

HAI-Agent: Improving Long Horizon Software Engineering with Handoff Interventions

Junhao Wang, Evander Yang, Zenan Xu, Taiqiang Wu, Feiyuan Zhang, Michael R. Lyu
The Chinese University of Hong Kong

Abstract

Multi-agent systems for software engineering often suffer from a *delegation tax*, where the coordination overhead of sub-agents degrades performance compared to single-agent baselines. We trace this penalty to failures at the inter-agent *handoff boundary*, defined by *context loss* during task delegation and *weak verification* upon return. We propose HAI-Agent, a framework that transforms this boundary into a governed interface through three *hand-off interventions*: (1) *Delegation Review*, a pre-dispatch gate ensuring task specifications are self-contained; (2) *Retrospective Reflection*, a mandatory self-certification by the agent; and (3) *Verification Fork*, an independent validation of the return by the orchestrator. Averaged over three independent runs, HAI-Agent resolves 47.2% of SWE-bench-Pro tasks with Qwen3.5-397B-A17B (+4.0 over a matched SWE-Agent baseline) and, on the longer-horizon Commit0-lite, reaches a 61.6% pass rate with Claude-4.5-Sonnet (+11.3) and 59.8% with Qwen3.5-397B-A17B (+8.0), while consistently reducing run-to-run variance. Ablations on the full SWE-bench-Pro confirm that ungoverned delegation degrades performance below the single-agent baseline and that each intervention is necessary. Our claims are deliberately scoped to multi-agent substrates in long-horizon settings: governing the handoff removes the delegation tax where decomposition is necessary, rather than replacing single-agent test-time scaling on atomic tasks.

1 Introduction

Large Language Model (LLM) agents have evolved to tackle sophisticated, long-horizon tasks, ranging from repository-level bug fixing (Jimenez et al., 2024; OpenAI, 2024) to the autonomous generation of entire libraries (Zhao et al., 2024). As task complexity scales, the prevailing paradigm has shifted toward delegation, a strategy that partitions objectives into subtasks handled by specialized agents such as planners, editors, and verifiers. This multi-agent coordination is now foundational to both academic research (Qian et al., 2024a; Hong et al., 2024) and industrial coding assistants (OpenAI, 2026; Anthropic, 2026). By leveraging collective intelligence, these Multi-Agent Systems (MAS) facilitate the decomposition of intricate problems, integration of knowledge and iterative refinement Park et al. (2023).

Despite these theoretical advantages, multi-agent systems often fail to surpass robust single-agent baselines due to a “delegation tax” inherent in the inter-agent handoff. This systemic fragility is primarily driven by **① Context Loss**: occurs when the primary agent’s rich internal state is compressed into a lossy natural-language specification, causing sub-agents to overlook implicit constraints or engage in redundant exploration. Simultaneously, **② Weak Verification** arises from a behavioral tendency where sub-agents bypass rigorous validation and report task completion perfunctorily. Lacking mechanisms to challenge these premature claims, the primary agent blindly incorporates flawed results, allowing errors to compound silently across delegation cycles and compromising overall reliability.

Current research in multi-agent coordination generally follows three trajectories. One focuses on structured roles and workflow governance, employing explicit hierarchies to mitigate miscommunication (Hong et al., 2024; Qian et al., 2024a). A second line refines orchestration through advanced prompting and search-based task allocation (Zhang et al.,

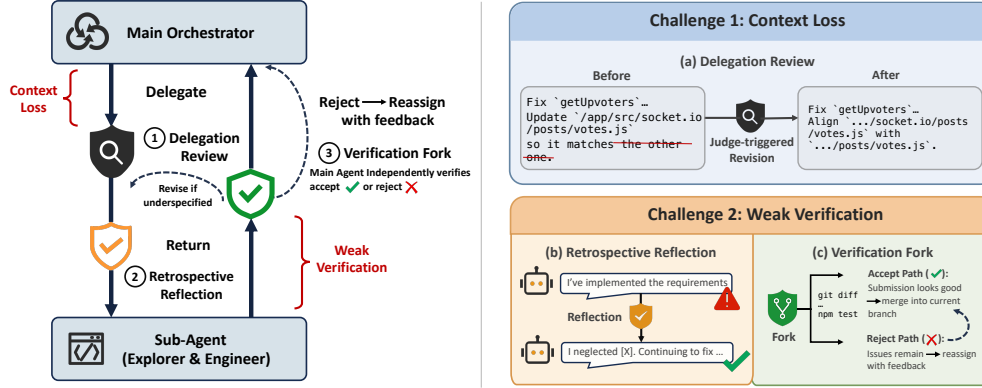


Figure 1: The HAI-Agent Architecture. The Orchestrator coordinates the lifecycle through a specialized agent tool. The handoff boundary is governed by three interventions: (1) Delegation Review (pre-delegate), (2) Retrospective Reflection (pre-return), and (3) an independent Verification Fork (post-return, Engineer-only)

2024). Concurrently, a third approach explores self-correction, using feedback loops to encourage agent introspection and output repair (Shinn et al., 2023). While these architectural and algorithmic advances offer incremental gains, they often remain domain-specific or introduce prohibitive complexity without yielding generalizable principles. Furthermore, existing failure analyses tend to be descriptive rather than prescriptive, identifying issues without providing robust mechanisms for prevention or real-time correction. Consequently, MAS remains inherently fragile when tasked with complex, open-ended objectives.

To move beyond descriptive analysis toward a prescriptive framework for robust coordination, we propose shifting the focus from agent-centric architectures to the protocols governing their interaction. We introduce three *handoff interventions* that specifically target the inter-agent boundary, addressing the root causes of the delegation tax at both the dispatch and return stages. (1) *Delegation Review*: an LLM-as-a-judge evaluates whether the task specification is self-contained (clear goal, relevant files, verifiable success criteria) before dispatch, prompting revision when the specification is underspecified. (2) *Retrospective Reflection*: before returning, the sub-agent evaluates its own output against the assigned task and produces supporting evidence (e.g., test results, diff summaries). (3) *Verification Fork*: the main agent forks a verification process from its current state to independently evaluate the return, producing a binary accept/reject decision; on rejection, the subtask is reassigned with feedback, turning silent failures into recovery opportunities.

We instantiate these interventions in HAI-Agent and evaluate it on *SWE-bench-Pro* (Deng et al., 2025) and *Commit0-lite* (Zhao et al., 2024). Averaged over three independent runs, HAI-Agent achieves a 47.2% resolve rate on *SWE-bench-Pro* with *Qwen3.5-397B-A17B* (+4.0 over the matched *SWE-Agent* baseline). On *Commit0-lite*, it achieves a 61.6% pass rate with *Claude-4.5-Sonnet* and 59.8% with *Qwen3.5-397B-A17B*, absolute gains of +11.3% and +8.0% respectively over the matched *SWE-Agent* baseline. We scope these claims deliberately: at matched test-time compute, parallel single-agent sampling remains competitive on *SWE-bench-Pro*’s atomic tasks (§4.4); the value of handoff governance lies in removing the delegation tax where multi-agent decomposition is needed, and gains are correspondingly largest on the long-horizon *Commit0-lite*.

Our contributions are as follows:

- We characterize the *Delegation Tax* in multi-agent software engineering and trace its root causes to *Context Loss* and *Weak Verification* at the inter-agent handoff boundary.

- We propose HAI-Agent, a prescriptive coordination framework that introduces three mechanical handoff interventions: *Delegation Review*, *Retrospective Reflection* and *Verification Fork*.
- We provide an empirical evaluation on two repository-scale benchmarks across diverse models, demonstrating that within a fixed multi-agent substrate, HAI-Agent removes the delegation tax, consistently outperforms matched baselines, and reduces run-to-run variance.

2 Related Work

Coding Agents. SWE-Agent (Yang et al., 2024) demonstrated that agent-computer interface design matters as much as model capability for repository-level tasks. Agentless (Xia et al., 2024) imposes a fixed localize-then-patch pipeline; OpenHands (Wang et al., 2025) provides a flexible multi-architecture platform; CodeR (Chen et al., 2024) combines retrieval with multi-turn repair. Industrial systems such as Codex (OpenAI, 2026) and Claude Code (Anthropic, 2026) rely on delegation as a core workflow. These systems focus on the interface between an agent and the codebase; we study the complementary surface: the interface *between* agents. These agents are evaluated on SWE-bench (Jimenez et al., 2024) and its extensions (SWE-bench-Pro, Multi-SWE-bench (Zan et al., 2025)), which test real-world GitHub issue resolution, and Commit0, which tests end-to-end code generation. Post-training data for coding agents includes SWE-Gym (Pan et al., 2025), SWE-Smith (Yang et al., 2025), SWE-Synth (Pham et al., 2025), SWE-Fixer (Xie et al., 2025), Lingma SWE-GPT (Ma et al., 2024), Skywork-SWE (Zeng et al., 2025), R2E-Gym (Jain et al., 2025), and SWE-RL (Wei et al., 2025). These corpora collect supervision at the trajectory level; our boundary-level verification yields finer-grained per-subtask labels as a byproduct.

Multi-Agent Coordination. Early LLM-based multi-agent systems organize collaboration through predefined roles: ChatDev (Qian et al., 2024a) and MetaGPT (Hong et al., 2024) simulate software teams with sequential role pipelines, while MapCoder (Islam et al., 2024) and CodeAgent (Tang et al., 2024) decompose tasks across specialized agents. Scaling analyses reveal the cost of implicit handoffs: coordination overhead grows superlinearly with agent count (Qian et al., 2024b), and naively adding agents can degrade performance (Li et al., 2024b). In dialogue-driven systems (Wu et al., 2024), delegation emerges through conversation without explicit control. Planning-oriented analyses report severe task overlap and inconsistent action sequences when delegation is ungoverned (Li et al., 2024a). Cemri et al. (2025) develop the MAST failure taxonomy from 1600+ annotated traces, identifying task verification failures as roughly one-fifth of all breakdowns. Recent work has begun to treat inter-agent transfer as a first-class design target: AgentAsk (Lin et al., 2025) introduces clarification at delegation boundaries; Barrak (2025) studies accountable handoffs with blame attribution; LDP (Prakash, 2026) elevates delegate identity and provenance to protocol primitives. Multi-Agent Verification (Lifshitz et al., 2025) scales test-time compute by composing aspect verifiers, and MALT (Motwani et al., 2024) trains verifier agents jointly through multi-agent reinforcement learning. In the LLM reasoning setting, debate (Du et al., 2023; Liang et al., 2023), reflexion (Shinn et al., 2023), and process reward models (Lightman et al., 2024) provide related verification mechanisms. Our work addresses a specific question: not how to organize agents into roles or train better verifiers, but how to enforce verification at the boundary where control changes hands through targeted handoff interventions. AgentAuditor (Hua et al., 2025) performs post-hoc evidence-based auditing of multi-agent traces; AgentDropoutV2 (Wang et al., 2026) intercepts erroneous outputs at runtime via a rectify-or-reject pipeline. In our setting, verification is not a separate post-hoc step but an enforced check at the handoff boundary.

3 HAI-Agent: Governing the Inter-Agent Handoff

3.1 System Architecture

Scaffolding. HAI-Agent adopts an orchestrator-worker multi-agent architecture. The system coordinates the high-level reasoning trajectory and global state by managing parallel Git worktrees. Specialized agents are spawned to execute subtasks within designated worktree scopes, ensuring that concurrent implementations remain logically and physically isolated.

Tools. We leverage the models’ native tool-calling capabilities to facilitate interaction with both the repository and our multi-agent harness. We distinguish between two functional interface categories:

- *Agent-Computer Interface (ACI)*: Provides atomic primitives for repository interaction, including `Glob` and `Grep` for search, `Read` for inspection, `Write` and `Replace` for modification, and a general-purpose Bash shell. To support multi-worktree management, the ACI includes `list_worktree` to view existing scopes and `create_worktree` to spawn new worktrees from `HEAD` or specific branches.
- *Agent-Agent Interface (AAI)*: The AAI governs the delegation lifecycle and is the primary object for implementing our handoff interventions. It consists of two coordination primitives:
 - `agent`: Exclusive to the Orchestrator for dispatching subtasks. It requires a *worker type*, a *task* objective, and a *worktree* identifier (required for Engineers). If the specified worktree does not exist, the system automatically initializes it. Calling this tool triggers the *Delegation Review*.
 - `finish`: A universal completion tool used by all agents to terminate their current trajectory. When invoked by an *Agent*, it initiates *Retrospective Reflection*. Crucially, if the agent is an *Engineer*, the return also triggers our independent *Verification Fork*. When invoked by the *Orchestrator*, it constitutes the final submission of the global solution.

Agent Roles. We define three distinct roles with asymmetric access to the ACI and AAI to enforce a strict separation of concerns:

- *Orchestrator*: The root agent responsible for global planning, worktree lifecycle management (`create_worktree`), and task delegation (`agent`). It maintains the high-level context and integrates agent returns into the final solution.
- *Explorer*: Read-only agent restricted to the read-only subset of the ACI (`Glob`, `Grep`, and `Read`). They are optimized for codebase comprehension and localization without introducing side effects.
- *Engineer*: Agent with full ACI access, including `Write`, `Replace`, and `Bash`. They operate on a dedicated worktree assigned by the Orchestrator, ensuring that implementations and local verifications remain isolated.

Crucially, the `agent`, `list_worktree`, and `create_worktree` tools are excluded from the tool definitions exposed to Explorer and Engineer agents—rather than merely discouraged by prompt—preventing unauthorized invocation during their local execution cycles. This ensures that delegation and structural worktree management remain privileged Orchestrator functions, preventing uncontrolled recursion and maintaining a centralized coordination structure.

3.2 Handoff Interventions

To mitigate the *Delegation Tax*, HAI-Agent transforms the inter-agent handoff from a simple message exchange into a series of governed synchronization points. These interventions are structurally integrated into the AAI tool-calling protocol, ensuring they cannot be

bypassed by the models. The full prompt texts for all roles and interventions are provided in Appendix C.

Delegation Review: Dispatch-side Quality Gate. Context loss is inherent in delegation, as the *Orchestrator* must compress its internal mental model into a natural-language *task description*. To ensure this compression is not lossy, the agent tool call is intercepted by a *Delegation Reviewer* (an LLM-as-a-judge). The reviewer evaluates the dispatch against a rigorous three-point checklist: (i) *Atomicity*: Does the task target a single, well-defined objective? (ii) *Self-containment*: Are all relevant file paths, identified dependencies, and previously discarded hypotheses explicitly stated? (iii) *Verifiability*: Are there clear, execution-based criteria to define success? If the reviewer identifies ambiguities or missing context, the agent call is rejected, and the *Orchestrator* is prompted to revise the task description. This forces the externalization of implicit reasoning before the sub-agent trajectory begins.

Retrospective Reflection: Worker-Side Self-Relection. To combat the behavioral tendency toward perfunctory verification, the `finish` tool triggers a mandatory *Retrospective Reflection*. Our agent harness automatically extracts the final `git diff` and reinjects the original task description into the sub-agent’s context, compelling a systematic self-audit. Before exiting, the agent must reconcile its modifications with the initial requirements. For an *Engineer*, this artifact must include logs from a newly passing test suite; for an *Explorer*, it must furnish a trace of evidence supporting its localization claim. This protocol transforms the exit signal from a mere assertion of completion into a documented certification, anchoring the agent’s final output to its original objective.

Verification Fork: Orchestrator-side Validation. The most critical intervention is the *Verification Fork*, which is triggered specifically upon an *Engineer*’s return to ensure implementation correctness. Upon receiving a `finish` tool call, the system forks an independent verification process—a separate LLM call that inherits the *Orchestrator*’s full global context but remains uncorrupted by the sub-agent’s internal reasoning chain. The fork consumes the complete evidence package (the patch *diff*, test logs, execution outputs, and the agent’s reflection) and produces a binary decision $v \in \{0, 1\}$ to either accept or reject the contribution. The decision is scored strictly against the assigned subtask, interpreted in the global task context, so a valid partial solution is not rejected for leaving the overall issue unsolved; the fork also cross-checks the reflection’s claims against the actual diff and logs, so a fabricated self-certification does not survive the boundary. On rejection ($v = 0$), the *Engineer*’s worktree changes are discarded and the subtask is re-delegated to a fresh agent spawned from the *Orchestrator*’s current state, with a feedback log containing the fork’s critique appended; merges into the main workspace occur only on acceptance ($v = 1$), at which point temporary evidence logs are purged. This mechanism exploits the fundamental asymmetry of software engineering—where verification is computationally cheaper and more objective than generation—to convert silent *task drift* into explicit recovery opportunities. By limiting this heavyweight check to *Engineers*, we prioritize computational resources on implementational boundaries where the risk of silent regression is highest.

4 Experiments

4.1 Experimental Setup

Benchmarks. We evaluate HAI-Agent on two long horizon software engineering benchmarks that stress different dimensions of multi-agent coordination:

1. *SWE-bench-Pro* (Deng et al., 2025): A high-fidelity benchmark of real-world GitHub issues requiring professional-grade, repository-level reasoning. Unlike the original SWE-bench, the “Pro” version focuses on non-trivial tasks where the solution often spans multiple files and necessitates a deep understanding of cross-module dependencies. Success is strictly evaluated by a patch’s ability to pass a hidden, human-verified test suite.
2. *Commit0-lite* (Zhao et al., 2024): A challenging benchmark that requires agents to implement entire Python libraries from scratch, given only a repository skeleton and a

<i>SWE-bench-Pro</i>						
Model	<i>Multi-Agent</i>			<i>HAI-Agent</i>		
	Score	Runtime	Cost	Score	Runtime	Cost
Claude-4.5-Sonnet	38.8	1250.6	1.4	45.9	1380.2	1.5
Qwen3.5-397B-A17B	39.6	1340.2	0.4	46.8	1420.1	0.5

<i>Commit0-Lite</i>						
Model	<i>Multi-Agent</i>			<i>HAI-Agent</i>		
	Score	Runtime	Cost	Score	Runtime	Cost
Claude-4.5-Sonnet	56.5	1780.4	4.8	62.3	1890.5	5.0
Qwen3.5-397B-A17B	55.1	1890.5	1.6	64.0	1950.2	1.8

Table 1: Main results on SWE-bench-Pro and Commit0-Lite. Score represents Resolved (%) for SWE-bench-Pro and Pass Rate (%) for Commit0-Lite. Runtime is the average task time in seconds; Cost is averaged per task.

comprehensive suite of unit tests. This is fundamentally an integration problem: a task succeeds only if the final implementation passes the entire test suite. In this setting, the *Orchestrator* must perform detailed dependency analysis—mapping tests to specific files and determining the optimal implementation order—before partitioning the codebase into file-level or function-level tasks for individual *Agents*. Commit0-lite is particularly suited for evaluating our handoff interventions because the high density of internal dependencies and the requirement for full test pass-rates penalize any information decay or weak verification during delegation.

Baselines. To rigorously evaluate the impact of our handoff interventions, we compare HAI-Agent against a primary *Naive Multi-agent* baseline built on the same underlying substrate. This baseline is a matched orchestrator-worker system that utilizes the same AAI hierarchy and toolset as HAI-Agent, but operates without our proposed interventions (Delegation Review, Reflection, and Verification Fork). By holding the agent substrate fixed—including the prompting structure, memory management, and execution policies—this comparison isolates the specific incremental value of *handoff governance*.

This design ensures that our evaluation does not conflate framework-level differences (e.g., tool schemas or environment handling) with coordination-level improvements; comparisons against monolithic single-agent configurations appear in the ablation analysis (Section 4.3).

Configuration. No artificial iteration or turn caps are imposed on any scaffold—neither on the *Orchestrator* nor on any spawned *Agent*; every trajectory terminates only when the agent itself emits a finish/submit action. Sampling temperature is set to 1.0 for both model families. To maximize the models’ capacity for logical deduction and self-certification, we enable thinking for all models across all experimental runs (enable_thinking for the Qwen series); for *Claude-4.5-Sonnet*, we use the xhigh reasoning-effort setting. Unless noted otherwise, we report the mean over three independent runs, with the standard deviation in parentheses.

Metrics. We report the *Resolved* (%) rate for SWE-bench-Pro, which requires passing both fail-to-pass (F2P) and pass-to-pass (P2P) test suites. For Commit0-lite, we report the *Pass Rate* (%) of the synthesized repositories against the full unit test suite.

4.2 Main Results

We first evaluate HAI-Agent against the *Multi-agent* baseline to isolate the impact of handoff governance. Table 1 presents the performance, efficiency, and cost metrics for our flagship models.

Compared with the naive *Multi-agent* baseline that lacks architectural governance, HAI-Agent delivers a substantial performance leap with marginal increases in latency and computational cost, transforming the inter-agent boundary into a high-yield quality-control gate.

High-Efficiency Reliability. On *SWE-bench-Pro*, HAI-Agent improves the resolve rate of *Qwen3.5-397B-A17B* by 7.2% absolute points, while the runtime overhead is restricted to a mere 6% increase (1340.2s vs. 1420.1s) and the per-task cost remains virtually identical. A similar trend is observed with *Claude-4.5-Sonnet*, where a 7.1% performance gain is achieved with less than a 10% increase in both runtime and financial cost. These results suggest that the *Delegation Tax* is not a necessary consequence of multi-agent architectures, but rather a symptom of ungoverned handoffs that can be neutralized through targeted interventions.

Maximized Return on Coordination. In the *Commit0-Lite* implementation task, the return on investment for handoff governance is even more pronounced. For *Qwen3.5-397B-A17B*, HAI-Agent achieves an 8.9% absolute improvement in pass rate (55.1% $\rightarrow$ 64.0%) with less than 3% additional runtime. Similarly, *Claude-4.5-Sonnet* sees a 5.8% improvement with only a 6% increase in latency and a marginal 4% increase in cost. By investing a small amount of additional inference in *Delegation Review* and the *Verification Fork*, HAI-Agent ensures that sub-agent efforts are anchored to the global objective, preventing costly *task drift* without the prohibitive overhead typically associated with redundant multi-agent verification.

Model	Agent	Score
<i>SWE-bench-Pro</i> (Resolved %)		
Qwen3-Coder-30B	SWE-Agent	22.2 (2.4)
	HAI-Agent	24.8 (1.1)
Qwen3.5-35B-A3B	SWE-Agent	28.9 (2.1)
	HAI-Agent	32.0 (1.1)
Claude-4.5-Sonnet	Claude Code	39.1 (2.5)
	SWE-Agent	43.6 [†]
	HAI-Agent	46.1[†]
Qwen3.5-397B-A17B	SWE-Agent	43.2 (1.8)
	HAI-Agent	47.2 (1.0)
<i>Commit0-lite</i> (Pass Rate %)		
Claude-4.5-Sonnet	SWE-Agent	50.3 (2.8)
	HAI-Agent	61.6 (1.3)
Qwen3.5-397B-A17B	SWE-Agent	51.8 (2.4)
	HAI-Agent	59.8 (1.1)

Table 2: Performance comparison against specialized scaffolds across diverse model scales, reported as mean (std) over three independent runs. †: single run.

Comparing with SWE-Agent. We compare HAI-Agent against *SWE-Agent* (Yang et al., 2024)—a widely adopted repository-level scaffold—on both benchmarks to evaluate the benefits of handoff governance over purely instructional delegation. As shown in Table 2, HAI-Agent consistently outperforms *SWE-Agent* across model families and scales. On *SWE-bench-Pro*, HAI-Agent achieves a +4.0% absolute improvement for *Qwen3.5-397B-A17B* and a +2.5% gain for *Claude-4.5-Sonnet*. This suggests that while *SWE-Agent* provides a robust interface for repository interaction (ACI), it lacks the mechanical governance at the inter-agent boundary required to prevent the *Delegation Tax*.

The performance gap is even more pronounced on *Commit0-Lite*, which requires dense, end-to-end implementation across multiple files. For *Claude-4.5-Sonnet*, HAI-Agent reaches a 61.6% pass rate, a +11.3% absolute gain over the 50.3% attained by *SWE-Agent*; for *Qwen3.5-397B-A17B*, HAI-Agent yields a +8.0% improvement. The interventions also stabilize

execution: HAI-Agent’s standard deviation ranges from 1.0 to 1.3 across all three-run configurations, versus 1.8 to 2.8 for the *SWE-Agent* and *Claude Code* baselines, indicating that governed handoffs yield more predictable execution paths. These results demonstrate that our handoff interventions—specifically the *Verification Fork* and *Retrospective Reflection*—are critical for maintaining structural and logical consistency in complex integration tasks.

4.3 Ablation of Handoff Interventions

Table 3 ablates the interventions of HAI-Agent on the *full* SWE-bench-Pro with *Qwen3.5-397B-A17B*, in two ways: removing each intervention entirely, and downgrading each intervention’s backbone to the smaller *Qwen3.5-35B-A3B* while the rest of the system continues to use *Qwen3.5-397B-A17B*.

Intervention ablated	Resolved (%)	
	Removed	Downgraded [‡]
Retrospective reflection	41.5 (1.5)	43.8 (1.4)
Verification fork	43.4 (1.3)	45.0 (1.2)
Delegation review	45.2 (1.5)	46.0 (1.1)
Single-Agent	41.6 (2.7)	
Multi-Agent (Naive) [†]	39.9 (3.0)	
HAI-Agent (Full)	47.2 (1.0)	

Table 3: Ablation on the full SWE-bench-Pro using Qwen3.5-397B-A17B, mean (std) over three runs. [†]: the orchestrator-worker system without handoff interventions. [‡]: the intervention’s backbone replaced with Qwen3.5-35B-A3B while the rest of the system uses Qwen3.5-397B-A17B.

Naive delegation hurts performance. A primary insight from Table 3 is that multi-agent coordination is not beneficial by default. Transitioning from a *Single-Agent* (41.6%) to a *Naive Multi-Agent* setup (39.9%) results in a performance regression below the single-agent baseline, with the highest run-to-run variance of any configuration (std 3.0). This empirical validation of the *Delegation Tax* suggests that the introduction of specialized agents—when left ungoverned—amplifies coordination overhead and context loss without providing a corresponding increase in reliability. Simply “adding more agents” to a software engineering task is counterproductive unless the handoff boundaries are strictly managed.

Handoff interventions drive the recovery and gain. HAI-Agent successfully reverses this tax, recovering the lost performance and driving the resolve rate to 47.2%, a +7.3% absolute gain over the naive multi-agent baseline. Among the three interventions, *Retrospective Reflection* is the most impactful: its removal causes the resolve rate to collapse to 41.5%, equivalent to the single-agent baseline. This highlights that enforcing self-certification at the worker side is the primary defense against *task drift*. The *Verification Fork* and *Delegation Review* provide substantial additive gains (+3.8% and +2.0% respectively over their removed configurations), with the fork being particularly effective in catching implementation-level regressions that bypass an agent’s internal reflection. Every partial configuration also exhibits higher variance than the full system.

The protocols matter beyond raw verifier capability. Downgrading any single intervention’s backbone to *Qwen3.5-35B-A3B* recovers a substantial share of the corresponding removed-intervention gap: 43.8% vs. 41.5% for reflection, 45.0% vs. 43.4% for the fork, and 46.0% vs. 45.2% for the review. A structurally enforced check executed by a weaker model still outperforms no check at all, indicating that the handoff protocols themselves provide systemic guardrails rather than merely proxying for verifier strength.

4.4 Efficiency and Compute-Matched Baselines

Overhead is benchmark-dependent. Appendix B reports per-run cost (LLM calls, tool calls, tokens) for HAI-Agent, *SWE-Agent*, and *Claude Code* under identical model configurations. On *SWE-bench-Pro*, HAI-Agent consumes roughly $5\times$ the tokens of a single *SWE-Agent* trajectory (13.7M vs. 2.74M): single-context baselines terminate early because one context window suffices for most atomic fixes, while HAI-Agent keeps spawning the sub-agent contexts that the multi-agent regime is built for. On the long-horizon *Commit0-lite*, the overhead collapses to roughly $1.4\times$ (356.5 vs. 254.8 LLM calls), because the baseline itself must execute a long trajectory. The relative cost of handoff governance is thus largely a function of task complexity and exploration depth rather than a fixed scaffold overhead, and it is smallest precisely in the long-horizon regime that multi-agent decomposition targets.

Compute-matched test-time scaling. Since HAI-Agent spends $\sim 5\times$ compute on *SWE-bench-Pro*, we compare against spending the same budget on single-agent test-time scaling: *SWE-Agent* with Best-of-5 sampling, using an independent LLM verifier to rank the parallel candidates without access to ground-truth tests (Table 4).

Configuration	Compute (rel.)	Resolved (%)
SWE-Agent	$1.0\times$	43.2 (1.8)
Multi-Agent (Naive)	$\sim 5.0\times$	39.9 (3.0)
SWE-Agent + Best-of-5	$\sim 5.0\times$	49.6
HAI-Agent	$\sim 5.0\times$	47.2 (1.0)

Table 4: Compute-matched comparison on the full *SWE-bench-Pro* with Qwen3.5-397B-A17B. Best-of-5 ranks five independent *SWE-Agent* trajectories with an LLM verifier (no ground-truth tests).

At matched compute, Best-of-5 *SWE-Agent* edges HAI-Agent by 2.4 points on *SWE-bench-Pro*. This is expected from the benchmark’s structure: its tasks are largely atomic, single-file fixes that fit within one context window, a regime where parallel sampling over single-agent contexts excels and coordination overhead is not repaid. We therefore make no claim that HAI-Agent outperforms single-agent test-time scaling on atomic tasks, nor that it is a generally superior scaffold. The comparison that isolates our contribution is within the multi-agent substrate: at the same $\sim 5\times$ budget, naive delegation collapses to 39.9% while HAI-Agent reaches 47.2% with the lowest variance of any configuration, converting the delegation tax into a +7.3-point gain. On *Commit0-lite*, where tasks exceed a single context window and parallel single-context attempts cannot scale, the compute overhead drops to $\sim 1.4\times$ and HAI-Agent outperforms *SWE-Agent* by up to +11.3 (Table 2). In short, handoff governance removes the delegation penalty wherever multi-agent structure is being paid for anyway (long-horizon scope, parallelism, specialization); whether that structure is worth its coordination cost on atomic tasks is a separate question, and there the answer is no by construction.

A detailed behavioral analysis of HAI-Agent’s delegation patterns—per-role call statistics, role bias across benchmarks, and trajectory depth—is provided in Appendix A.

5 Discussion and Limitations

5.1 Handoff-Level Failure Accounting

Each intervention triggers at a substantial rate. Averaged across three *SWE-bench-Pro* runs, *Delegation Review* flags 28% of dispatches as underspecified, typically for missing target paths, vague success criteria, or lost upstream context. *Retrospective Reflection* surfaces self-detected issues on 34% of return attempts, most often untested code, compilation breakages, or partial feature implementation. The *Verification Fork* rejects 31% of Engineer returns ($v = 0$), catching local edits that violate global constraints (e.g., calls to deprecated helpers) and *task drift*. Rejection is recoverable: 75% of rejections are resolved by direct

retries (52% on the first retry, 23% on the second), and the remaining 25% recover after the Orchestrator revises the task specification and re-delegates—in our runs, every such re-delegated subtask subsequently passed the fork.

The three layers are complementary, not redundant. Among 200 rejected returns on SWE-bench-Pro, 24% are caught *only* by the *Verification Fork*, 21% only by *Delegation Review*, and 17% only by *Retrospective Reflection*; overlaps account for the remainder (reflection and fork 19%, review plus post-revision rejection 13%, all three 6%). The 24% caught only by the Orchestrator-side fork—including reflections whose claims do not match the actual diff or logs—underscores why worker-side self-certification alone is insufficient.

Residual failures concentrate in planning, not the boundary. Decomposing 150 failed HAI-Agent runs by root cause: 52% stem from decomposition and integration (wrong subtask planning 28%, missed cross-file dependencies 24%), 19% from implementation errors within correctly scoped subtasks, 22% from external or data issues (environment and test setup 13%, ambiguous task specifications 9%), and 7% from other noise. The primary residual bottleneck is thus Orchestrator-level planning quality, which handoff interventions regulate only indirectly. This scopes our contribution: HAI-Agent is a boundary-governance mechanism, not a general fix for multi-agent planning failures, and planning-layer verification is a natural direction for future work.

5.2 Limitations

Scope of claims. Our results support a deliberately narrow claim: within a fixed multi-agent substrate on long-horizon, decomposition-heavy tasks, governing the handoff removes the delegation tax. They do not establish superiority over single-agent test-time scaling on atomic tasks (§4.4).

Computational overhead. HAI-Agent consumes significantly more computational resources than a monolithic single-agent trajectory: roughly $5\times$ the tokens of a single SWE-Agent trajectory on SWE-bench-Pro and roughly $1.4\times$ on Commit0-lite (Appendix B); the overhead is a function of task horizon rather than a fixed cost. However, HAI-Agent represents a structured form of *test-time scaling*. Unlike naive multi-agent systems where additional compute leads to performance degradation (39.9%, below the single-agent baseline), our governed handoffs ensure that increased inference-time compute translates into higher reliability (47.2%) and lower variance at the same budget (§4.4). The *Verification Fork* and re-delegation loops thus provide a mechanical way to trade additional computation for functional correctness.

Potential training usage. A significant byproduct of our architecture is the generation of high-fidelity training signals. The binary outcomes $v \in \{0, 1\}$ produced by the *Verification Fork* at each handoff boundary provide a natural filtering mechanism for synthetic data. Specifically, implementation traces where $v = 1$ can be extracted as verified demonstrations even from trajectories that fail to resolve the global task. This offers a promising path for *amortizing inference costs* into long-term model improvement by fine-tuning on locally-verified subtasks.

6 Conclusion

We study handoffs in multi-agent systems and identify two inherent challenges: context loss during dispatch and weak verification upon return. We introduce three handoff interventions—delegation review, retrospective reflection, and a verification fork—and instantiate them in HAI-Agent. The resulting gains on SWE-bench-Pro and Commit0-lite, obtained within a fixed multi-agent substrate and largest in long-horizon settings, are primarily attributable to enforced verification. The broader lesson: where decomposition is necessary, delegation becomes reliable not by improving what agents generate, but by enforcing what gets verified before it is accepted.

Ethics Statement

Our goal is to improve the reliability of software engineering assistants for legitimate development and maintenance tasks. The same capabilities could also be misused for malicious code modification, vulnerability exploration, or automating low-quality patch generation at scale. To reduce those risks, we evaluate only in controlled benchmark settings over public repositories, and we do not advocate autonomous deployment without human code review. The proposed self-verification mechanism should be understood as a reliability aid rather than a safety guarantee: verified subtasks can still compose into an incorrect overall fix, and benchmark success does not imply production readiness. We therefore recommend that any practical deployment of such systems include repository access controls, audit logs, and mandatory human approval before patches are merged.

References

- Anthropic. Claude code. <https://www.anthropic.com/claude-code/>, 2026. [Accessed 24-03-2026].
- Amine Barrak. Traceability and accountability in role-specialized multi-agent llm pipelines, 2025. URL <https://arxiv.org/abs/2510.07614>.
- Mert Cemri, Melissa Z Pan, Shuyi Yang, Lakshya A Agrawal, Bhavya Chopra, Rishabh Tiwari, Kurt Keutzer, Aditya Parameswaran, Dan Klein, Kannan Ramchandran, et al. Why do multi-agent llm systems fail? *arXiv preprint arXiv:2503.13657*, 2025.
- Dong Chen, Shaoxin Lin, Muhan Zeng, Daoguang Zan, Jian-Gang Wang, Anton Cheshkov, Jun Sun, Hao Yu, Guoliang Chen, Xin Tian, et al. Coder: Issue resolving with multi-agent and task graphs. *arXiv preprint arXiv:2406.01304*, 2024.
- Xiang Deng, Jeff Da, Edwin Pan, Yannis Yiming He, Charles Ide, Kanak Garg, Niklas Lauffer, Andrew Park, Nitin Pasari, Chetan Rane, Karmini Sampath, Maya Krishnan, Srivatsa Kundurthy, Sean Hendryx, Zifan Wang, Vijay Bharadwaj, Jeff Holm, Raja Aluri, Chen Bo Calvin Zhang, Noah Jacobson, Bing Liu, and Brad Kenstler. Swe-bench pro: Can ai agents solve long-horizon software engineering tasks?, 2025. URL <https://arxiv.org/abs/2509.16941>.
- Yilun Du, Shuang Li, Antonio Torralba, Joshua B Tenenbaum, and Igor Mordatch. Improving factuality and reasoning in language models through multiagent debate. *arXiv preprint arXiv:2305.14325*, 2023.
- Sirui Hong, Mingchen Zhuge, Jonathan Chen, Xiwu Zheng, Yuheng Cheng, Ceyao Zhang, Jinlin Wang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, Liyang Zhou, Chenyu Ran, Lingfeng Xiao, Chenglin Wu, and Jürgen Schmidhuber. Metagpt: Meta programming for a multi-agent collaborative framework, 2024. URL <https://arxiv.org/abs/2308.00352>.
- Wenyue Hua, Yibing Zhang, Kexun Chen, Yifan Li, Yejin Wu, and Yu Wang. Auditing multi-agent llm reasoning trees outperforms majority voting. *arXiv preprint arXiv:2602.09341*, 2025.
- Md. Ashraful Islam, Mohammed Eunus Ali, and Md Rizwan Parvez. Mapcoder: Multi-agent code generation for competitive problem solving. *arXiv preprint arXiv:2405.11403*, 2024.
- Naman Jain, Jaskirat Singh, Manish Shetty, Liang Zheng, Koushik Sen, and Ion Stoica. R2e-gym: Procedural environments and hybrid verifiers for scaling open-weights swe agents, 2025. URL <https://arxiv.org/abs/2504.07164>.
- Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. Swe-bench: Can language models resolve real-world github issues?, 2024. URL <https://arxiv.org/abs/2310.06770>.

- Ao Li, Yuexiang Xie, Songze Li, Fugee Tsung, Bolin Ding, and Yaliang Li. Agent-oriented planning in multi-agent systems. *arXiv preprint arXiv:2410.02189*, 2024a.
- Junyou Li, Qin Zhang, Yangbin Yu, Qiang Fu, and Deheng Ye. More agents is all you need. *arXiv preprint arXiv:2402.05120*, 2024b.
- Tian Liang, Zhiwei He, Wenxiang Jiao, Xing Wang, Yan Wang, Rui Wang, Yujiu Yang, Zhaopeng Tu, and Shuming Shi. Encouraging divergent thinking in large language models through multi-agent debate. *arXiv preprint arXiv:2305.19118*, 2023.
- Shalev Lifshitz, Sheila A McIlraith, and Yilun Du. Multi-agent verification: Scaling test-time compute with multiple verifiers. *arXiv preprint arXiv:2502.20379*, 2025.
- Hunter Lightman, Vineet Kosaraju, Yura Burda, Harri Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let’s verify step by step, 2024. URL <https://arxiv.org/abs/2305.20050>.
- Bohan Lin, Kuo Yang, Yingchuan Lai, Yudong Zhang, Chen Zhang, Guibin Zhang, Xinlei Yu, Miao Yu, Xu Wang, and Yang Wang. Agentask: Multi-agent systems need to ask, 2025. URL <https://arxiv.org/abs/2510.07593>.
- Yingwei Ma, Rongyu Cao, Yongchang Cao, Yue Zhang, Jue Chen, Yibo Liu, Yuchen Liu, Binhua Li, Fei Huang, and Yongbin Li. Lingma swe-gpt: An open development-process-centric language model for automated software improvement, 2024. URL <https://arxiv.org/abs/2411.00622>.
- Sumeet Ramesh Motwani, Chandler Smith, Rocktim Jyoti Das, Rafael Rafailov, Ivan Laptev, Philip H S Torr, Fabio Pizzati, Ronald Clark, and Christian Schroeder de Witt. MALT: Improving reasoning with multi-agent LLM training. *arXiv preprint arXiv:2412.01928*, 2024.
- OpenAI. Introducing SWE-Bench Verified. <https://openai.com/index/introducing-swe-bench-verified/>, 2024. Accessed: 07 Jun 2024.
- OpenAI. Introducing the codex app. <https://openai.com/index/introducing-the-codex-app/>, 2026. [Accessed 24-03-2026].
- Jiayi Pan, Xingyao Wang, Graham Neubig, Navdeep Jaitly, Heng Ji, Alane Suhr, and Yizhe Zhang. Training software engineering agents and verifiers with swe-gym, 2025. URL <https://arxiv.org/abs/2412.21139>.
- Joon Sung Park, Joseph C. O’Brien, Carrie J. Cai, Meredith Ringel Morris, Percy Liang, and Michael S. Bernstein. Generative agents: Interactive simulacra of human behavior, 2023. URL <https://arxiv.org/abs/2304.03442>.
- Minh V. T. Pham, Huy N. Phan, Hoang N. Phan, Cuong Le Chi, Tien N. Nguyen, and Nghi D. Q. Bui. Swe-synth: Synthesizing verifiable bug-fix data to enable large language models in resolving real-world bugs, 2025. URL <https://arxiv.org/abs/2504.14757>.
- Sunil Prakash. Ldp: An identity-aware protocol for multi-agent llm systems, 2026. URL <https://arxiv.org/abs/2603.08852>.
- Chen Qian, Wei Liu, Hongzhang Liu, Nuo Chen, Yufan Dang, Jiahao Li, Cheng Yang, Weize Chen, Yusheng Su, Xin Cong, Juyuan Xu, Dahai Li, Zhiyuan Liu, and Maosong Sun. Chatdev: Communicative agents for software development, 2024a. URL <https://arxiv.org/abs/2307.07924>.
- Chen Qian, Zihao Xie, Yifei Wang, Wei Liu, Kunlun Zhu, Hanchen Xia, Yufan Dang, Zhuoyun Du, Weize Chen, Cheng Yang, et al. Scaling large language model-based multi-agent collaboration. *arXiv preprint arXiv:2406.07155*, 2024b.
- Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. *Advances in Neural Information Processing Systems*, 2023.

- Kechi Tang, Ge Li, Jia Liu, and Zhi Jin. Codeagent: Enhancing code generation with tool-integrated agent systems for real-world repo-level coding challenges. *arXiv preprint arXiv:2401.07339*, 2024.
- 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, Robert Brennan, Hao Peng, Heng Ji, and Graham Neubig. Openhands: An open platform for ai software developers as generalist agents, 2025. URL <https://arxiv.org/abs/2407.16741>.
- Yutong Wang, Siyuan Xiong, Xuebo Liu, Wenkang Zhou, Liang Ding, Miao Zhang, and Min Zhang. Agentdropoutv2: Optimizing information flow in multi-agent systems via test-time rectify-or-reject pruning. *arXiv preprint arXiv:2602.23258*, 2026.
- Yuxiang Wei, Olivier Duchenne, Jade Copet, Quentin Carbonneaux, Lingming Zhang, Daniel Fried, Gabriel Synnaeve, Rishabh Singh, and Sida I. Wang. Swe-rl: Advancing llm reasoning via reinforcement learning on open software evolution, 2025. URL <https://arxiv.org/abs/2502.18449>.
- Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, et al. Autogen: Enabling next-gen llm applications via multi-agent conversations. In *First conference on language modeling*, 2024.
- Chunqiu Steven Xia, Yinlin Deng, Soren Dunn, and Lingming Zhang. Agentless: Demystifying llm-based software engineering agents, 2024. URL <https://arxiv.org/abs/2407.01489>.
- Chengxing Xie, Bowen Li, Chang Gao, He Du, Wai Lam, Difan Zou, and Kai Chen. Swe-fixer: Training open-source llms for effective and efficient github issue resolution, 2025. URL <https://arxiv.org/abs/2501.05040>.
- 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, 2024. URL <https://arxiv.org/abs/2405.15793>.
- John Yang, Kilian Leret, Carlos E. Jimenez, Alexander Wettig, Kabir Khandpur, Yanzhe Zhang, Binyuan Hui, Ofir Press, Ludwig Schmidt, and Diyi Yang. Swe-smith: Scaling data for software engineering agents, 2025. URL <https://arxiv.org/abs/2504.21798>.
- Daoguang Zan, Zhirong Huang, Wei Liu, Hanwu Chen, Linhao Zhang, Shulin Xin, Lu Chen, Qi Liu, Xiaojuan Zhong, Aoyan Li, et al. Multi-swe-bench: A multilingual benchmark for issue resolving. *arXiv preprint arXiv:2504.02605*, 2025.
- Liang Zeng, Yongcong Li, Yuzhen Xiao, Changshi Li, Chris Yuhao Liu, Rui Yan, Tianwen Wei, Jujie He, Xuchen Song, Yang Liu, and Yahui Zhou. Skywork-swe: Unveiling data scaling laws for software engineering in llms, 2025. URL <https://arxiv.org/abs/2506.19290>.
- Guibin Zhang, Yanwei Yue, Zhixun Li, Sukwon Yun, Guancheng Wan, Kun Wang, Dawei Cheng, Jeffrey Xu Yu, and Tianlong Chen. Cut the crap: An economical communication pipeline for llm-based multi-agent systems, 2024. URL <https://arxiv.org/abs/2410.02506>.
- Wenting Zhao, Nan Jiang, Celine Lee, Justin T Chiu, Claire Cardie, Matthias Gallé, and Alexander M Rush. Commit0: Library generation from scratch, 2024. URL <https://arxiv.org/abs/2412.01769>.

A Analysis of Agent Behavior

To quantify the operational scope and coordination complexity of HAI-Agent, we report several behavioral metrics in Table 5: (i) *LLM Calls*, the cumulative number of model invocations across the entire trajectory (including both Orchestrator and Agents); (ii) *Tool Calls*, the total number of successful interactions with the ACI or AAI; and (iii) *Agent Calls*, the frequency of sub-agent spawning for specific *Explorer* or *Engineer* roles via the agent tool.

Metric	Mean	Max	Min
<i>SWE-bench-Pro</i>			
LLM Calls	79.0	314	22
Tool Calls	84.4	314	22
Explorer Calls	3.1	8	1
Engineer Calls	1.1	7	1
Agent Calls (Total)	4.3	14	1
<i>Commit0-lite</i>			
LLM Calls	357.0	916	74
Tool Calls	410.3	1015	87
Explorer Calls	2.4	4	1
Engineer Calls	3.9	9	1
Agent Calls (Total)	6.3	12	1

Table 5: Statistics on SWE-bench-Pro and Commit0-lite.

Shifting Role Bias across Benchmarks. The per-task call statistics in Table 5 reveal distinct behavioral profiles for bug-fixing versus implementation-from-scratch tasks. On *SWE-bench-Pro*, the *Orchestrator* exhibits a clear bias toward *Exploration*, with *Explorer* calls outnumbering *Engineer* calls nearly three to one (3.1 vs. 1.1 mean calls). This confirms our hypothesis that repository-level bug fixing is primarily an exploration-centric problem, where the core bottleneck lies in navigating massive search spaces to localize root causes. Conversely, on *Commit0-lite*, the ratio flips in favor of implementation: the mean number of *Engineer* calls (3.9) significantly exceeds that of *Explorers* (2.4). This reflects the high implementational density required to synthesize an entire library, where the *Orchestrator* must frequently delegate function-level and file-level implementation tasks to specialized agents.

Managing Trajectory Depth and Scale. Table 5 also highlights the extreme reasoning horizons encountered in multi-agent software engineering. While *SWE-bench-Pro* tasks are relatively concise (mean 79 LLM calls), *Commit0-lite* tasks demand a massive increase in trajectory depth, averaging 357 LLM calls and 410.3 tool calls per task, with maximums exceeding 1,000. In such “long-horizon” scenarios, the role of our handoff interventions becomes critical. By offloading implementation to isolated *Engineer* agents, the *Orchestrator* prevents its own context from being saturated by low-level implementation details. The high maximum of 14 total *Agent* calls on *SWE-bench-Pro* further demonstrates HAI-Agent’s ability to coordinate a large number of discrete subtasks without succumbing to trajectory collapse.

B Per-Run Efficiency Across Scaffolds

Table 6 reports per-run efficiency metrics for HAI-Agent, *SWE-Agent*, and *Claude Code* under identical model configurations, complementing the compute-matched analysis in §4.4.

C Prompts

This appendix lists the system prompts used by HAI-Agent. The role prompts (§C.1) define the three agent roles of §3; the intervention prompts (§C.2–C.4) implement the three handoff

Model	Scaffold	Score (%)	LLM calls	Tool calls	Tokens
<i>SWE-bench-Pro</i>					
Qwen3.5-397B-A17B	SWE-Agent	43.2	78.6	78.6	2.74M
Qwen3.5-397B-A17B	HAI-Agent	47.2	394.8	419.6	13.7M
Claude-4.5-Sonnet	Claude Code	39.1	80.4	85.2	2.70M
<i>Commit0-lite</i>					
Claude-4.5-Sonnet	SWE-Agent	50.3	254.8	254.8	–
Claude-4.5-Sonnet	HAI-Agent	61.6	356.5	409.8	–
Qwen3.5-397B-A17B	SWE-Agent	51.8	255.2	255.2	–
Qwen3.5-397B-A17B	HAI-Agent	59.8	357.3	410.2	–

Table 6: Per-run efficiency under identical model configurations: average LLM calls, tool calls, and total tokens per task. All scaffolds run with their native tool sets and unlimited iteration budgets; trajectories terminate only when the agent emits a finish/submit action.

interventions of §3.2. Placeholders in braces (e.g., {task}) are filled by the harness at runtime.

C.1 Agent Role Prompts

Orchestrator.

You are the Orchestrator coordinating the resolution of a software engineering task in this repository.

Work in two phases:

1. PLAN: explore the repository, understand the issue, and produce a modular breakdown into self-contained subtasks.
2. EXECUTE: delegate subtasks via the ``agent`` tool. Use ``explorer`` agents for read-only investigation and ``engineer`` agents for implementation; assign each engineer a dedicated worktree (create one with ``create_worktree`` if needed).

When delegating, every task description MUST contain:

- Goal: one atomic, well-defined objective for this subtask.
- Context: the relevant file paths, identified dependencies, and everything you already know that the agent needs (findings, ruled-out hypotheses, scope constraints). The agent sees ONLY your task description.
- Success criteria: an explicit, execution-based condition (e.g., a test command that must pass).

When an agent returns, evaluate the return before building on it; if a return is rejected, re-delegate the subtask with the feedback log describing what was wrong.

Call ``finish`` only when the original task is resolved and verified.

Explorer.

You are an Explorer agent. You have READ-ONLY access to the repository through Glob, Grep, and Read. You cannot edit files or run commands.

Assigned subtask:
{task}

Carry out the investigation described above. Be precise: report exact file paths, symbol names, and line numbers. Distinguish facts you verified by reading code from hypotheses. Your return message

should contain everything the Orchestrator needs to act on your findings without re-reading the files itself. Call `finish` with your findings when done.

Engineer.

You are an Engineer agent working in an isolated worktree at {worktree_path}. You have full repository access (Read, Write, Replace, Bash) within this worktree. Your changes are merged into the main workspace only if your return is accepted.

Assigned subtask:
{task}

Implement exactly what the subtask specifies -- no unrelated refactoring. Produce a minimal, precise patch. Run the tests named in the success criteria (and any tests relevant to the files you touched) before returning. Call `finish` with a return message that includes: the files you changed, a summary of the diff, the exact test commands you ran, and their results.

C.2 Delegation Review Prompt

You are reviewing a task description that the Orchestrator is about to dispatch to a sub-agent. The sub-agent will see ONLY this description -- it has no access to the Orchestrator's history.

Task description:
{dispatch_message}

Check the description against three criteria:

1. Atomicity: does it target a single, well-defined objective?
2. Self-containment: are all relevant file paths, identified dependencies, and previously discarded hypotheses explicitly stated?
3. Verifiability: are there clear, execution-based criteria to define success?

If all three hold, output: PASS.

Otherwise output: REVISE, followed by the specific missing or ambiguous items, so the Orchestrator can rewrite the description.

C.3 Retrospective Reflection Prompt

Injected as a system reminder when the agent calls finish; the harness automatically extracts the final git diff and re-injects the original task description.

Before your return is delivered, review your own work.

Your assigned subtask was:
{task}

Your produced changes (git diff):
{diff}

Answer, with evidence:

1. Does your output satisfy the assigned subtask? Judge against the stated success criteria, not your intentions.
2. What verification did you actually perform (exact commands, test names, outputs)? For engineering tasks, include logs from the passing test suite; for exploration tasks, include the evidence trace supporting your claims.
3. What remains unverified or incomplete, if anything?

If verification failed or could not be performed, say so explicitly instead of reporting completion. Your reflection and evidence are checked independently against the workspace; unsupported claims will cause the return to be rejected.

C.4 Verification Fork Prompt

The fork inherits the Orchestrator's full message history; the following system reminder is appended upon an Engineer's return.

An Engineer agent has returned from a delegated subtask. Verify the return independently before it is accepted.

Subtask assigned to the agent:
{task}
Worktree containing the agent's changes:
{worktree_path}
Agent's return message and reflection:
{return_message}

Judge the return STRICTLY against the assigned subtask, interpreted in the context of the overall task. A valid partial solution that satisfies its subtask must be accepted even if the overall task is not yet solved.

Inspect the worktree yourself: read the diff, and re-run the tests named in the success criteria if needed. Cross-check the agent's reflection against the actual diff and logs; treat claims that do not match the workspace as failures.

Output exactly one verdict:
ACCEPT -- the return satisfies the subtask.
REJECT: <feedback> -- otherwise, with concrete feedback on what is wrong, to be passed to a fresh agent on re-delegation.