SKILL-IN-SKILL: AGENT SKILLS THAT CARRY THEIR OWN EXECUTION CONDITIONS ACROSS DOMAINS

Anonymous authors
Paper under double-blind review

ABSTRACT

Agent skills provide LLMs with reusable procedural knowledge, but existing skills typically specify *what* to do without carrying the conditions that govern valid use. As a result, activation scope, authorization, and execution evidence remain external to the skill and may be lost when the skill is transferred or reused. We introduce **Skill-in-Skill (SIS)**, a portable skill architecture that binds a reusable capability with its governance contract as a single self-contained unit. SIS carries activation, authorization, execution, and verification conditions with the skill, and records runtime evidence to support independent validation of governed execution. We compare SIS with an **External Governance Sidecar (SIDECAR)** that keeps equivalent governance outside the skill package. Across open-weight models, SIS consistently reduces context-access and tool-use overhead, with up to 90.3% **fewer repeated task-context reads** and 13.2% **lower end-to-end latency**. In cross-boundary transfer tests, SIS preserves authorized execution after site or principal changes while blocking effects when destination governance is absent. These results show that binding governance to the capability improves the portability and efficiency of governed skill execution without weakening enforcement.

1 INTRODUCTION

Agent skills are reusable packages of instructions, scripts, and resources that agents load dynamically for specialized tasks (Ling et al., 2026; Xu & Yan, 2026). They provide LLM agents with modular procedural knowledge that can be discovered and used when needed (Destefanis, 2026; Li, 2026). However, correct skill activation and execution often depend on the external harness or environment (Han et al., 2026; Guo, 2026). Resource access likewise depends on runtime permission checks and enforcement (Pan et al., 2026; Shi et al., 2025).

This dependency becomes problematic when a skill moves across execution environments with different runtime interfaces (Ouyang et al., 2026). Under the current setting, the skill files can be transferred, but host-specific configuration and behavior remain tied to the original environment (Destefanis, 2026; Fan & Lan, 2026). A new environment may use different communication protocols, control points, or authorization mechanisms (Yang et al., 2025; Wang, 2026). As a result, the skill may no longer be used under the conditions assumed by its original environment. If a skill is activated under unsuitable new conditions (Guo, 2026), it may access resources beyond its intended scope (Schmotz et al., 2026) or execute without required checks (Dantanarayana et al., 2026).

For clarity, we distinguish two aspects of a skill: its **capability**, which captures what the skill enables an agent to do, and its **governance**, which specifies the conditions under which that capability may be used. We refer to a mismatch between these two aspects as the **Capability–Governance Gap**. In current skill deployments, capability is carried by the skill itself, while governance is often provided by the surrounding execution environment. Recent work explicitly treats skills as transferable artifacts, distilling trajectory-local lessons into reusable skills and optimizing skills for transfer across execution harnesses (Ni et al., 2026; Yang et al., 2026). As a result, when a skill is transferred, updated, or reused, the capability may move without the governance remaining aligned with it. A skill update may change the permissions it needs (Zhu & Wang, 2026), while delegation may place the same capability under a different authorization context (Xu et al., 2026). This gap makes portable execution difficult to govern consistently across environments.

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

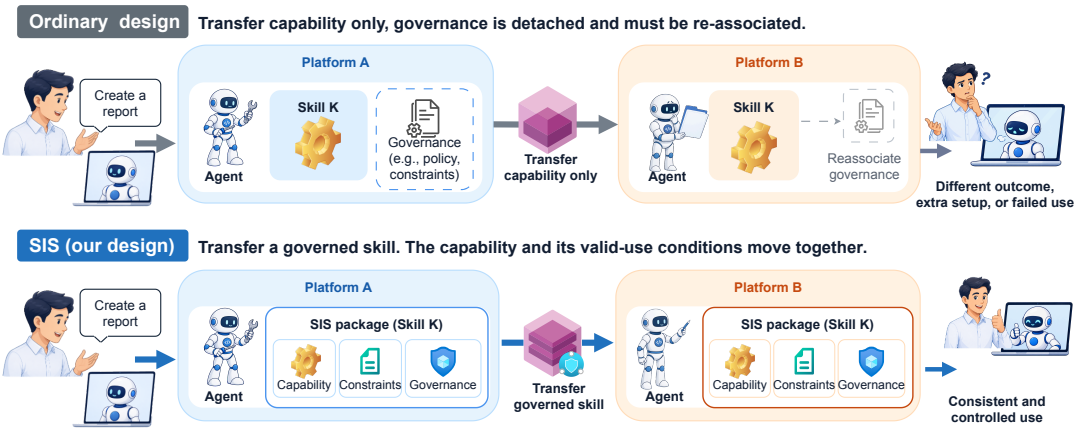


Figure 1: Cross-platform transfer under two packaging designs. **Top**: the capability moves separately from external governance, which must be re-associated at the destination and may require extra setup or lead to a different outcome or failed use. **Bottom**: SIS transfers the capability and its use conditions—constraints and governance—together as one governed skill, supporting consistent, controlled use at the destination.

A straightforward solution is to transfer the skill together with an external governance specification, such as a permission manifest, and re-associate the two at the destination (Pan et al., 2026; Shen et al., 2026). While this makes governance explicit, it still treats capability and governance as separate artifacts. Their association must be reconstructed at each destination and may break when interfaces, permissions, or runtime controls differ (Pan et al., 2026; Zhu & Wang, 2026). Thus, the **key challenge** is to preserve the binding between capability and governance across execution environments without relying on destination-specific reassociation.

To address this challenge, we propose a new skill format that packages a reusable capability together with the governance required for its valid use. The two are bound at creation and remain associated as the skill is shared, transferred, or reused. In this way, governance travels with the skill rather than being reconstructed at each destination, preserving valid use across execution boundaries.

We instantiate this new format as **Skill-in-Skill (SIS)**, a portable skill architecture that carries both the reusable capability and the governance required for its valid use. Each SIS specifies not only what the skill can do, but also when it may be activated, what resources it may access, how it should execute, and how its execution should be verified. These conditions remain attached to the skill across execution environments, allowing the receiving environment to determine whether the skill may proceed and whether its execution remains valid.

SIS realizes this design through three tightly coupled components: a capability $\mathbf{K}$, execution constraints $\mathcal{C}$, and governance rules $\mathcal{G}$. We denote the resulting package as $\mathrm{SIS}(\mathbf{K}, \mathcal{C}, \mathcal{G})$. Before execution, the destination evaluates whether the current task, scope, and authorization satisfy the carried conditions. Invalid requests are blocked before the capability is activated. For accepted requests, the constraints guide how the capability is materialized and executed. SIS then records runtime evidence of the resulting actions and effects. An independent verifier checks this evidence against the carried contract. This creates a complete lifecycle from activation to execution and verification, while keeping the governing conditions bound to the capability throughout.

Our contributions are:

- **Portable and Governed Skills.** We introduce Skill-in-Skill (SIS), a new scheme that enables reusable skills to carry and enforce their own conditions of valid use across execution environments.
- **Efficient Governance Execution.** We develop a self-contained governance mechanism that avoids repeated coordination with external sidecars and substantially improves execution efficiency.
- **Cross-Environment Skill Transfer.** We develop a destination-side validation mechanism that allows transferred skills to establish valid governance in new execution environments without relying on their original environment.

2 RELATED WORK

Skill artifacts, lifecycle, utility, and failure. Agent-skill surveys and software-oriented accounts describe skills as reusable behavioral artifacts with acquisition, lifecycle, composition, maintenance, and security concerns (Ling et al., 2026; Xu & Yan, 2026; Li, 2026; Destefanis, 2026; Fan & Lan, 2026). Orchestration and self-improvement systems make skills composable and revisable (Wang et al., 2026b; Robeyns et al., 2025; Lin et al., 2026). Benchmarks and live evaluations compare skill-enabled and baseline agents, while transfer studies show that induction granularity and representation affect reuse (Li et al., 2026; Kevin et al., 2026; Feng et al., 2026). Mechanism studies decompose triggering, procedural anchoring, compliance, boundary following, regressions, grounding displacement, verification displacement, and excessive procedure (Jiang et al., 2026; Han et al., 2026; Tank & Nama, 2026; Dong et al., 2026); SkillCoach and SkillShapley add process rubrics and step-level attribution (Zhu et al., 2026; Liu et al., 2026a). These works establish that skill utility is conditional and that aggregate success is incomplete. SIS adds the missing artifact boundary: validity conditions are part of the portable treatment object rather than an external assumption.

Runtime evidence, provenance, and attribution. SkillTrace audits expression, implementation, and operational provenance (Chen et al., 2026); Counterfactual Trace Auditing aligns with-skill and without-skill trajectories (Zhou et al., 2026); and runtime reconstruction and provenance surveys separate lifecycle stages, evidence grades, inferred diagnoses, and typed dependencies across heterogeneous agents (Gao, 2026; Wang et al., 2026c). TelemetrySuffBench shows that telemetry can preserve failure detection while losing origin-localization evidence (Zhu & Pu, 2026). A^2E, ECP, agent-native telemetry, ClawProBench, and cost–utility alignment extend this concern to standardized traces, lifecycle-aware evaluation, explicit status, inspectable evidence, and trajectory-level resource attribution (Wang et al., 2026a; Wattamwar et al., 2026; He & Yu, 2026; Xiao, 2026; Liu & Li, 2026). The shared lesson is that exposure, invocation, behavioral change, effect, and verification are distinct claims; SIS moves the validity contract upstream so that the evidence layer can test conditions carried by the capability.

Portable governance, authorization, and composition security. Proof-Carrying Agent Actions and Mandato make authorization, approval, receipts, and proof closure explicit at action or protocol chokepoints (Wang, 2026; Racioppi, 2026); MasDrift shows that delegated goals can survive while authorization boundaries drift; and ClosureBound binds transitive dependencies and operation-time effects rather than trusting a root identity (Xu et al., 2026; Zhu & Wang, 2026). Configuration and harness-risk studies treat runtime declarations and lifecycle coverage as governance objects (Xiao, 2026; Wattamwar et al., 2026). Security work exposes complementary risks from indirect prompt injection, skill injection, malicious skills, per-skill scanning, and scanner-passing compositions (Greshake et al., 2023; Yi et al., 2025; Zhan et al., 2024; Debenedetti et al., 2024; Schmotz et al., 2026; Chen et al., 2025; Xiang et al., 2026; Liu et al., 2026b). These systems motivate explicit scope, authorization, dependency, and composition boundaries. SIS connects the three perspectives at the skill boundary: it makes capability-plus-conditions portable, then uses paired evidence to test whether those conditions remain effective after transfer.

3 SKILL-IN-SKILL FORMULATION

We formalize a conventional skill package of instructions, scripts, and resources as a reusable capability $\mathbf{K}$, which forms the inner skill of **Skill-in-Skill (SIS)**. SIS extends this capability with a portable contract that governs its valid use. We denote the resulting package as

$$\mathcal{W}(\mathbf{K}) = \big(\mathbf{K}, \mathcal{C}, \mathcal{G}\big), \tag{1}$$

where $\mathcal{C}$ specifies how an admitted capability should execute, while $\mathcal{G}$ specifies whether that execution is permitted and how it should be verified. The package manifest records the identities and digests of all three components, preserving their association as the skill is transferred or reused.

In our implementation, $\mathcal{C}$ and $\mathcal{G}$ are defined when the skill package is created and serialized together with $\mathbf{K}$. They therefore travel as part of the same portable unit rather than being reconstructed from destination-specific configuration. We use *governance cost* to refer to the measured overhead of governed execution, including file reads, tool calls, token usage, and elapsed time.

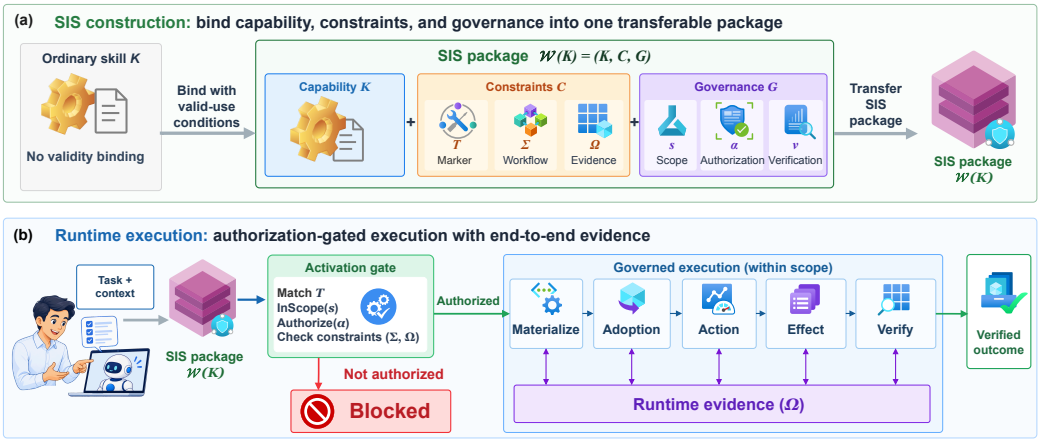


Figure 2: SIS package structure and execution lifecycle. (a) The portable package $\mathcal{W}(\mathbf{K})$ binds capability $\mathbf{K}$ with execution constraints $\mathcal{C}$ and governance rules $\mathcal{G}$. (b) At the destination, the carried contract determines whether the capability may be activated, constrains its execution, and defines the evidence required for verification.

3.1 Execution Constraints

The execution-constraint component is defined as

$$\mathcal{C} = (T, \Sigma, \Omega), \tag{2}$$

where T specifies task applicability, Σ specifies the expected execution workflow, and Ω specifies the runtime evidence that must be preserved.

T determines whether the capability is relevant to the current task, but does not itself grant execution authority. Σ defines the required execution structure, including ordering and milestone constraints, such as inspecting required context before an action or validating an effect before reporting completion. Ω defines the evidence surfaces that must remain observable during execution, including relevant tool calls, state changes, produced artifacts, and validation results.

These components serve different roles. T determines whether the capability is applicable, Σ constrains how it should execute, and Ω determines what evidence must be available for subsequent verification.

3.2 Governance Rules

The governance component is defined as

$$\mathcal{G} = (s, \alpha, v), \tag{3}$$

where s specifies the permitted execution scope, α specifies the authorization binding, and v specifies the verification interface.

The scope s defines the tasks, resources, workspace regions, and action classes for which the capability may be used. The authorization binding α identifies the principal, approval requirements, and delegation context under which execution is permitted. The verifier specification v defines which runtime evidence and terminal states must be checked independently of the executing agent.

This separation distinguishes applicability, authority, and correctness. A task may match T while falling outside s; it may fall within s while lacking the required authorization α; and it may satisfy both while still failing the verification requirements defined by v.

3.3 Destination-Side Activation and Execution

When a SIS package reaches a new execution environment, the destination runtime first validates the package manifest and reconstructs the bound tuple $(\mathbf{K}, \mathcal{C}, \mathcal{G})$. The destination then evaluates whether the capability is admissible under the current task and execution context.

Let t denote the task and c the execution context. We define activation as

$$\textsc{Activate}(\mathbf{K}, t, c) = \textsc{Match}(t, T) \wedge \textsc{InScope}(t, s) \wedge \textsc{Authorize}(c, \alpha). \tag{4}$$

The three terms correspond to applicability, scope, and authority, respectively. Execution proceeds only when all three conditions are satisfied. Otherwise, the capability is blocked before any state-changing action.

For an admitted request, the destination runtime materializes the resources required by $\mathbf{K}$ and applies the workflow constraints in Σ. State-mutating actions additionally require a destination-issued execution grant. At transfer destinations, the runtime validates package digests, checks actor scope and delegation provenance, and binds the resulting grant to the authorized capability and request. This preserves local control: the package carries its governance, but the destination retains authority over whether execution may proceed.

3.4 Evidence Lifecycle and Verification

We track execution through six evidence stages:

$$\textsc{Activate} \rightarrow \textsc{Materialize} \rightarrow \textsc{Adopt} \rightarrow \textsc{Act} \rightarrow \textsc{Effect} \rightarrow \textsc{Verify}.$$

These stages distinguish package availability from actual use. **Materialization** records that the required capability and resources are available to the agent, whereas **adoption** requires observable downstream use. We recognize adoption only when the runtime records a corresponding authorization chain rather than inferring it from prompt exposure or temporal proximity.

Actions and effects are also recorded separately. An action denotes an attempted operation, while an effect denotes an admitted state change. Each admitted effect must be associated with a valid execution grant. Unmatched or unauthorized writes are rejected before they can modify system state.

The verifier receives the carried verification specification v, the runtime evidence Ω, and the observed terminal state e_{final} . We write the resulting decision as

$$\textsc{Verify}(v, \Omega, e_{\text{final}}) \in \{\texttt{valid}, \texttt{invalid}, \texttt{unknown}\}, \tag{5}$$

where `unknown` is returned when the required evidence is incomplete. Terminal task success is therefore not used to infer missing authorization or adoption evidence.

This design provides an end-to-end governed execution path: the carried contract determines admissibility before activation, constrains execution after admission, and supplies the evidence required for independent verification. Capability and governance therefore remain bound from package construction through destination-side execution and final verification.

4 Experiments

4.1 Experimental Setup

Our evaluation investigates three core questions:

- **Agent behavior:** Does contract placement alter how agents gather execution context and information?
- **Efficiency generalization:** Do the observed efficiency gains persist across model scales and task families?
- **Governance portability:** Does the transferred package preserve authorized execution after moving to a new site?

We evaluate these questions through a controlled comparison between SIS and an *External Governance Sidecar* (SIDECAR). To isolate the effect of contract locality, both conditions share identical capabilities, contract definitions, tool interfaces, permissions, and compute budgets. The sole architectural divergence lies in placement: SIDECAR maintains governance externally, whereas SIS encapsulates the contract directly within the capability package. This controlled setup isolates how locality affects context-gathering overhead—disentangling targeted context lookups from baseline tool and file operations—and verifies whether self-contained packages successfully retain execution authority upon cross-boundary transfer.

Table 1: Ten representative business task classes spanning finance, support, legal, operations, and sales. Each class requires the agent to retrieve task-specific context, apply procedural or policy constraints, and produce a verifiable output.

Domain	Task class	Core operation
Finance	Reconciliation	Record matching and structured reconciliation
Finance	Journal preparation	Posting-rule application and balanced entries
Finance	Variance analysis	Actual–plan comparison and variance explanation
Support	Ticket triage	Routing-rule lookup and case assignment
Support	Customer escalation	Threshold checking and escalation handoff
Legal	NDA triage	Clause classification and exception routing
Legal	Compliance check	Policy matching and constraint verification
Operations	Change request	State, approval, and constraint checking
Operations	Capacity planning	Capacity assessment and constrained planning
Sales	Forecast	Input integration and verifiable forecasting

4.1.1 CONTROLLED WORKFLOW BENCHMARK

Our evaluation suite contains 120 frozen task instances across ten representative business workflow categories (Table 1). Each instance is an executable workflow rather than a static question-answer task: the agent must retrieve task-specific context, identify applicable rules or constraints, perform the required operation, and produce a verifiable artifact or decision. The ten categories differ in policy structure, input schema, and permitted actions, allowing us to test whether governance remains effective across heterogeneous execution requirements. At the same time, their inputs, mock APIs, expected outputs, and verification assertions are fixed, ensuring that SIS and SIDECAR are evaluated under identical conditions. This setup provides a controlled and reproducible testbed for measuring both governed execution and its operational overhead.

Each model is evaluated on the same 120 task instances to ensure a controlled cross-model comparison. Each of the ten categories contains 12 instances, with 8 used for confirmation and 4 reserved as held-out cases. Every task is executed three times under both SIDECAR and SIS, resulting in 720 runs per model and 2,160 runs in total. These repeats are used to measure execution variability and are not treated as independent tasks. Our analysis therefore focuses on paired comparisons between SIDECAR and SIS over the same underlying task instances.

4.1.2 PLATFORM AND MODELS

We evaluate our framework across representative open-weight model families spanning diverse parameter scales, from lightweight models to high-capacity flagships. Choosing open models provides the inspection access needed to trace granular agent behaviors and extend system tools, naturally aligning with the modular and portable design of SIS. All evaluations are orchestrated using DeepSeek Harness (DSH) (DeepSeek-AI, 2026b), an open-source Cordis-based runtime. DSH provides a unified runtime for skill loading, tool interaction, governance enforcement, and execution tracing across different model backbones. More details specifications on model checkpoints, deployment routes, and runtime hyperparameters are detailed in **Appendix B**.

Specifically, we benchmark two frontier open models alongside a lightweight alternative:

- **DeepSeek-V4.1-Flash** (DeepSeek-AI, 2026a): Developed by **DeepSeek**, this high-capacity model is used as the main backbone to evaluate SIS under complex execution settings, including long reasoning chains, multiple tool interactions, and transfers across execution boundaries.
- **MiMo-V2.6-Pro** (LLM-Core Xiaomi, 2026): Developed by the **Xiaomi MiMo Team**, this strong reasoning model is used to evaluate whether SIS generalizes across different model families and maintains consistent governance behavior after skill transfer.
- **Qwen3.8-27B** (Qwen Team, 2026): Developed by the **Qwen Team at Alibaba**, this smaller deployment-oriented model is used to evaluate whether SIS remains effective under reduced model capacity and whether its efficiency benefits persist in more resource-constrained settings.

4.1.3 Evaluation Metrics and Protocol

We evaluate performance across both **terminal outcomes** and **operational efficiency**. The evaluation metrics are summarized as follows:

- **Verified completion.** A run is counted as complete only when an independent verifier confirms that the terminal system state and produced artifacts satisfy the task specification. Incomplete runs are counted as failures.
- **Tool calls and context file reads.** Tool calls count all runtime tool invocations, while context file reads measure accesses to external task and execution context. These metrics capture the operational overhead required to complete a task.
- **Token usage.** We record fresh input tokens, cache-read tokens, and output tokens for each run. Together, they measure the language-model context and generation cost incurred during execution.
- **Latency.** We measure end-to-end wall-clock time from trial start to run completion. Additional process-level timing measurements are reported in Appendix C.

Each task setting is evaluated over 3 runs. We report paired contrasts between SIS and SIDECAR with 95% bootstrap confidence intervals over 10,000 resamples. The evaluation covers 120 tasks, including 80 confirmation tasks and 40 holdouts. Detailed metric definitions and coverage statistics are provided in **Appendix C**.

4.2 Execution Efficiency and Process Evidence

4.2.1 Cross-model efficiency

Table 2 summarizes the cross-model efficiency results. Across all three models, SIS consistently reduces **file reads** and **tool calls**, showing that its main benefit lies in lowering context-access and coordination overhead. MiMo (LLM-Core Xiaomi, 2026) further reduces token usage and wall-clock time, while Qwen3.8-27B (Qwen Team, 2026) shows the broadest gains, improving all seven measured resource indicators and reducing wall-clock time by 8.29 **seconds** (13.2%) per run. DeepSeek (DeepSeek-AI, 2026a) shows smaller gains and slight increases in several secondary metrics, indicating that the efficiency improvement is model-dependent. Overall, the results show that SIS consistently reduces governance-related access overhead, with broader end-to-end savings emerging on MiMo and Qwen.

Table 2: Cross-model resource changes from SIDECAR to SIS. Each entry reports the task-paired difference per run, computed as SIS minus SIDECAR, with the relative change from the SIDECAR mean shown in parentheses. Negative values indicate lower resource use with SIS. Confidence intervals and metric coverage are reported in Appendix C.

Outcome	DeepSeek (DeepSeek-AI, 2026a)	MiMo (LLM-Core Xiaomi, 2026)	Qwen3.8-27B (Qwen Team, 2026)
File reads ↓	−0.367 (−8.3%)	−0.278 (−7.6%)	−0.475 (−11.2%)
Tool calls ↓	−0.269 (−2.6%)	−0.847 (−8.8%)	−0.933 (−9.1%)
Agent steps ↓	+0.197 (+2.6%)	+0.001 (+0.0%)	−0.474 (−5.9%)
Fresh input tokens ↓	−143 (−1.2%)	−461 (−5.6%)	−734 (−5.6%)
Cache-read tokens ↓	+1,768 (+2.6%)	−1,886 (−3.7%)	−7,515 (−11.0%)
Output tokens ↓	+85 (+3.2%)	−123 (−6.2%)	−298 (−11.3%)
Wall-clock time ↓	+0.47 s (+2.6%)	−3.84 s (−6.6%)	−8.29 s (−13.2%)

On the 80-task confirmation split, the efficiency gains largely preserve verified completion across models. DeepSeek (DeepSeek-AI, 2026a) and MiMo (LLM-Core Xiaomi, 2026) maintain essentially unchanged success rates, while Qwen (Qwen Team, 2026) shows only a minor decrease. Additional process measurements further show that MiMo reduces early-stage context and drafting overhead, while Qwen achieves broader savings in tool use, token consumption, and runtime. Together with Table 2, these results indicate that SIS improves execution efficiency with little impact on verified completion.

Table 3: Process-level savings from binding governance to the skill. (a) Cumulative task-`context.json` reads across 360 runs per model and placement, where lower counts indicate less repeated context recovery. (b) MiMo (LLM-Core Xiaomi, 2026) lifecycle timing, with boot and exposure reported as phase durations and later milestones as cumulative time from run start. **Change** is relative to the SIDECAR baseline; negative values indicate fewer reads or lower latency.

(a) Repeated task-context reads

Model	SIDECAR	SIS	Change
DeepSeek (DeepSeek-AI, 2026a)	199	47	−76.4%
MiMo (LLM-Core Xiaomi, 2026)	93	9	−90.3%
Qwen3.8-27B (Qwen Team, 2026)	198	24	−87.9%

(b) MiMo lifecycle milestones

Milestone	SIDECAR	SIS	Change
Boot	88 ms	90 ms	+2.5%
Skill exposed	85 ms	87 ms	+2.6%
Skill loaded	11.5 s	10.6 s	−8.2%
`GET /input`	20.7 s	19.3 s	−6.5%
`POST /draft`	41.6 s	37.5 s	−10.0%
Execution complete	58.3 s	54.2 s	−7.1%
Run complete	58.4 s	54.2 s	−7.1%

Table 4: DeepSeek (DeepSeek-AI, 2026a) file reads across ten task classes from five application domains. SIDECAR and SIS denote mean file reads per run, while **Change** reports the relative reduction from the SIDECAR baseline. Lower values indicate lower context-access overhead; highlighted columns show the largest reductions.

Domain	**Finance**			**Support**		**Legal**		**Operations**		**Sales**
Task	**Recon.**	Journal	Variance	Ticket	Escalation	NDA	**Compliance**	**Change Req.**	**Capacity**	Forecast
SIDECAR	4.22	3.81	4.36	4.44	4.31	4.53	4.67	4.58	4.69	4.61
SIS	3.75	3.81	4.19	4.03	4.06	4.25	4.11	4.06	4.17	4.14
Change	−11.2%	0.0%	−3.8%	−9.4%	−5.8%	−6.1%	−11.9%	−11.5%	−11.2%	−10.2%

4.2.2 WHERE THE SAVED WORK OCCURS

Table 3 shows that the efficiency gain of SIS mainly comes from reducing repeated context access during execution. As shown in Table 3(a), reads of `task-context.json` decrease by 76.4% for DeepSeek, 90.3% for MiMo, and 87.9% for Qwen. This suggests that binding governance to the skill substantially reduces the need to repeatedly recover execution context from external files.

Table 3(b) further shows the corresponding runtime effect on MiMo. SIS introduces a small overhead during boot and skill exposure (+2.5% and +2.6%), but becomes faster once the skill is loaded. Skill loading is reduced by 8.2%, while subsequent execution milestones improve by 6.5%–10.0%. Overall run time decreases from 58.4 **s** to 54.2 **s** (7.1%). These results indicate that SIS trades a small initialization overhead for lower context-recovery cost during execution, leading to a net runtime saving.

4.2.3 CONSISTENCY ACROSS TASK CATEGORIES AND HELD-OUT INSTANCES

Table 4 further examines whether the efficiency gains of SIS are concentrated in only a few task types. For DeepSeek (DeepSeek-AI, 2026a), file reads decrease in nine of the ten task classes, with the largest reductions appearing in legal compliance, change requests, reconciliation, and capacity planning. This pattern suggests that the reduction in context-access overhead is broadly shared across task categories rather than driven by a small subset of tasks. Similar cross-class trends are observed for MiMo (LLM-Core Xiaomi, 2026) and Qwen (Qwen Team, 2026), both of which reduce tool calls across all ten classes. More detailed class-level comparisons and confidence intervals are provided in **Appendix D**.

The same trend is preserved on held-out instances from the ten task classes. File-read reductions remain consistent across all three models, while MiMo (LLM-Core Xiaomi, 2026) and Qwen (Qwen Team, 2026) also retain lower tool usage. These results indicate that the observed efficiency gains extend beyond the confirmation set rather than arising from a small number of task-specific cases. Detailed paired estimates and confidence intervals are provided in **Appendix F**.

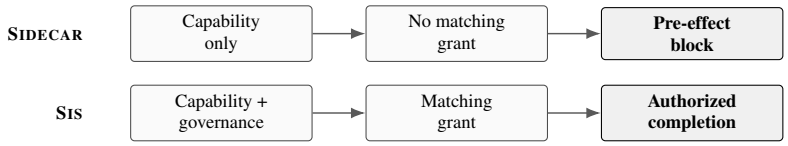


Figure 3: Cross-boundary governance protocol for SIDECAR and SIS. The destination independently validates the transferred capability and governance before issuing a local execution grant. SIDECAR arrives without the required governance binding, whereas SIS carries the capability and its governance together across the execution boundary.

Table 5: DeepSeek (DeepSeek-AI, 2026a) transfer outcomes. (a) Authorized completion and pre-effect blocking under site rebinding, principal delegation, and the no-governance control. (b) Effect-level enforcement for valid SIS transfers, reporting issued grants, grant-covered effects, and rejected writes. Each terminal-rate cell contains six trials.

(a) Terminal outcomes

	Authorized completion		Pre-effect block	
Boundary event	SIDECAR	SIS	SIDECAR	SIS
Site rebinding	0%	100%	100%	0%
Principal delegation	0%	100%	100%	0%
No governance	0%	0%	100%	100%

(b) SIS *effect-level enforcement*

	Grants, effects, and rejected writes		
Boundary event	Grants	Covered	Rejected
Site rebinding	6	6	0
Principal delegation	7	8	29
Total	13	14	29

4.3 GOVERNANCE PORTABILITY ACROSS BOUNDARIES

We evaluate whether capability–governance bindings remain valid when execution crosses site or principal boundaries. We consider two transfer scenarios: **site rebinding** and **principal delegation**. In both cases, the destination runtime performs local authorization before any state change. A transfer proceeds only when the carried capability, governance, and authorization evidence satisfy destination checks; otherwise, it is blocked before effect. Figure 3 illustrates the protocol, while Table 5 reports the outcomes.

Table 5 shows a clear difference in governance portability. After both site rebinding and principal delegation, SIS achieves **100% authorized completion**, whereas SIDECAR fails to complete after transfer. When destination governance is removed, both methods are blocked before effect. At the effect level, all realized effects under SIS are grant-covered, while unmatched writes are rejected before changing system state.

All realized effects are grant-covered, and all unmatched writes are blocked before effect.

5 DISCUSSION AND CONCLUSION

Our findings demonstrate that **governance placement is a fundamental architectural variable**, even when the underlying policy remains unchanged. By co-locating operational contracts with capability packages, SIS reduces redundant context acquisition across diverse deployments: high-capacity models require fewer exploratory tool calls, cross-lineage models consume fewer tokens, and compact models translate lower overhead into wall-clock speedups. At the same time, SIS enables portable governance through a clear division of responsibility: the skill carries its contractual bounds, while the destination runtime retains authority over whether effects are admitted. This allows destination hosts to validate transferred capabilities locally without relying on brittle external reassociation.

More broadly, treating skills as unconstrained prompts limits both portability and accountability. SIS instead represents a skill as a self-contained unit in which capability and governance remain bound across execution boundaries. This design preserves local enforcement while supporting verifiable transfer and auditable execution. As agent systems increasingly operate across administrative and multi-tenant environments, binding governance directly to portable capabilities provides a practical foundation for accountable and transferable agent execution.

AI USE STATEMENT

A generative AI assistant assisted with manuscript writing, figure construction, and LaTeX formatting. Numerical claims were checked against the frozen analysis exports and independent verification records; the assistant did not generate experimental outcomes.

REFERENCES

Jialuo Chen, Minghe Wang, Lingqi Jiang, Jianan Ma, Xinhao Deng, Xiaohu Du, Ruixiao Lin, Yunhao Feng, Linkang Du, and Jingyi Wang. SkillTrace: Multi-Trace Provenance Auditing for LLM-Agent Skill Reuse. arXiv preprint arXiv:2608.05204, 2026. URL https://arxiv.org/abs/2608.05204.

Sizhe Chen, Arman Zharmagambetov, David Wagner, and Chuan Guo. Meta SecAlign: A Secure Foundation LLM Against Prompt Injection Attacks. arXiv preprint arXiv:2507.02735, 2025. URL https://arxiv.org/abs/2507.02735.

Jayanaka Dantanarayana, Savini Kashmira, Lingjia Tang, and Jason Mars. SIGIL: Compiling Agent Skills into Typed Harnesses. arXiv preprint arXiv:2607.27309, 2026. URL https://arxiv.org/abs/2607.27309.

Edoardo Debenedetti, Jie Zhang, Mislav Balunović, Luca Beurer-Kellner, Marc Fischer, and Florian Tramèr. AgentDojo: A Dynamic Environment to Evaluate Prompt Injection Attacks and Defenses for LLM Agents. arXiv preprint arXiv:2406.13352, 2024. URL https://arxiv.org/abs/2406.13352.

DeepSeek-AI. DeepSeek-V4.1-Flash: Pushing the Limits of KV Cache Compression. arXiv preprint arXiv:2609.19969, 2026a. URL https://arxiv.org/abs/2609.19969.

DeepSeek-AI. DeepSeek Harness: Everything Is a Plugin. Official software repository, 2026b. URL https://github.com/deepseek-ai/deepseek-harness.

Giuseppe Destefanis. Authoring Agent Skills: A Software-Engineering Approach. arXiv preprint arXiv:2607.25032, 2026. URL https://arxiv.org/abs/2607.25032.

Gen Dong, Yanjie Gao, Liqun Li, Tianyin Xu, Yu Hua, and Fan Yang. Agent Skills Can Be Harmful: An Empirical Study of Skill-Induced Failures in LLM Agents. arXiv preprint arXiv:2608.11888, 2026. URL https://arxiv.org/abs/2608.11888.

Haodi Fan and Zucong Lan. Skillware: A Software Ontology and Engineering Lifecycle for Persistent Behavioral Artifacts. arXiv preprint arXiv:2607.18970, 2026. URL https://arxiv.org/abs/2607.18970.

Yiyang Feng, Biddut Sarker Bijoy, Niranjan Balasubramanian, and Jiawei Zhou. Break It Down, Pass It On: Cross-Task Skill Transfer in LLM Agents. arXiv preprint arXiv:2608.20274, 2026. URL https://arxiv.org/abs/2608.20274.

Xueping Gao. Evidence-Calibrated Runtime Reconstruction for Agent Skills Across Heterogeneous Coding Agents. arXiv preprint arXiv:2608.08793, 2026. URL https://arxiv.org/abs/2608.08793.

Kai Greshake, Sahar Abdelnabi, Shailesh Mishra, Christoph Endres, Thorsten Holz, and Mario Fritz. Not what you've signed up for: Compromising Real-World LLM-Integrated Applications with Indirect Prompt Injection. arXiv preprint arXiv:2302.12173, 2023. URL https://arxiv.org/abs/2302.12173.

Shuang Guo. SkillApt: Learning When to Activate Agent Skills from Counterfactual Evidence. arXiv preprint arXiv:2609.26863, 2026. URL https://arxiv.org/abs/2609.26863.

Jinyi Han, Yuanjian Xu, Ying Liao, Xinyi Wang, Zishang Jiang, Zixiang Di, Fanyang Lu, Zhichao Hu, and Yanghua Xiao. Skill-Use: Can LLMs Actually Use Skills in Agentic Harnesses? arXiv preprint arXiv:2608.04828, 2026. URL https://arxiv.org/abs/2608.04828.

Jun He and Deying Yu. Agent-Native Telemetry: Verifiable State-Delta Evidence for Autonomous Operations. arXiv preprint arXiv:2608.16178, 2026. URL https://arxiv.org/abs/2608.16178.

Zhiyuan Jiang, Fangrui Huang, Hanwen Xing, Xander Wu, Yipeng Gao, Rui Cao, Mengdi Wang, Shilong Liu, and Yijiang Li. Demystifying Agent Skills: Why They Work-Until They Don't. arXiv preprint arXiv:2608.14036, 2026. URL https://arxiv.org/abs/2608.14036.

Christopher Kevin, Narendran Raghavan, Jean-Francois Puget, Roshni Malani, Meghana Puvvadi, Moshe Abramovitch, Mohit Gupta, Rama Akkiraju, Subodh Prabhu, Yogesh Dangi, Wei Luo, and Seong Hee Lee. Evaluating Skills, Not Just Agents: Agentic Continuous Evaluation of Skills. arXiv preprint arXiv:2608.20614, 2026. URL https://arxiv.org/abs/2608.20614.

Xiangyi Li, Yimin Liu, Wenbo Chen, Bingran You, Zonglin Di, Yifeng He, Shenghan Zheng, Kyoung Whan Choe, Jiankai Sun, Shuyi Wang, Chujun Tao, Binxu Li, Xuandong Zhao, Hejia Geng, Xiaojun Wu, Junwei Zhou, Xiaokun Chen, Hanwen Xing, Yubo Li, Qunhong Zeng, Di Wang, Yuanli Wang, Roey Ben Chaim, Penghao Jiang, Haotian Shen, Luyang Kong, Xinyi Liu, Runhui Wang, Xuanqing Liu, Jiachen Li, Xin Lan, Yueqian Lin, Wengao Ye, Junwei He, Songlin Li, Yue Zhang, Yipeng Gao, Yijiang Li, Ze Ma, Liqiang Jing, Tianyu Wang, Kaixin Li, Yiqi Xue, Haoran Lyu, Yizhuo He, Yuchen Tian, Shutong Wu, Bowei Wang, Yixuan Gao, Bo Chen, Litong Liu, Sikai Cheng, Jiajun Bao, Shuaicheng Tong, Shuwen Xu, Terry Yue Zhuo, Tinghan Ye, Qi Qi, Miao Li, Longtai Liao, Zelin Tan, Chang Shi, Xilin Tang, Srinath Tankasala, Boqin Yuan, Yaoyao Qian, Jianhong Tu, Chenguang Wang, Yizhou Sun, Wei Wang, Aaron Taylor, Ziyue Yang, Changkun Guan, Zhikang Dong, Xinyu Zhang, Steven Dillmann, Han-chung Lee, and Dawn Song. SkillsBench: Benchmarking How Well Agent Skills Work Across Diverse Tasks. arXiv preprint arXiv:2602.12670, 2026. URL https://arxiv.org/abs/2602.12670.

Yubo Li. Dynamic Agent Skills: A Lifecycle Survey and Taxonomy of Evolving Skill Libraries. *Transactions on Machine Learning Research*, 2026. URL https://openreview.net/forum?id=cjU3YbcRr8.

Huawei Lin, Peng Li, Jie Song, Fuxin Jiang, and Tieying Zhang. MUSE-Autoskill: Self-Evolving Agents via Skill Creation, Memory, Management, and Evaluation. arXiv preprint arXiv:2605.27366, 2026. URL https://arxiv.org/abs/2605.27366.

George Ling, Shanshan Zhong, and Richard Huang. Agent Skills: A Data-Driven Analysis of Claude Skills for Extending Large Language Model Functionality. arXiv preprint arXiv:2602.08004, 2026. URL https://arxiv.org/abs/2602.08004.

Chang Liu, Yuqi Zhang, Yiman Zhong, Boyi Liu, Hengjun Wang, and Shuyue Wei. SkillShapley: Boundary-Adaptive Shapley Valuation for Skill Step Attribution in LLM Agents. arXiv preprint arXiv:2608.13173, 2026a. URL https://arxiv.org/abs/2608.13173.

Dan Liu and Jian Li. Cost-Utility Alignment in LLM Agent Trajectories: Profiling, Attribution, Diagnosis, Adaptation, and Evaluation. arXiv preprint arXiv:2608.26195, 2026. URL https://arxiv.org/abs/2608.26195.

Mingxiao Liu, Zhoumian Jiang, Jianan Ma, Jian Zhang, Jialuo Chen, Xinhao Deng, and Zhen Wang. CompoSkill: Compositional Skill Chain Attacks from Individually Scanner-Passing LLM Agent Skills. arXiv preprint arXiv:2608.16246, 2026b. URL https://arxiv.org/abs/2608.16246.

LLM-Core Xiaomi. MiMo-V2.6: Scaling Reinforcement Learning Towards Self-Improvement. Technical report, Xiaomi, 2026. URL https://huggingface.co/XiaomiMiMo/MiMo-V2.6-Pro-RL/resolve/main/MiMo_V2_6_technical_report.pdf.

Jingwei Ni, Yihao Liu, Xinpeng Liu, Yutao Sun, Mengyu Zhou, Pengyu Cheng, Dexin Wang, Erchao Zhao, Xiaoxi Jiang, and Guanjun Jiang. Trace2skill: Distill trajectory-local lessons into transferable agent skills. *arXiv preprint arXiv:2603.25158*, 2026.

Yipeng Ouyang, Yi Xiao, Yuhao Gu, and Xianwei Zhang. SkCC: Portable and Secure Skill Compilation for Cross-Framework LLM Agents. arXiv preprint arXiv:2605.03353, 2026. URL https://arxiv.org/abs/2605.03353.

Shidong Pan, Xiaoyu Sun, Tianyi Zhang, Dianshu Liao, Kaiwen Yang, and Zhenchang Xing. SkillGuard: A Permission-Centric Framework for Agent Skill Security. arXiv preprint arXiv:2606.03024, 2026. URL https://arxiv.org/abs/2606.03024.

Qwen Team. Qwen3.8-Max: A New Bar for Coding and Cowork. Official Qwen release article, August 2026. URL https://qwen.ai/blog?id=qwen3.8.

Giovanni Racioppi. Mandato: Protocol-Level Enforcement of Digitally Signed Mandates on AI Agent Actions with Cryptographically Chained Audit Trails. arXiv preprint arXiv:2608.14074, 2026. URL https://arxiv.org/abs/2608.14074.

Maxime Robeyns, Martin Szummer, and Laurence Aitchison. A Self-Improving Coding Agent. arXiv preprint arXiv:2504.15228, 2025. URL https://arxiv.org/abs/2504.15228.

David Schmotz, Luca Beurer-Kellner, Sahar Abdelnabi, and Maksym Andriushchenko. Skill-Inject: Measuring Agent Vulnerability to Skill File Attacks. arXiv preprint arXiv:2602.20156, 2026. URL https://arxiv.org/abs/2602.20156.

Tingda Shen, Yebo Feng, Konglin Zhu, Xiaojun Jia, Yang Liu, and Lin Zhang. Sealing the audit–runtime gap for LLM skills. *arXiv preprint arXiv:2605.05274*, 2026.

Tianneng Shi, Jingxuan He, Zhun Wang, Hongwei Li, Linyu Wu, Wenbo Guo, and Dawn Song. Progent: Securing AI Agents with Privilege Control. arXiv preprint arXiv:2504.11703, 2025. URL https://arxiv.org/abs/2504.11703.

Yifan Shi, Wei Zhang, and Tianyi Cui. A Programming Paradigm for Spatiotemporal Composability. arXiv preprint arXiv:2608.25512, 2026. URL https://arxiv.org/abs/2608.25512.

Darshan Tank and Baran Nama. The Regression Tax: Decomposing Why Skills Help and Hurt LLM Agents. arXiv preprint arXiv:2607.22520, 2026. URL https://arxiv.org/abs/2607.22520.

Haoning Wang, Mingxun Zhang, Chenyue Yu, Yingjun Shang, Xing Xie, Xia Hu, Guanchu Wang, and Na Zou. A^2E : An End-to-End Agent Auditing Engine. arXiv preprint arXiv:2608.07346, 2026a. URL https://arxiv.org/abs/2608.07346.

Jiayu Wang, Yifei Ming, Zixuan Ke, Shafiq Joty, Aws Albarghouthi, and Frederic Sala. SkillOrchestra: Learning to Route Agents via Skill Transfer. arXiv preprint arXiv:2602.19672, 2026b. URL https://arxiv.org/abs/2602.19672.

Yiqi Wang, Jiaqi Zhang, Zhangkai Wu, Taotao Cai, Zirui Liu, Zequn Sun, Manqing Dong, Mingkai Zheng, Yiqun Duan, Xuefei Yin, and Yanming Zhu. From Agent Traces to Trust: A Survey of Evidence Tracing and Execution Provenance in LLM Agents. arXiv preprint arXiv:2606.04990, 2026c. URL https://arxiv.org/abs/2606.04990.

Zexun Wang. Proof-Carrying Agent Actions: Model-Agnostic Runtime Governance for Heterogeneous Agent Systems. arXiv preprint arXiv:2606.04104, 2026. URL https://arxiv.org/abs/2606.04104.

Aniket Wattamwar, Manav Anandani, and Mrunal Kakirwar. The Evaluation Context Protocol (ECP): A Portable Contract for AI Agent Evaluation. arXiv preprint arXiv:2608.19263, 2026. URL https://arxiv.org/abs/2608.19263.

Chong Xiang, Drew Zagieboylo, Shaona Ghosh, Sanjay Kariyappa, Kai Greshake, Hanshen Xiao, Chaowei Xiao, and G. Edward Suh. Architecting Secure AI Agents: Perspectives on System-Level Defenses Against Indirect Prompt Injection Attacks. arXiv preprint arXiv:2603.30016, 2026. URL https://arxiv.org/abs/2603.30016.

YuanHang Xiao. ClawProBench: Trace-Aware Evaluation of AI Agents with Runtime Coverage and Frozen Workplace-Style Holdouts. arXiv preprint arXiv:2608.22510, 2026. URL https://arxiv.org/abs/2608.22510.

Renjun Xu and Yang Yan. Agent Skills for Large Language Models: Architecture, Acquisition, Security, and the Path Forward. arXiv preprint arXiv:2602.12430, 2026. URL https://arxiv.org/abs/2602.12430.

Zhuoning Xu, Xiucheng Zhang, Hanjun Luo, Yingbin Jin, Yinpeng Dong, and Hanan Salam. MasDrift: Benchmarking Authorization Preservation Across Multi-Agent Architectures. arXiv preprint arXiv:2608.07556, 2026. URL https://arxiv.org/abs/2608.07556.

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. *arXiv preprint arXiv:2605.23904*, 2026.

Yingxuan Yang, Huacan Chai, Yuanyi Song, Siyuan Qi, Muning Wen, Ning Li, Junwei Liao, Haoyi Hu, Jianghao Lin, Gaowei Chang, Weiwen Liu, Ying Wen, Yong Yu, and Weinan Zhang. A Survey of AI Agent Protocols. arXiv preprint arXiv:2504.16736, 2025. URL https://arxiv.org/abs/2504.16736.

Jingwei Yi, Yueqi Xie, Bin Zhu, Emre Kiciman, Guangzhong Sun, Xing Xie, and Fangzhao Wu. Benchmarking and Defending against Indirect Prompt Injection Attacks on Large Language Models. In *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining V.1*, pp. 1809–1820, 2025. doi: 10.1145/3690624.3709179. URL https://doi.org/10.1145/3690624.3709179.

Qiusi Zhan, Zhixiang Liang, Zifan Ying, and Daniel Kang. InjecAgent: Benchmarking Indirect Prompt Injections in Tool-Integrated Large Language Model Agents. In *Findings of the Association for Computational Linguistics: ACL 2024*, pp. 10471–10506, 2024. doi: 10.18653/v1/2024.findings-acl.624. URL https://aclanthology.org/2024.findings-acl.624/.

Xiaolin Zhou, Jinbo Liu, Li Li, Ryan A. Rossi, and Xiyang Hu. Counterfactual Trace Auditing of LLM Agent Skills. arXiv preprint arXiv:2605.11946, 2026. URL https://arxiv.org/abs/2605.11946.

Genliang Zhu and Chu Wang. Versioned Transitive Dependency-Closure Binding and Operation-Time Effect Governance for Agent Skills: ClosureBound. arXiv preprint arXiv:2609.05920, 2026. URL https://arxiv.org/abs/2609.05920.

Jiayin Zhu, Kelong Mao, Yudong Guo, Dengbo He, Sulong Xu, Simiu Gu, and Yutao Yue. SkillCoach: Self-Evolving Rubrics for Evaluating and Enhancing Agentic Skill-Use. arXiv preprint arXiv:2607.01874, 2026. URL https://arxiv.org/abs/2607.01874.

Yuxuan Zhu and Peng Pu. TelemetrySuffBench: Is Agent Telemetry Sufficient for Failure-Origin Diagnosis? arXiv preprint arXiv:2608.07899, 2026. URL https://arxiv.org/abs/2608.07899.

A Task Suite and Verifiers

The ten task categories in Table 1 span four fixture strata: normal, boundary, conflict, and reject. Each category has three instances in each stratum; eight instances form the confirmation split and four are held out. Every instance has a frozen input, expected artifact, scope, and independent verifier. A reconciliation task, for example, reads fixed records, applies reconciliation rules, and submits a structured result. The other workflows follow the same read–transform–submit–verify shape with different constraints and artifacts.

Table 6: Shared verifier checks applied to both placements. Effect authorization concerns accepted effects, not the absence of attempted actions.

Check	Verified property
Artifact correctness	Output matches the frozen task oracle.
Actor identity	The submitted actor is the expected task principal.
Input-before-write order	Required input was retrieved before a write.
Effect authorization	No disallowed action was accepted as an effect.
Final-state correctness	Final business state matches the expected state.
Output schema	Output fields have the declared types.
Write scope	Any accepted write is within the task's scope.

The transfer protocol additionally checks destination grants and terminal effects (Appendix G).

B Models and Runtime Configuration

Table 7: Frozen hosted routes for the three models (DeepSeek-AI, 2026a; LLM-Core Xiaomi, 2026; Qwen Team, 2026).

Paper name	Recorded route / provider	Experimental role
DeepSeek-V4.1-Flash (DeepSeek-AI, 2026a)	`deepseek-flash` Official provider route	Strong-model process and boundary transfer
MiMo-V2.6-Pro (LLM-Core Xiaomi, 2026)	`xiaomi/mimo-v2.6-pro` OpenRouter	Second strong-model comparison
Qwen3.8-27B (Qwen Team, 2026)	`qwen3.8-27B` Aliyun MaaS	Smaller-model resource comparison

All three models use DeepSeek Harness (DSH) and the same local-process task harness (DeepSeek-AI, 2026b; Shi et al., 2026). Frozen run settings allow up to 90 requests, 32,768 output tokens, and one retry.

C Cross-Model Governance Cost Savings

SIS reduces file reads and tool calls across all three deployments. Qwen (Qwen Team, 2026) extends these gains to all seven measured resource indicators, including a 13.2% reduction in wall-clock time; MiMo (LLM-Core Xiaomi, 2026) also reduces fresh-input and output tokens, time to the first draft request, and peak per-request context. Table 8 reports the exact paired estimates behind Figure 4. Each estimate averages three repeats within each of the 120 shared tasks before comparing SIS with SIDECAR; relative changes use the corresponding SIDECAR mean.

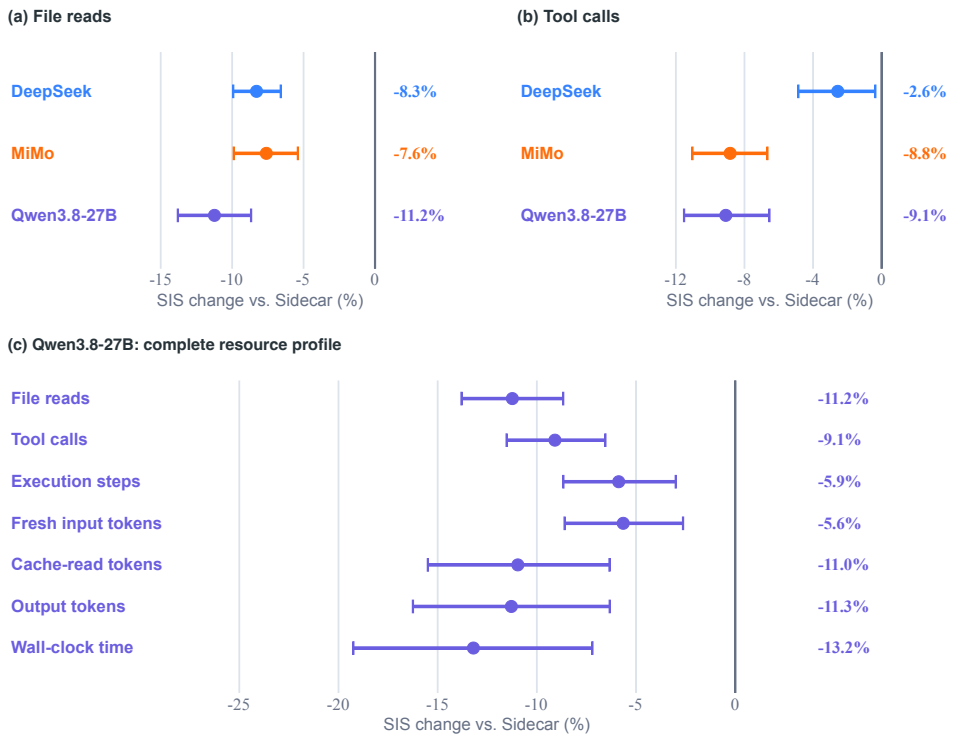


Figure 4: Paired governance cost changes with 95% intervals. (a) File reads and (b) tool calls across all three models. (c) Seven measured resource indicators for Qwen3.8-27B (Qwen Team, 2026). Points and interval endpoints are divided by each model's SIDECAR mean; negative values indicate lower use with SIS. Model colors match the main figures.

Table 8: Paired SISminus-SIDECAR resource differences per run. The first two measures compare every model; the remaining Qwen (Qwen Team, 2026) rows complete its seven-measure profile. The final column divides each point difference by that model's SIDECAR mean.

Model	Measure	SIDECAR mean	Difference [95% CI]	Change
DeepSeek (DeepSeek-AI, 2026a)	File reads	4.42	-0.367 $[-0.439, -0.292]$	-8.3%
	Tool calls	10.52	-0.269 $[-0.511, -0.039]$	-2.6%
MiMo (LLM-Core Xiaomi, 2026)	File reads	3.65	-0.278 $[-0.361, -0.197]$	-7.6%
	Tool calls	9.59	-0.847 $[-1.058, -0.639]$	-8.8%
Qwen3.8-27B (Qwen Team, 2026)	File reads	4.23	-0.475 $[-0.583, -0.367]$	-11.2%
	Tool calls	10.27	-0.933 $[-1.183, -0.672]$	-9.1%
	Execution steps	8.06	-0.474 $[-0.699, -0.242]$	-5.9%
	Fresh input tokens	13,006	-734 $[-1119, -342]$	-5.6%
	Cache-read tokens	68,578	$-7{,}515$ $[-10626, -4345]$	-11.0%
	Output tokens	2,642	-298 $[-429, -167]$	-11.3%
	Wall-clock seconds	62.78	-8.29 $[-12.09, -4.52]$	-13.2%

Table 9: Additional resource and process savings for MiMo (LLM-Core Xiaomi, 2026). Relative reductions use the corresponding SIDECAR mean; peak context is reported as an absolute paired difference.

Measure	SIS minus SIDECAR [95% CI]	Reduction
Fresh input tokens	-461 $[-879, -32]$	5.6%
Output tokens	-123 $[-230, -15]$	6.2%
Time to first draft request	-3.96 $[-7.47, -0.52]$ seconds	9.5%
Peak per-request context	-512 $[-912, -97]$ tokens	–

Verified completions among all registered runs are 360/360 → 360/360 for DeepSeek (DeepSeek-AI, 2026a), 355/360 → 356/360 for MiMo (LLM-Core Xiaomi, 2026), and 354/360 → 355/360 for Qwen (Qwen Team, 2026) (SIDECAR → SIS). Resource coverage is 99.0–100% across the reported metric groups and models; missing process values are not imputed. The first-draft measure is elapsed time from trial start to the first `/draft` request.

D TASK-CLASS RESOURCE PROFILE

SIS lowers file reads in 28 of 30 model–class comparisons, with no change in the other two. Qwen (Qwen Team, 2026) reduces reads in all ten classes; MiMo (LLM-Core Xiaomi, 2026) and Qwen (Qwen Team, 2026) reduce tool calls in all ten. Figure 5 displays the complete task-class pattern in the order of Table 1. The heatmaps divide each paired estimate by its class-specific SIDECAR mean; the Qwen (Qwen Team, 2026) token and time panels show absolute paired differences with class-bootstrap intervals.

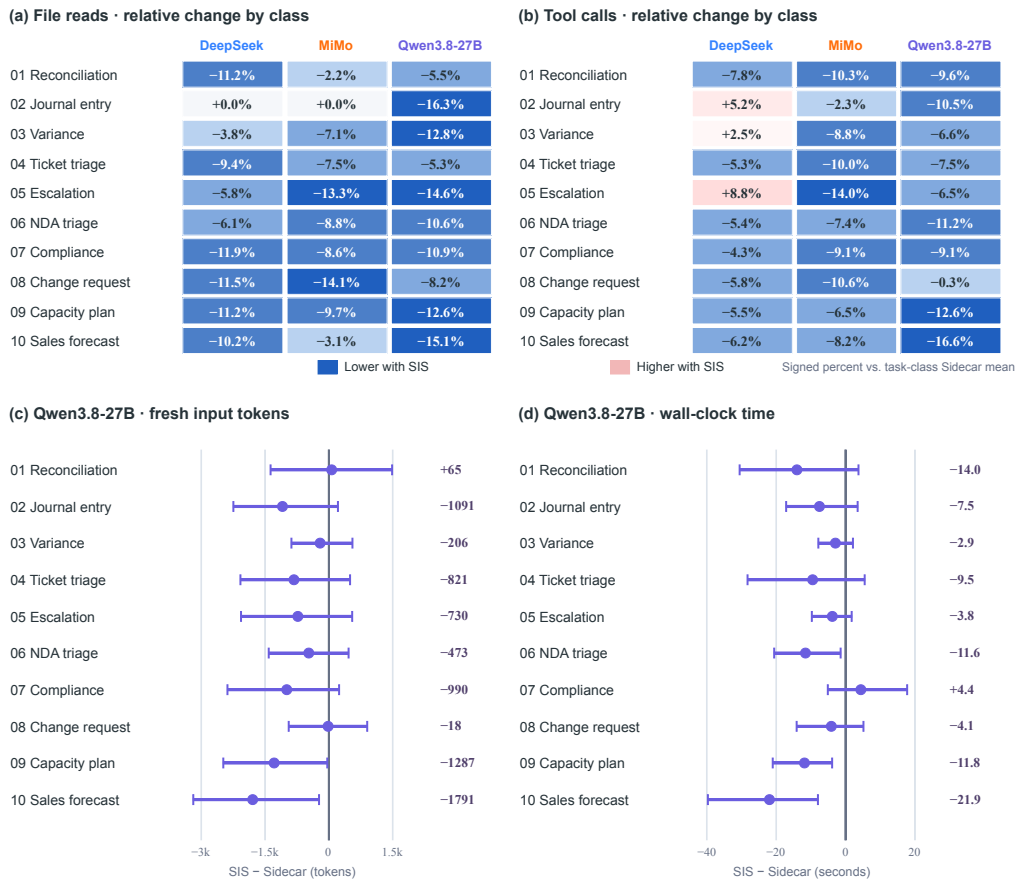


Figure 5: Governance cost across all ten task classes. (a) File-read and (b) tool-call changes for DeepSeek (DeepSeek-AI, 2026a), MiMo (LLM-Core Xiaomi, 2026), and Qwen3.8-27B (Qwen Team, 2026), relative to each class's SIDECAR mean; blue denotes lower use with SIS and pale red denotes higher use. (c) Qwen (Qwen Team, 2026) fresh-input-token and (d) wall-clock-time paired differences with 95% class-bootstrap intervals. Forest-plot values are absolute differences per run. Numeric task IDs follow Table 1.

Tables 10 and 11 give the absolute paired contrasts and category-bootstrap intervals behind the read and tool panels.

Table 10: Task-level paired SIS minus Sidecar contrasts in file reads per run, with 95% category bootstrap intervals.

Task category	DeepSeek	MiMo	Qwen3.8-27B
reconciliation	-0.47 $[-0.67, -0.31]$	-0.08 $[-0.36, +0.19]$	-0.22 $[-0.50, +0.08]$
journal entry	$+0.00$ $[-0.19, +0.19]$	$+0.00$ $[-0.31, +0.31]$	-0.69 $[-1.08, -0.33]$
variance	-0.17 $[-0.44, +0.08]$	-0.25 $[-0.58, +0.06]$	-0.58 $[-0.81, -0.36]$
ticket triage	-0.42 $[-0.69, -0.14]$	-0.28 $[-0.50, -0.06]$	-0.22 $[-0.58, +0.19]$
escalation	-0.25 $[-0.47, +0.00]$	-0.50 $[-0.78, -0.22]$	-0.61 $[-0.92, -0.31]$
NDA	-0.28 $[-0.58, +0.06]$	-0.28 $[-0.47, -0.08]$	-0.44 $[-0.64, -0.28]$
compliance	-0.56 $[-0.75, -0.36]$	-0.36 $[-0.56, -0.14]$	-0.47 $[-0.86, -0.14]$
change request	-0.53 $[-0.69, -0.36]$	-0.53 $[-0.72, -0.33]$	-0.33 $[-0.64, +0.03]$
capacity plan	-0.53 $[-0.72, -0.33]$	-0.39 $[-0.61, -0.17]$	-0.53 $[-0.81, -0.25]$
forecast	-0.47 $[-0.64, -0.28]$	-0.11 $[-0.39, +0.22]$	-0.64 $[-1.22, -0.11]$

Table 11: Task-level paired SIS minus Sidecar contrasts in tool calls per run, with 95% category bootstrap intervals.

Task category	DeepSeek	MiMo	Qwen3.8-27B
reconciliation	-0.81 $[-1.42, -0.11]$	-1.06 $[-1.89, -0.28]$	-0.97 $[-1.97, +0.00]$
journal entry	$+0.47$ $[-0.25, +1.14]$	-0.19 $[-0.78, +0.39]$	-1.06 $[-1.67, -0.39]$
variance	$+0.28$ $[-0.44, +0.97]$	-0.83 $[-1.72, -0.00]$	-0.69 $[-1.25, -0.25]$
ticket triage	-0.56 $[-1.14, +0.11]$	-0.97 $[-1.58, -0.31]$	-0.81 $[-1.69, +0.11]$
escalation	$+0.86$ $[+0.14, +1.69]$	-1.39 $[-1.92, -0.86]$	-0.64 $[-1.47, +0.25]$
NDA	-0.61 $[-1.47, +0.11]$	-0.64 $[-1.42, +0.06]$	-1.14 $[-1.47, -0.75]$
compliance	-0.47 $[-1.19, +0.22]$	-0.97 $[-1.64, -0.28]$	-0.92 $[-1.58, -0.19]$
change request	-0.58 $[-1.17, +0.03]$	-1.00 $[-1.33, -0.67]$	-0.03 $[-0.97, +0.86]$
capacity plan	-0.61 $[-1.75, +0.42]$	-0.64 $[-1.36, +0.22]$	-1.31 $[-2.19, -0.44]$
forecast	-0.67 $[-1.28, +0.06]$	-0.78 $[-1.36, -0.22]$	-1.78 $[-2.78, -0.78]$

Table 12: Qwen3.8-27B (Qwen Team, 2026) sales forecast: paired savings in tool use, fresh input, and end-to-end time.

Measure	SIS minus SIDECAR [95% CI]
Tool calls	-1.78 $[-2.78, -0.78]$
Fresh input tokens	$-1{,}791$ $[-3187, -231]$
Wall-clock seconds	-21.94 $[-39.69, -7.97]$

The sales-forecast class connects fewer tool calls with lower fresh-input use and a 21.94-second time saving per run. Tables 10 and 11 retain the full read and tool intervals for every class, including near-zero and positive estimates.

E PROCESS DECOMPOSITION

The largest process change is less repeated consultation of task context (Table 3(a)). These are cumulative counts for each placement's 360 runs, distinct from standard per-run file reads. The target taxonomy below further isolates the Qwen (Qwen Team, 2026) search and read operations.

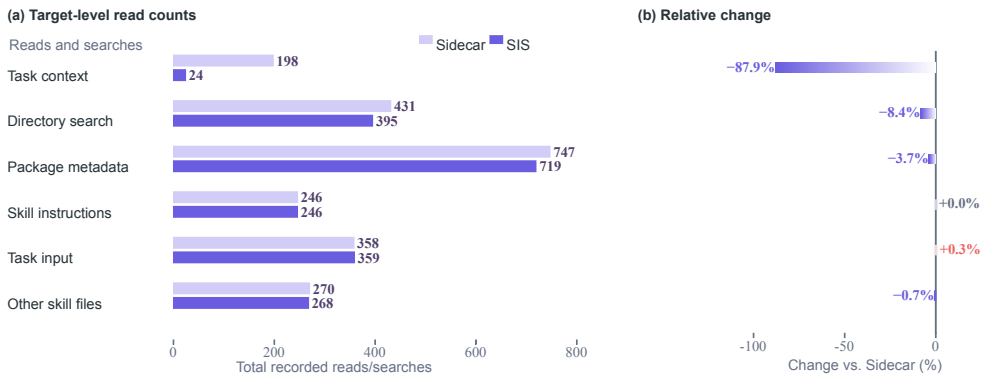


Figure 6: Qwen3.8-27B (Qwen Team, 2026) read and search targets. (a) SIDECAR and SIS counts for the six recorded target categories; (b) the corresponding relative changes. Task-context reads decline by 87.9%. Counts mix searches and file-target reads and are distinct from the standard file-read endpoint.

Across the six displayed Qwen (Qwen Team, 2026) read/search classes, task-context reads account for 174 of the 239 net recorded events avoided (72.8%). This share locates the saving in repeated context lookup; it is a share of this displayed taxonomy, not of all standard file reads or tool calls. MiMo (LLM-Core Xiaomi, 2026) independently shows 38.7% fewer directory searches and 59.9% fewer repeated read targets while business-execution calls remain near their SIDECAR level.

Table 13: MiMo (LLM-Core Xiaomi, 2026) process components per run. Changes use lifecycle SIDECAR means and are distinct from the standard resource estimator.

Process component	SIDECAR mean	SIS mean	Change
Directory searches	1.703	1.045	−38.7%
Repeated read targets	0.564	0.226	−59.9%
Distinct read targets	9.025	8.521	−5.6%
Business-execution calls	3.097	3.064	−1.1%

F HELD-OUT CHECKS AND STATISTICAL DETAIL

The held-out instances preserve the reduction in file reads across all three models. MiMo (LLM-Core Xiaomi, 2026) and Qwen (Qwen Team, 2026) also use fewer tool calls; in the Qwen (Qwen Team, 2026) setting, mean run time falls by 10.29 seconds. The instances come from the same ten task categories as the confirmation set, testing whether the resource pattern extends to reserved inputs within these workflows.

Table 14: Governance cost savings on held-out instances. Shown measures are paired SISminus-SIDECAR differences per run with 95% category-bootstrap intervals.

Model	Measure	Paired difference [95% CI]
DeepSeek (DeepSeek-AI, 2026a)	File reads	−0.392 [−0.533, −0.258]
MiMo (LLM-Core Xiaomi, 2026)	File reads	−0.225 [−0.392, −0.050]
Qwen3.8-27B (Qwen Team, 2026)	File reads	−0.308 [−0.475, −0.142]
MiMo (LLM-Core Xiaomi, 2026)	Tool calls	−0.858 [−1.267, −0.450]
Qwen3.8-27B (Qwen Team, 2026)	Tool calls	−0.625 [−1.067, −0.183]
Qwen3.8-27B (Qwen Team, 2026)	Wall-clock seconds	−10.29 [−18.24, −3.17]

For each task and placement, the analysis averages three repeats before forming a paired SISminus-SIDECAR difference. It then resamples tasks within the ten fixed categories for 10,000 draws. Main resource estimates use all 120 shared tasks; held-out checks use the 40 reserved instances. Intervals describe variation within the evaluated suite.

G BOUNDARY PROTOCOL AND TRANSFER OUTCOMES

The destination runtime operates as an authoritative policy enforcement point, subjecting all incoming capability placements to symmetric validation. Prior to issuing an execution grant, the host evaluates the package's capability, contract, and verifier digests against local policy, asserts principal identity, and ensures the requested operational scope is strictly bounded. Under delegated execution, the host additionally attests a signed delegation record that formally narrows the principal's authority. Execution integrity requires that every realized state modification corresponds to an active, destination-issued grant, and that the final output satisfies independent verification. Conversely, unauthorized attempts trigger strict pre-effect containment, intercepting the operation at the policy perimeter and guaranteeing zero business state mutation (Table 5).

Our cross-boundary protocol evaluates this governance gatekeeping across three operational regimes:

1. **Site Rebinding:** Evaluates whether an updated capability version can establish execution authority under a newly scoped destination contract.
2. **Principal Delegation:** Tests whether an agent can act under delegated credentials, where destination grants are issued for multi-step effects while maintaining decoupled audit tracking between grant issuance and atomic writes.
3. **Governance-Withdrawal Control:** Acts as a negative control by withholding destination approval, confirming that neither architecture can circumvent local gatekeeping to induce unauthorized effects.

These evaluations highlight the architectural divergence between the two paradigms. Because SIS packages verifiable contracts alongside executable capabilities, destination runtimes can establish local cryptographic provenance and issue execution grants autonomously. In contrast, SIDECAR tethers governance to its origin domain; when transferred across operational boundaries, it lacks the self-contained credentials required to satisfy destination gatekeeping, resulting in predictable pre-effect containment without out-of-band re-provisioning.