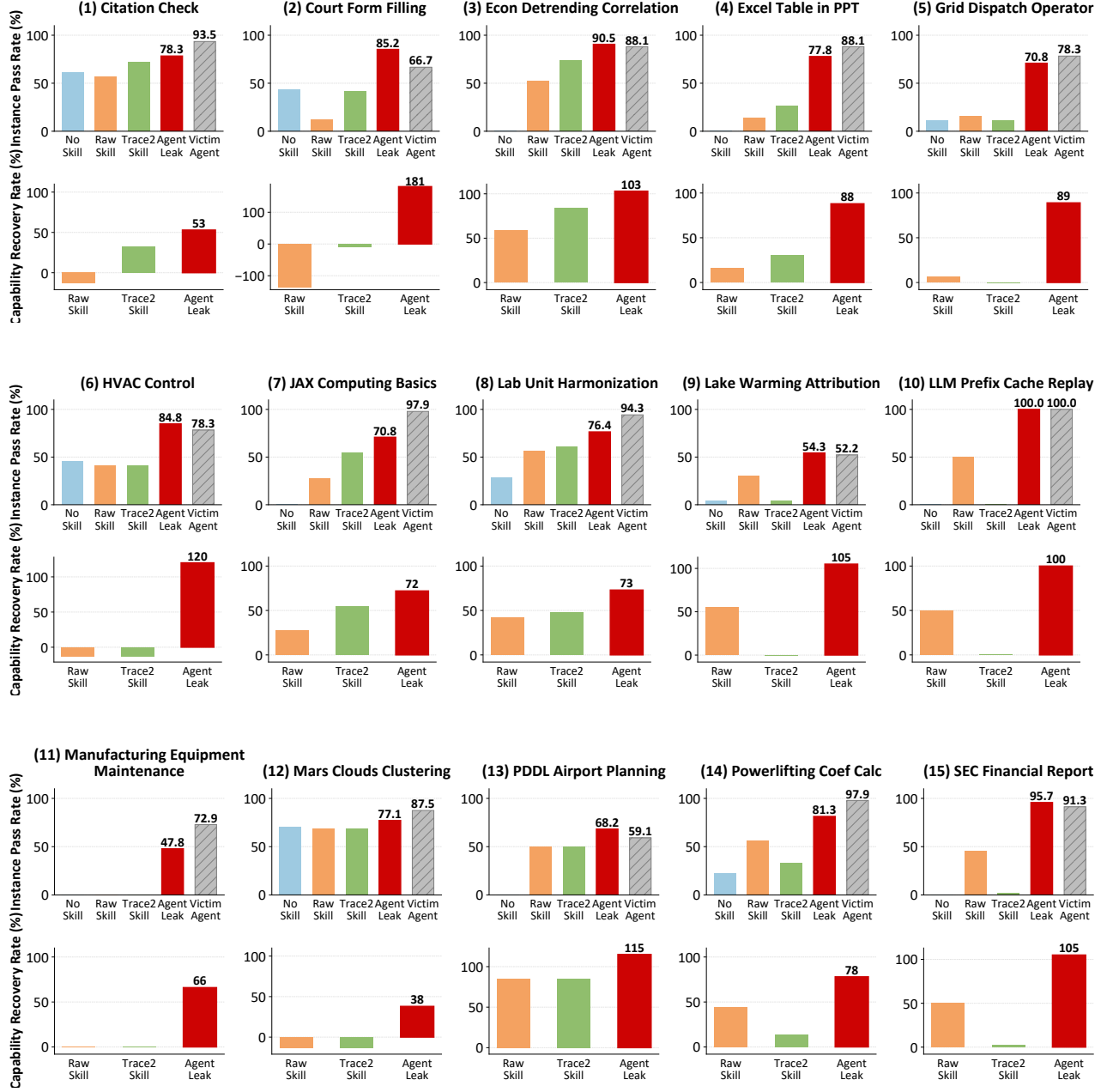


Figure 6: Capability cloning performance on representative task scenarios across different application domains.