

Coordinating Agents on a Shared Git Workspace: An Empirical Study of Action-Space Observation for Agentic Software Engineering

Hasen Özyaytürk 

Istanbul Technical University, İstanbul, Türkiye

Feza Buzluca 

Istanbul Technical University, İstanbul, Türkiye

Abstract

Background. Multi-agent approaches to Large Language Model (LLM)-based agentic software engineering, specifically in automated program repair and issue resolution, have converged on architectural patterns that favour isolation: role specialisation, task decomposition in isolated worktrees, or best-of- N with external selection. Collaboration of homogeneous agents on a single task in a shared Git workspace remains understudied, and is actively discouraged in production guidance due to file-level write collisions.

Aims. We investigate whether a shared-workspace configuration can be made viable and effective for autonomous bug fixing and patch production through action-space coordination – a paradigm in which agents observe one another’s commits, lifecycle transitions, and recent test or error outputs rather than exchanging natural-language messages.

Method. We propose PASC (Peer-aware Action-Space Coordination), a layer in which two homogeneous LLM agents share a single Docker container and Git tree. Each agent’s effects are auto-committed under its identity, and each next observation is prepended with a structured peer-activity block. The final submission is the team patch derived from the shared history.

Results. We evaluated PASC on the full Python subset of SWE-Bench Pro using two independently-developed LLMs, against an isolated single-agent baseline and a “silent” two-agent baseline without the peer-activity feed. PASC delivers a statistically significant lift over the single-agent baseline on both models. Crucially, the silent baseline is statistically equivalent to the single-agent one, confirming that the gain stems from action-space coordination rather than parallelism. PASC also outperforms a deployable federated alternative in which agents share the peer-activity feed but work in isolated worktrees. Relative to the silent baseline, PASC reduces cost per resolved task by $\sim 20\%$ and destructive concurrent edits by $\sim 47\%$. Preliminary observations indicate that beyond two agents, active interference increases several-fold, suggesting that larger populations will require supplementary coordination mechanisms.

Conclusions. A shared-workspace configuration of homogeneous agents coordinated through action-space observation improves effectiveness over an isolated single-agent baseline, and does so more economically than a comparable multi-agent baseline without peer awareness.

2012 ACM Subject Classification Software and its engineering $\rightarrow$ Software maintenance tools; Computing methodologies $\rightarrow$ Multi-agent systems; Software and its engineering $\rightarrow$ Software testing and debugging

Keywords and phrases Multi-agent systems, agentic software engineering, large language model agents, shared workspace coordination, SWE-Bench Pro, empirical software engineering

Digital Object Identifier 10.4230/LIPIcs.ESEM.2026.18

Category Technical Track Paper

Supplementary Material

Software (Replication Package): <https://doi.org/10.5281/zenodo.20276750>

Software (Development Repository): <https://github.com/hasenoz/PASC> [29]

archived at `swh:1:dir:f44739f72961a5fb3ed2c5f4c8b2d88faf946f81`

Funding Supported by the Research Fund of Istanbul Technical University. Project Number: 48456.



© Hasen Özyaytürk and Feza Buzluca;
licensed under Creative Commons License CC-BY 4.0

20th International Symposium on Empirical Software Engineering and Measurement (ESEM 2026).

Editors: Robert Feldt, Maria Paasivaara, Daniel Mendez, Stefan Wagner, and Marvin Muñoz Barón;

Article No. 18; pp. 18:1–18:20



Leibniz International Proceedings in Informatics

Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

Issue resolution, bug fixing, and patch production are core software-engineering activities that have traditionally relied on individual developer effort or coordinated team workflows mediated by version control. The recent emergence of LLM-based autonomous agents has introduced a new class of contributor to these workflows, capable of reading issue descriptions, exploring codebases, and producing patches. Agentic software engineering is the discipline that studies how these activities are carried out by such agents. Its systems take several forms. A single agent may be responsible for the entire task, several agents may work on one task together, or the task may be split among agents by subtask or by role. The aim of this study is to test a configuration that has received little attention: two agents of the same kind work on one task in one shared workspace, with no manager assigning subtasks and no selector choosing between their patches, and with each agent kept aware of its peer’s actions as they happen. This configuration is evaluated against a single agent, against two agents that share the workspace without that awareness, and against two agents that receive it but work in separate worktrees. All four are evaluated on the full Python subset of SWE-Bench Pro [11], a contamination-resistant benchmark of long-horizon, multi-file issue-resolution tasks.

Single-agent scaffolds such as SWE-agent [38] resolve a substantial fraction of GitHub issues on SWE-Bench [17] and SWE-Bench Verified [25]. Multi-agent extensions for autonomous program repair are a natural lever for further progress, and the research community has explored them extensively. Most published systems fall into one of a few architectural patterns. Role-specialised frameworks assign distinct agents to roles such as project manager, architect, developer, and tester [30, 16, 24, 2]. Task-decomposition systems route subtasks to agents operating in isolated git worktrees [15]. Best-of- N approaches run independent agents in parallel and reconcile their outputs through an external selector or re-ranker [39, 34]. Sequential trajectory replay runs one agent to completion and feeds its trajectory to the next [19]. Production tools converge on worktree-isolated parallelism, placing each agent in its own worktree with a separate task [9, 4, 5]. Production guidance actively discourages the shared workspace on mechanistic grounds: git itself refuses to check out the same branch in two worktrees [5], a rule designed to prevent index and working-tree corruption, and practitioner reports cite shared-file collisions as the principal failure mode of co-located agents [4].

We argue that the standard objections to a shared-workspace configuration are problems of communication rather than architecture. Prior multi-agent systems that target the same task either communicate through natural-language messages [30, 16] or do not communicate during execution at all [39, 15]; both routes pay an architectural price (role heterogeneity, central managers, or external selectors). We instead make agents communicate through their actions, and we place them on one tree rather than in separate worktrees. After each step, each agent’s bash-command effects auto-commit to a shared git tree under the agent’s identity, and the next observation each agent sees contains a structured digest of what its peer has just done: which files were changed, what tests passed or failed, whether the peer has submitted or appears stuck. The agents do not exchange natural-language messages.

This paper makes the following contributions:

1. We identify a comparatively understudied point in the multi-agent software-engineering design space (Section 2) and build PASC as a coordination layer on top of an existing single-agent framework [32] (Section 3).

2. We evaluate PASC on the full Python subset of SWE-Bench Pro [11] under two independently-developed open-weight LLMs (MiniMax-M2.7 and DeepSeek-V4-Pro), with three independent runs per configuration. On both models the system delivers a statistically detectable improvement over the isolated single-agent baseline, and a $\sim 20\%$ reduction in cost per resolved instance over the multi-agent baseline without peer awareness (Sections 5.1 and 5.4).
3. We isolate the source of the lift through a controlled comparison against a two-agent shared-workspace baseline that disables the peer-activity render (SILENT). Equivalence testing rules out the parallelism explanation: SILENT is statistically equivalent to SINGLE. The lift is attributable to the peer-activity feed (Section 5.2).
4. We isolate the contribution of workspace topology through a controlled federated comparison in which two agents share the same peer-activity feed but work in isolated worktrees rather than a shared tree. The shared-tree configuration outperforms the deployable federated alternative, and holds up even when that alternative is scored on whichever of its two patches happens to be correct – a non-deployable upper bound that requires perfect external selection (Section 5.3).
5. We provide quantitative diagnostics on shared-workspace behaviour. The peer-activity feed reduces destructive co-edits across both LLMs (Section 5.5), and preliminary observations on $N \in \{2, 3, 5\}$ characterise where the current coordination layer plateaus and what an additional mechanism for larger agent populations would need to address (Section 5.6).

2 Related Work

We organise related work along the five families introduced in Section 1, then turn to single-agent baselines on SWE-Bench-style benchmarks, the use of git as a coordination substrate, the literature on inter-agent communication, and coordination findings from pre-LLM collaborative software development.

2.1 Multi-Agent LLM Systems for Software Engineering

Role-specialised systems. ChatDev [30], MetaGPT [16], AgileCoder [24], and SWE-Search [2] instantiate multi-agent systems in which agents are heterogeneous by construction. The pattern recurs at smaller task scales: Melo et al. [21] apply a sequential four-agent pipeline of role-specialised small language models to method-level test-smell refactoring. AutoGen [36] provides a general orchestration framework for heterogeneous agents. None of these systems addresses the case in which two agents of the same role collaborate on the same task; the collaboration mechanism is always role differentiation.

Task-decomposition systems with isolated worktrees. The closest published neighbour to PASC is the CAID paradigm of Geng and Neubig [15]. Both systems use git as the coordination substrate and reach the conclusion that branch-and-merge is the right primitive. The remaining design choices distinguish the two on three axes. First, CAID introduces a central manager that constructs a dependency-aware plan and delegates subtasks; PASC has no central manager. Second, CAID assigns each engineer agent its own isolated git worktree so that concurrent edits cannot interfere; PASC places both agents in a single shared worktree. Third, CAID’s parallelism comes from decomposition (each agent works on a different subtask); PASC’s agents work on the same task end-to-end. Where CAID asks how to make isolation work better through a centralised planner and test-based integration, PASC asks whether isolation is necessary at all.

Best-of- N with external selection. DEI [39] pools heterogeneous SWE-agents (Aider [14], Moatless [27], Agentless [37], OpenHands [35]) and applies a multi-stage rating-and-ranking pipeline. Production systems such as Verdant [34] spawn agents that explore distinct approaches in parallel and orchestrate them through a plan-code-verify loop with a verifier agent for selection. These systems do not share workspace state during the run: agents do not see one another’s edits, and outputs are reconciled by an external selector.

Sequential trajectory replay. SE-Agent [19] addresses the same question as PASC (how should information flow between agents working on the same task?) through a different route: one agent finishes, the next starts from scratch with the prior trajectory as context, with state-of-the-art numbers reported on SWE-Bench Verified. PASC’s answer is parallel action-space coordination: agents run concurrently and observe each other’s commits and test outputs at every step. SWE-Exp [8] is a related variant that operates across issues rather than within a single one, distilling lessons from prior trajectories into an experience bank.

Worktree isolation in production tooling. Developer-facing tools have converged on the same prescription as CAID for the practical setting: place each parallel agent in its own `git worktree`, give it a separate task, and merge sequentially. Cursor 2.0 [9] exposes up to eight parallel agents through worktree or remote-machine isolation; Augment Code’s guides [4, 5] frame the shared-workspace configuration as a primary failure mode and recommend decomposition with explicit boundaries and worktree isolation. Git itself enforces the isolation by refusing to check out the same branch in two worktrees. The empirical question of whether this prescription remains necessary when agents are made aware of one another’s actions is the question this paper takes up.

2.2 Single-Agent Baselines and Benchmarks

The single-agent baseline against which our multi-agent system is compared is MINI-SWE-AGENT [32], a lightweight scaffold descended from SWE-agent [38]. Its developers report that it matches the performance of the original SWE-agent while being substantially simpler, and now recommend it as the default scaffold for research use over SWE-agent itself [33]. Its minimalism is therefore a feature rather than a limitation: it provides a strong single-agent baseline with a small surface area, offering cleaner ground on which the contribution of any extension built atop it can be isolated and attributed.

The original SWE-Bench benchmark [17] has driven a wave of single-agent systems including Agentless [37] (which avoids the agent loop entirely through static structural reasoning), AutoCodeRover [40] and its successor Sonar Foundation Agent [23], Aider [14], and Moatless Tools [27].

2.3 Git as a Coordination Substrate

PASC shares CAID’s [15] choice of git as a coordination substrate, but reverses the isolation polarity by co-locating agents in a single shared tree and treating the commit log as the coordination signal. The closest non-multi-agent precedent is the use of repository state as feedback in self-refining single-agent systems [31, 20], which exploit version-control state for the agent’s own introspection rather than for inter-agent communication.

2.4 Inter-Agent Communication

A separate strand of work has investigated how LLM agents should communicate once spawned. Free-form natural-language chat is the default in role-specialised systems [30, 16]; structured envelopes have been proposed as a way to make routing more reliable. PASC’s premise is that on long-horizon software-engineering tasks, where most useful information is structural (which file was edited, which test passed or failed), the substrate of communication should be the agent’s actions rather than its words.

2.5 Coordination in Collaborative Software Development

The challenge of coordinating multiple contributors on a shared codebase predates LLM agents by decades, and findings from that literature shape PASC’s design. Dourish and Bellotti [13] distinguished passive awareness – coordination information exploited through the shared workspace itself – from active mechanisms that require explicit communication, arguing that the latter penalise providers and presuppose relevance to recipients. PASC’s three-layer peer-activity feed (Section 3) is an instantiation of the passive-awareness principle for LLM agents: commits, lifecycle transitions, and output digests surface through the shared git tree and in-memory boards, with no explicit inter-agent message channel. Brun et al. [7] established that concurrent-edit conflicts in collaborative development are frequent and manifest not only as textual overlaps but also as subsequent build and test failures, and demonstrated that proactive detection mitigates them; PASC’s reduction of destructive co-edits (Section 5.5) is the corresponding agent-coordination result.

3 PASC: Design and Implementation

PASC is built on MINI-SWE-AGENT [32], whose core loop calls a model, runs the model’s bash commands in a sandboxed environment, feeds the outputs back as observations, and terminates when the model emits a designated submission marker. Three protocol-style abstractions define the extension points: **Model** (`query(messages)`), **Environment** (`execute(action)`), and **Agent** (`run(task)`). The single override point inside the loop is `step()`, which constructs one query/observation pair; all extensions in this section are realised through subclasses or wrappers around these three protocols. We describe the system as implemented for two agents; the underlying mechanism is not specific to that choice, and Section 5.6 discusses generalisation to $N > 2$.

3.1 Architecture Overview

For a single SWE-Bench Pro instance the orchestrator constructs the following objects:

- Shared environment.** A single Docker container is started that holds the target repository at the benchmark’s specified base commit. Its `container_id` is captured and exposed.
- Git operations object.** Encapsulates `docker exec ... git ...` invocations against the detected repository path (typically `/testbed` or `/app` depending on the benchmark image).
- Commit tracker.** Owns a per-agent “last-seen” pointer into the git history and serialises commit operations so that attribution remains clean even when agents step concurrently.
- Peer-activity builder.** Renders, on demand, the structured block that will be injected into an agent’s next observation, drawing from the commit tracker and from two thread-safe in-memory boards that hold per-agent lifecycle status and per-agent output digests.
- Shared environment wrapper.** Makes the same container addressable from both agent threads while leaving lifecycle management to the orchestrator.

Two worker threads then run, one per agent. Each thread holds a fresh `Model` instance, points its agent at the same shared environment, and uses the same shared commit tracker and peer-activity builder. A thread terminates when its agent emits the `MINI-SWE-AGENT` submission marker, hits a failure, or exhausts its step limit; the marker is treated as a terminate hint and the string itself is ignored. After both threads have joined, the orchestrator diffs `HEAD` against `base_commit` and writes the result as `final.patch`. Per-agent “raw” patches are persisted for debugging but are not part of the submission. Figure 1 summarises the resulting structure and the data flow at the step boundary.

The agent class extends `MINI-SWE-AGENT`’s `DefaultAgent` by overriding `step()` to insert two hooks: a *pre-step* hook that prepends the peer-activity block to the most recent observation message, and a *post-step* hook that commits any working-tree changes, updates a stuck-detection counter, and publishes a one-shot output digest. We do not override `query()` or `execute_actions()`; both extensions operate at the step boundary because that is the granularity at which peer-observable events occur.

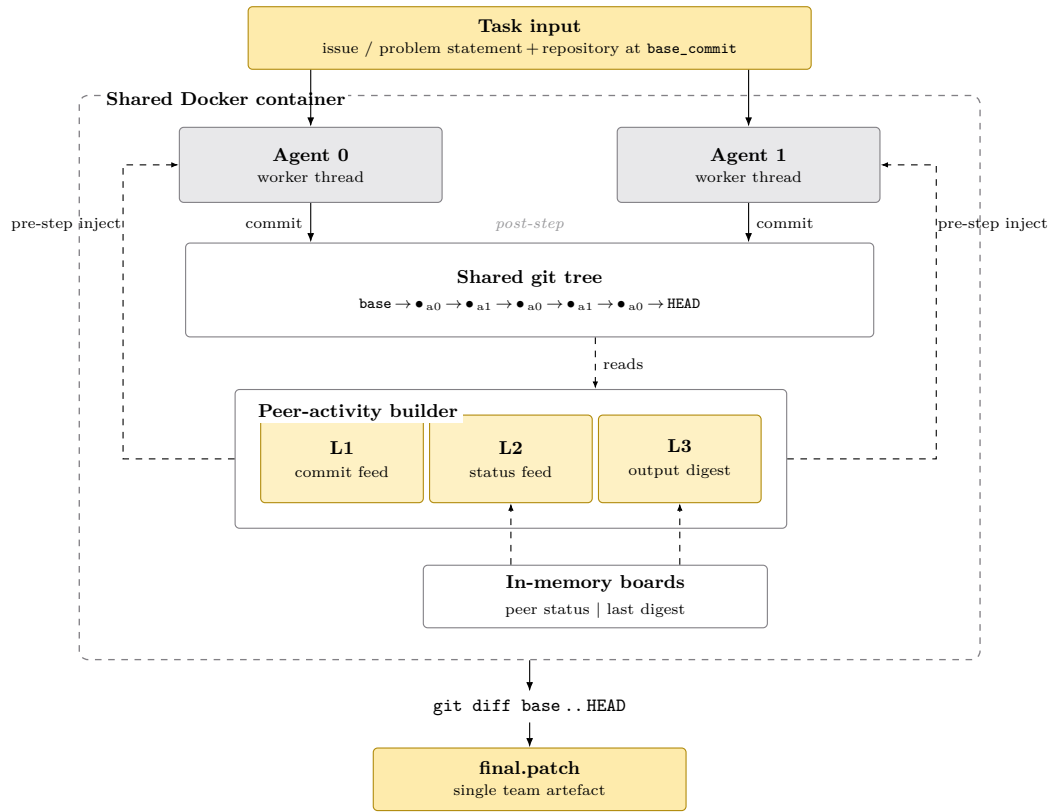
These design choices place PASC outside the multi-agent families discussed in Section 2 on three axes: agents do not exchange natural-language messages, no central coordinator mediates their outputs, and patch selection is not delegated to an external mechanism. The final submission is uniquely determined by the shared git state at the time both agents stop.

3.2 Shared Environment via Composition

A naive subclass of `DockerEnvironment` would not work for this use case: the upstream constructor starts its own container, so two instantiations would launch two containers. We therefore use composition. The orchestrator instantiates one base environment, captures its `container_id`, and wraps it in a `SharedDockerEnvironment` that forwards all methods invoked by the agent loop: `execute` for action execution and `get_template_vars` / `serialize` for prompt and trajectory bookkeeping. The wrapper additionally converts `cleanup()` into a no-op so that container teardown remains the orchestrator’s responsibility, ensuring the shared container survives until both agent threads have joined. Concurrent `shared_env.execute()` calls translate to concurrent `docker exec` invocations into the same container; in the shared configuration, however, a workspace-level lock holds bash execution and the post-step commit together as a single critical section (Section 3.3). LLM queries – typically the dominant component of per-step latency – remain fully parallel.

3.3 Auto-Commit Pipeline and Attribution

After each step the agent’s thought (the assistant text that precedes the bash command) is extracted, normalised to a short single-line summary, and used as the commit subject, prefixed with the agent’s identifier and step number. The post-step commit is then issued under the agent’s identity, with concurrent commits protected by a workspace-level lock that holds bash execution and the commit together as a single critical section: were only the commit serialised, one agent’s uncommitted bash write could be swept into the other’s concurrent staging step and leak attribution. If a commit nevertheless fails under transient contention, the pipeline retries with bounded backoff and, on terminal failure, resets the staged index so that the failing agent’s working-tree changes are not absorbed into any peer’s next commit. Details of the bash pipeline and the git configuration that suppresses background maintenance during the run are documented in the replication package.



■ **Figure 1** Architecture of PASC at $N=2$. Solid arrows show post-step commits to the shared git tree. Dashed arrows show pre-step reads from the tree and in-memory boards (L1/L2/L3 sources) and the resulting peer-activity injection into each agent’s next observation. No oracle or external selector is involved in producing the final submission.

3.4 Three-Layer Peer-Activity Feed

On the next step, before the agent issues its query, the peer-activity builder constructs a single block of text and prepends it to the most recent user message. The block contains up to three layers, each independently computed; layers are omitted when they have nothing to contribute, and an empty block results in no injection at all.

L1 – Commit feed

Using the per-agent “last-seen” pointer maintained by the commit tracker, the builder enumerates commits authored by the peer in the range $(\text{last_seen}, \text{HEAD}]$ and renders each as a short SHA, the commit subject, and a per-file `git show -stat` summary. The pointer is then advanced, so each commit is shown to a given viewer at most once. A configurable cap bounds the number of commits surfaced in a single block, ensuring that the peer-activity injection remains a small fraction of the agent’s context window. The progressive semantics reflect that each commit is new information once and never again; a monotone pointer suffices.

L2 – Status feed

Lifecycle events (the peer has submitted, the peer appears stuck through consecutive format errors above a threshold, the peer’s worker has failed) are written to a thread-safe status board. The renderer checks whether the current peer status differs from the most recently surfaced status for this viewer agent and, if so, emits a one-shot notice with an actionable hint. This layer addresses a gap of the L1-only design: the submission marker produces no commit of its own and was therefore invisible to the peer. L2 is event-shaped: the “submitted” state is a transition, and only the transition needs surfacing.

L3 – Output digest

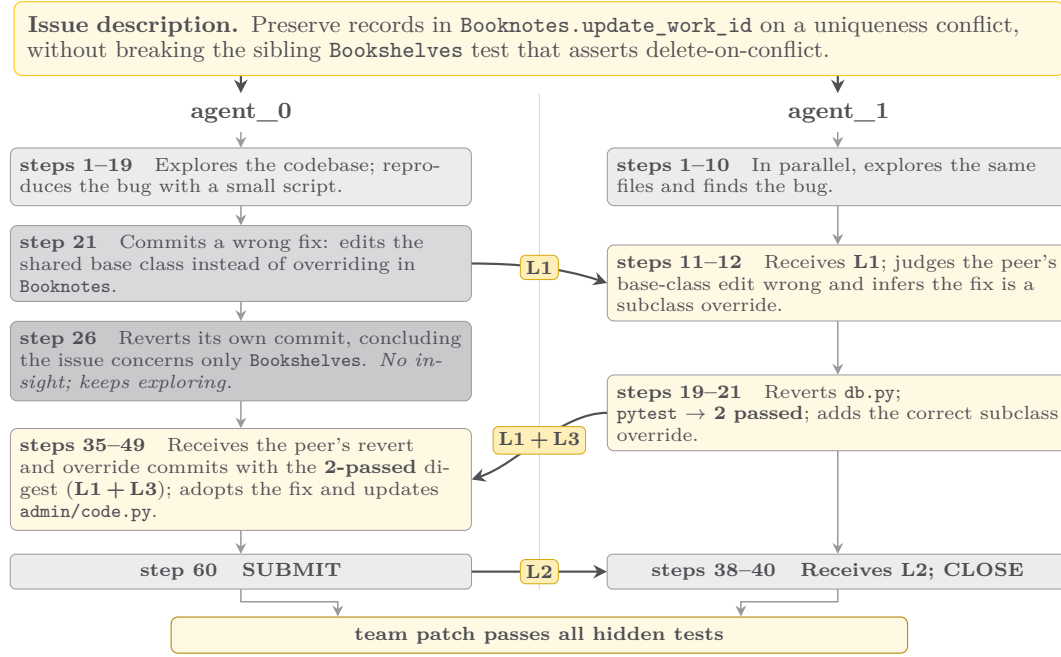
After a step that produced a bash observation, six regular-expression patterns are tried in priority order against the command and its output: a `pytest` summary line, a `unittest` “Ran N tests in T s” line, a Python traceback, a `git apply` or `git am` failure, an invocation of a peer-authored Python script, and a generic non-zero return code. The first match is recorded as a single-line digest in a thread-safe digest board under the producing agent. The peer’s next render reads this digest, emits it, and clears it, so the same digest is never replayed on subsequent steps. The peer-authored-script pattern catches the case where one agent runs a reproduction script the other has just committed; without that pattern, an exit-zero “BUG NOT REPRODUCED” output would remain invisible despite being structurally important.

The L3 extractor applies one additional annotation uniformly: when a peer’s `pytest` run produces only `PASSES`, an annotator automatically wraps the digest with a reminder that the test set was peer-selected rather than the hidden gold suite. Without this annotation, an all-pass result could be misread by the viewer as evidence that the peer has already produced a working patch, leading the viewer to terminate prematurely on what is, in fact, a peer-selected outcome.

The three layers have distinct cardinality semantics: L1 is progressive (each commit shown once, then never again), L2 is event-shaped (only transitions surface), and L3 is read-and-clear (the most recent volatile signal, replaced rather than accumulated). Figure 2 illustrates how the three layers are triggered on a single SWE-Bench Pro instance. `agent_0` commits an incorrect fix and resumes searching without recognising its architectural error; on the next observation `agent_1` receives this commit through L1, analyses it as the peer’s work rather than its own, and arrives at the correct fix. The revert and override commits, together with the passing test digest, return to `agent_0` through L1 + L3, and L2 synchronises submission.

3.5 Design Verification on a Pilot Corpus

The three-layer design was settled empirically on a 50-instance pilot corpus drawn from the Python subset of SWE-Bench Pro through balanced sampling across the `ansible`, `openlibrary`, and `qutebrowser` repositories. We inspected agent trajectories to settle qualitative design choices – which regular-expression patterns L3 should recognise, where to bound the L1 commit cap, when L2 should fire – and to verify that each layer’s marginal addition was budget-neutral or budget-positive at small scale. No design choice was conditioned on pilot pass counts: at $n=50$, pass-count differences between consecutive layer additions fall within a 1–3 instance range, well below the statistical resolution of any paired test. Per-layer token, commit, and cost diagnostics are reported in the replication package.



■ **Figure 2** End-to-end trace of PASC on a single SWE-Bench Pro instance (`internetarchive/openlibrary @ 5069b09`) under DeepSeek-V4-Pro. The two agents work in parallel from the issue description; horizontal arrows mark the inter-agent signals carried by L1 (peer commit), L1 + L3 (commit with test digest), and L2 (submission transition); the joined flow at the bottom is the submitted team patch.

4 Experimental Setup

4.1 Research Questions

We organise the evaluation around six research questions:

- RQ1 (effectiveness).** Does action-space coordination over a shared workspace improve task resolution over an isolated single-agent baseline, at the same model and decoding regime, and is the effect consistent across independently-developed LLMs?
- RQ2 (mechanism – parallelism).** Is the observed lift attributable to *parallelism* or to the *peer-activity feed*? Specifically, does a two-agent baseline that shares the workspace but disables the peer-activity render (SILENT) match PASC, or is it instead equivalent to the single-agent baseline?
- RQ3 (mechanism – topology).** When two agents share the same peer-activity feed but work in isolated worktrees rather than a shared tree, does the shared-tree configuration of PASC still provide an advantage?
- RQ4 (cost-effectiveness).** How does PASC’s cost per resolved instance compare to the single-agent baseline and to a multi-agent baseline without peer awareness?
- RQ5 (failure-mode mitigation).** Does the peer-activity feed mitigate the destructive concurrent-edit thrash that motivates industry guidance against shared workspaces? If so, is the mitigation a routing effect, a cooperation effect, or both?
- RQ6 (preliminary scaling observations).** Does the $N=2$ design point generalise to larger agent populations $N \in \{3, 5\}$ under the present coordination layer, or do additional mechanisms become necessary?

4.2 Benchmark and Corpora

We evaluate on the Python subset of SWE-Bench Pro [11], comprising 266 instances. SWE-Bench Pro is a contamination-resistant corpus of long-horizon, multi-file coding tasks drawn from 11 publicly released repositories. We deliberately adopt it rather than the earlier SWE-Bench Verified [25], which has been shown to suffer from solution leakage and weak test oracles in roughly one-third of instances [1], and whose results OpenAI has itself ceased reporting, citing contamination and recommending SWE-Bench Pro as the successor [26]. Resolution is determined by a hidden test suite combining fail-to-pass tests (verifying issue resolution) and pass-to-pass tests (verifying no regressions). The Python subset was selected as the evaluation target because the benchmarks that established this literature are Python-only [17, 25], maximising comparability with the baselines against which PASC is positioned.

The full 266-instance corpus serves as the primary evaluation target for RQ1–RQ5. A 76-instance subset is used for the $N \in \{2, 3, 5\}$ scaling observations of RQ6. The 50-instance pilot corpus used for verification of the three-layer feed (Section 3.5) is a subset of the 266-instance evaluation corpus; we discuss the implications in Section 6.

4.3 Models and Decoding Regime

To reduce model-specific external-validity threats, we evaluate on two independently-developed open-weight LLMs: MiniMax-M2.7 [22] and DeepSeek-V4-Pro [10]. Both are recent open-weight models, competitive with frontier closed-weight systems on coding tasks at a fraction of the per-token cost. Open weights also keep the evaluation replicable at a later date, whereas proprietary endpoints are subject to undocumented updates and deprecation. All runs use greedy decoding (temperature 0) to reduce decoding-side variance. Greedy decoding alone does not make a run reproducible, and the residual variance can shift measured accuracy [3, 28]; each configuration is therefore repeated three times. Each agent operates under MINI-SWE-AGENT’s default per-agent step limit of 250 steps; no token or cost ceiling was imposed on any configuration. The multi-agent configurations apply the same per-agent step limit. All costs are reported under provider list pricing: MiniMax-M2.7 at \$0.30/M input and \$1.20/M output tokens, DeepSeek-V4-Pro at \$0.435/M input (cache miss) and \$0.87/M output tokens. The cost-computation scripts and full pricing details are included in the replication package.

4.4 Configurations

Four configurations are evaluated on the full 266-instance corpus:

- single.** A single MINI-SWE-AGENT with no coordination layer; serves as the lower-bound single-agent reference.
- silent.** Two agents on a shared container and shared git tree with auto-commit enabled, but with the peer-activity render *disabled*. Agents observe peer activity only through explicit git inspection commands they choose to issue. This is the mechanism control for RQ2.
- federated.** Two agents working in isolated git worktrees on the same task, with prompts adapted to inform each agent that its peer works in a separate worktree. Final patches are derived per-agent and aggregated post-hoc; we report both individual-agent submissions (deployable) and an oracle UNION that counts an instance as resolved if either agent resolved it (upper bound). This is the topology control for RQ3.
- PASC.** The full system as described in Section 3: shared container, shared git tree, auto-commit, and the three-layer peer-activity feed ($L1 + L2 + L3$).

4.5 Metrics

Pass count: the number of instances whose final patch passes the hidden test suite under the SWE-Bench Pro evaluator. **Total cost:** dollar spend across the corpus at list pricing. **Cost per pass:** total cost divided by pass count, the cost-effectiveness metric most directly comparable across configurations of differing token budgets. **Co-edit rate:** the fraction of instances in which both agents issue mutating bash commands targeting overlapping source paths in the repository. Detection is performed on the bash command text via regex against mutating patterns, not on post-hoc git diffs: the metric targets interference attempts, including those git refuses or that exit non-zero, which leave no trace in the committed history. **Any-destructive rate:** a strict subset of co-edit in which the two agents' edits exhibit an overwrite signature. **Sed-reversal rate:** a strict subset of any-destructive, in which an agent's bash command targets a line the peer wrote in a preceding commit. By construction, $\text{sed-reversal} \subseteq \text{any-destructive} \subseteq \text{co-edit}$.

4.6 Statistical Analysis

The unit of analysis is the SWE-Bench Pro instance; outcomes are paired across configurations within model. For RQ1 we report McNemar's test, the appropriate paired test when both conditions are evaluated on the same instances [12], per stratum (model) and a stratified pooled McNemar test across the six strata. We complement these frequentist tests with two model-based analyses, which capture different aspects of the effect: a population-averaged GEE logistic model estimates the marginal odds ratio, while a generalised linear mixed-effects model (GLMM) with random intercepts for instance and run absorbs the clustering induced by repeated evaluation of the same instances and estimates the within-instance conditional odds ratio. Stratum heterogeneity is assessed via the Breslow-Day test [6]; this is the formal precondition for pooling.

For RQ2, equivalence between SILENT and SINGLE is assessed by paired two one-sided tests (TOST) [18] at an equivalence margin of ± 5 percentage points, both per-stratum and pooled. Effect sizes are reported as risk-difference (percentage-point lift) with 95% bootstrap confidence intervals, and as discordant-pair odds ratios (b/c).

5 Results

5.1 RQ1: Headline Results on the Full Corpus

Table 1 reports outcome counts and statistical contrasts on the full 266-instance Python subset of SWE-Bench Pro for the two LLMs and four configurations.

PASC resolves +5.3 pp more instances than SINGLE on MiniMax (138.3 $\rightarrow$ 152.3) and +4.5 pp on DeepSeek (152.0 $\rightarrow$ 164.0). The primary inference is the pooled stratified McNemar analysis on $n=1596$ paired observations. It is significant for PASC against both SINGLE and SILENT (Table 1). Pooling is licensed by the Breslow-Day homogeneity test, which rejects neither contrast ($p=0.969$ for PASC vs. SINGLE, $p=0.513$ for PASC vs. SILENT). The effect of method on outcome is homogeneous across strata. Bootstrap 95% confidence intervals for the pooled risk differences are [+2.1, +7.3] pp for PASC vs. SINGLE and [+1.6, +6.3] pp for PASC vs. SILENT, both excluding zero.

The model-based analyses agree. A GEE logistic with exchangeable working correlation gives a marginal odds ratio of 1.22 (95% CI [1.10, 1.36], $p=1.66 \times 10^{-4}$) for PASC vs. SINGLE. A generalised linear mixed-effects model with random intercepts for instance and run gives a conditional odds ratio of **1.75** (95% CI [**1.48**, **2.07**]). The marginal OR describes the

■ **Table 1** Outcome and cost on the full 266-instance Python subset of SWE-Bench Pro under two independently-developed LLMs, averaged across three independent runs per configuration. Costs are reported at provider list pricing. The FEDERATED row evaluates each agent’s patch separately and reports the mean of the two pass counts, with the corresponding cost-per-pass; the oracle UNION upper bound and its associated contrasts are reported in Section 5.3.

LLM	Configuration	Pass / 266	%	Total \$	\$/pass
DeepSeek-V4-Pro	SINGLE	152.0	57.1%	\$9.07	\$0.060
	SILENT	157.0	59.0%	\$24.85	\$0.158
	FEDERATED	156.4	58.8%	\$21.43	\$0.137
	PASC	164.0	61.7%	\$22.47	\$0.137
MiniMax-M2.7	SINGLE	138.3	52.0%	\$12.02	\$0.087
	SILENT	138.3	52.0%	\$30.56	\$0.221
	FEDERATED	140.5	52.8%	\$20.66	\$0.147
	PASC	152.3	57.3%	\$24.36	\$0.160

Pooled stratified McNemar ($n=1596$, six strata = two models $\times$ three runs): PASC vs. SINGLE $\chi^2=22.20$, $p=2.45 \times 10^{-6}$; PASC vs. SILENT $\chi^2=15.21$, $p=9.64 \times 10^{-5}$; SILENT vs. SINGLE $\chi^2=0.82$, $p=0.366$.

population-averaged effect; the conditional OR describes the within-instance effect controlling for instance difficulty. The divergence reflects the dominant instance-level heterogeneity: the instance random effect has SD 3.79 on the logit scale (95% CI [3.48, 4.13]) and accounts for 81% of total logit-scale variance. Run-level variance, by contrast, is negligible (SD 0.27, 95% CI [0.11, 0.69], 0.4% of total variance).

As an additional fairness check, we considered an extended-budget single-agent configuration in which the step limit was doubled to 500 steps. Since a larger budget can only alter a run that the limit actually truncated, we examined the step-bound instances: across all three runs, only one instance reached the 250-step ceiling on MiniMax-M2.7 and two on DeepSeek-V4-Pro. We reran these under the extended budget, and none were resolved. The observed PASC lift is therefore not attributable to single-agent step truncation, and an extended-budget single-agent configuration would not close the gap.

The consistency across MiniMax and DeepSeek (two independently-developed LLMs) is the central external-validity control of this section. It supports a positive answer to RQ1: action-space coordination over a shared workspace improves task resolution over an isolated single-agent baseline, and the effect is detectable on both models.

5.2 RQ2: The Lift Is Not from Parallelism

A natural alternative explanation for the RQ1 result is compute scaling: any two-agent configuration might beat one agent simply because two agents explore more. The SILENT baseline is the controlled test of this explanation. It runs two agents in the shared container and on the shared git tree, with auto-commit on, but with the peer-activity render disabled. If parallelism alone drove the lift, SILENT should outperform SINGLE.

It does not. SILENT resolves 157.0 of 266 instances on DeepSeek against 152.0 for SINGLE, and resolves the same number as SINGLE on MiniMax (138.3 of 266); neither difference reaches significance (pooled $\chi^2=0.82$, $p_{\text{McN}} = 0.366$). Paired two one-sided tests for equivalence at a ± 5 pp margin confirm the pattern: both strata are statistically equivalent ($p_{\text{TOST}} = 3.12 \times 10^{-4}$ MiniMax; $p_{\text{TOST}} = 0.0173$ DeepSeek), and the pooled analysis across $n = 1596$ paired observations is equivalent with $p_{\text{TOST}} = 4.64 \times 10^{-5}$. A population-averaged

logistic regression gives an odds ratio of 0.96 (95% CI [0.87, 1.06], $p = 0.449$) and a mixed-effects model gives a conditional odds ratio of 0.91 ([0.78, 1.06]), neither distinguishable from no effect. Against this null, the pooled McNemar test for PASC versus SILENT is significant at $p = 9.64 \times 10^{-5}$ (Section 5.1). Adding a second agent to the shared workspace, in the absence of the peer-activity render, does not measurably change resolution effectiveness. The lift observed in RQ1 is therefore attributable to the peer-activity feed rather than to the addition of a second agent. Figure 2 provides a qualitative instance of this mechanism: on a SWE-Bench Pro instance where the single-agent baseline and both agents under the SILENT baseline fall into the same architectural error as `agent_0` and submit a wrong fix, the peer-activity feed allows PASC to recover.

5.3 RQ3: Shared Workspace Outperforms Federation

Both industry guidance [4, 5] and the closest published multi-agent neighbour, CAID [15], prescribe isolated git worktrees as the safe topology for multi-agent software-engineering agents. RQ3 isolates this single design choice. Holding all other PASC design decisions constant – no central manager, same task end-to-end, identical three-layer peer-activity feed – does the shared-tree topology outperform the isolated-tree alternative?

We evaluated this contrast as a federated configuration: two agents in separate worktrees observing the same peer-activity feed (commits, lifecycle status, output digests) as in the shared setting, with prompts adapted to inform each agent of the federated topology. CAID’s other design choices (central manager, task decomposition into different subtasks) are deliberately not adopted, so the comparison isolates the worktree topology decision rather than reproducing CAID. Final patches are derived per-agent; we report both individual-agent submissions (deployable) and an oracle UNION (upper bound).

Under the deployable regime, PASC outperforms each federated agent on both models. On MiniMax, PASC resolves 152.3 of 266 against 140.7 for federated `agent_0` and 140.3 for federated `agent_1`; on DeepSeek, PASC resolves 164.0 against 156.0 for `agent_0` and 156.7 for `agent_1`. Per-stratum stratified McNemar tests (pooled across the three runs within each model, $n=798$ per cell) are significant in all four model-by-agent contrasts: MM vs. `agent_0` $\chi^2=7.80$, $p=5.22 \times 10^{-3}$; MM vs. `agent_1` $\chi^2=8.42$, $p=3.72 \times 10^{-3}$; DS vs. `agent_0` $\chi^2=5.54$, $p=0.019$; DS vs. `agent_1` $\chi^2=4.65$, $p=0.031$. Per-stratum odds ratios cluster tightly between 1.54 and 1.61, and the Breslow-Day test rejects no heterogeneity, licensing pooled inference. The pooled stratified McNemar on $n=1596$ paired observations is $\chi^2=13.34$, $p=2.60 \times 10^{-4}$ against `agent_0` and $\chi^2=13.04$, $p=3.05 \times 10^{-4}$ against `agent_1`. The two federated agents are themselves statistically indistinguishable within each stratum (all per-run McNemar $p \geq 0.42$), so the contrast does not depend on which federated agent is selected.

Under the oracle regime, FEDERATED reaches 151.7 of 266 on MiniMax and 160.3 on DeepSeek, neither distinguishable from PASC per-stratum (MM $\chi^2=0.03$, $p=0.865$; DS $\chi^2=1.22$, $p=0.269$). The pooled stratified contrast on $n=1596$ is $\chi^2=0.71$, $p=0.398$. The shared-workspace configuration thus matches the federated upper bound on both models even though that bound requires information the deployed system does not have. Within this comparison the outcome tracks the shared-tree topology rather than the mere fact of running two agents on the same task: the shared tree is not surpassed even when the federated alternative is allowed oracle selection, and outperforms it once the alternative is restricted to the selection a deployed system could actually perform. The cost comparison between the two configurations, including selector-mechanism caveats, is reported in Section 5.4.

5.4 RQ4: Cost-Effectiveness

Cost figures are reported in Table 1 alongside outcomes.

Two cost facts stand out. First, `SINGLE` has the lowest cost per pass on both models; this reflects the two- to threefold total spend of the multi-agent configurations and is a genuine trade-off rather than an inefficiency. Second, both `PASC` and `FEDERATED` substantially improve cost-per-pass over `SILENT`: \$0.137 and \$0.137 vs. \$0.158 on DeepSeek-V4-Pro (−13.4%, −13.4%), \$0.160 and \$0.147 vs. \$0.221 on MiniMax-M2.7 (−27.6%, −33.5%). The peer-activity feed is budget-positive regardless of topology: agents waste fewer tokens on redundant exploration when they can see each other’s commits and recent outputs.

Between `PASC` and `FEDERATED` on a deployable basis, cost-per-pass is identical on DeepSeek-V4-Pro (\$0.137 for both) and slightly higher for `PASC` on MiniMax-M2.7 (\$0.160 vs. \$0.147, +8.8%). These federated figures already credit the two-agent run with the average resolution rate of its agents; an oracle-selected federated configuration would push the pass count higher (Section 5.3) but is not deployable, since any practical selector – whether an LLM judge, a test-based verifier, an external re-ranker, or another mechanism – adds inference or infrastructure cost not reflected in the table, and offers no guarantee of perfect selection accuracy. `PASC`’s submission is uniquely determined by the shared git state and incurs no such selector overhead. Combined with the outcome contrast of Section 5.3, `PASC` is effectiveness-dominant against the deployable federated alternative on both models and matches the oracle federated upper bound that requires non-deployable patch selection, at a cost identical on DeepSeek-V4-Pro and modestly higher on MiniMax-M2.7 in exchange for the no-selector deployability property.

5.5 RQ5: The Peer-Activity Feed Mitigates Destructive Thrash

Industry guidance against shared-workspace multi-agent configurations rests on a specific failure mode: two agents repeatedly overwrite a contested file, with one agent’s edit undoing what the peer has just written [4, 5]. RQ5 asks whether action-space coordination mitigates this empirically, and whether the mitigation is a routing effect (agents stay out of each other’s files) or a cooperation effect (agents observe peer edits and do not overwrite them on shared files).

We measure contention at three nested granularities defined in Section 4.5: *co-edit* (broad), *any-destructive* (narrow), and *sed-reversal* (narrowest). Table 2 reports all three on the full $n=266$ corpus under both LLMs.

Routing effect: The peer-activity feed reduces the co-edit rate by 43% on MiniMax (128.3 $\rightarrow$ 72.7) and 52% on DeepSeek (43.7 $\rightarrow$ 21.0), both significant under CMH-pooled McNemar (MH-OR 3.67 and 3.43 respectively). Agents that observe which files the peer is working on tend to select different files.

Destructive subset: Within the co-edit subset, destructive-overwrite signatures account for 72% of cases under `SILENT` on MiniMax (92.7/128.3) and 71% on DeepSeek (31.0/43.7), so co-edit on this corpus typically manifests as actual interference rather than cooperative editing. The feed reduces the any-destructive rate by 45% on MiniMax and 49% on DeepSeek (MH-OR 3.78 and 3.30).

Cooperation effect: Sed-reversal is observed only on MiniMax; DeepSeek produces zero events under either configuration, suggesting that the canonical sed-based thrash signature is partly a model-specific manifestation. Where the pattern is present, the feed nearly eliminates it: 4.0 $\rightarrow$ 0.3 on MiniMax (12 silent events vs. 1 `PASC` event total across the three runs), with CMH-pooled MH-OR 12.0 ($p=2.28 \times 10^{-3}$). The per-co-edit sed-reversal

■ **Table 2** Contention diagnostics on the full 266-instance Python subset of SWE-Bench Pro. Counts are 3-run mean flagged instances out of 266. CMH-pooled McNemar across 3 runs: MiniMax co-edit $\chi^2=96.0$, $p<10^{-15}$, MH-OR 3.67; any-destructive $\chi^2=72.7$, $p<10^{-15}$, OR 3.78; sed-reversal $\chi^2=9.31$, $p=2.28 \times 10^{-3}$, OR 12.0 (12 silent events vs. 1 PASC event total across 3 runs); DeepSeek co-edit $\chi^2=37.3$, $p=1.0 \times 10^{-9}$, OR 3.43; any-destructive $\chi^2=24.6$, $p=7.0 \times 10^{-7}$, OR 3.30; sed-reversal undefined (no events in either condition).

LLM	Signal	Silent	PASC
MiniMax-M2.7	co-edit	128.3	72.7
	any-destructive	92.7	51.0
	sed-reversal	4.0	0.3
DeepSeek-V4-Pro	co-edit	43.7	21.0
	any-destructive	31.0	15.7
	sed-reversal	0	0

rate on MiniMax drops from 3.1% (4.0/128.3) under SILENT to 0.4% (0.3/72.7) under PASC: even when routing fails, the feed prevents agents from targeting peer-written lines. The conditional sample is small, but the direction is consistent with two mitigation effects – routing and cooperation – operating independently.

The peer-activity feed thus reduces contention at all three granularities on both LLMs and largely removes the sed-reversal pattern where it occurs. The mechanism delivering the RQ1 outcome lift also addresses the failure mode that motivates industry guidance against shared workspaces.

5.6 RQ6: Preliminary Observations on $N > 2$

We evaluated $N \in \{2, 3, 5\}$ on a 76-instance subset of the SWE-Bench Pro Python corpus under MiniMax-M2.7, repeated across two independent runs. At $n=76$, paired tests detect only effects of roughly ± 10 pp, so we report effect directions rather than per-cell significance. Table 3 reports pass rates and contention diagnostics.

Pass rate remains within a ± 5 pp band across all three configurations (Cochran’s Q non-significant in both runs, $p \geq 0.28$; pairwise McNemar all $p \geq 0.13$). At $n=76$ a sweet-spot or modest degradation of ~ 5 pp would be invisible; we therefore read the data as consistent with the coordination layer continuing to operate at higher N , while small effects on outcome remain undetectable at this sample size.

The substantive effect of N is on contention. Co-edit rises from 16.8% at $N=2$ to 53.0% at $N=5$, and any-destructive from 10.8% to 46.3% – approximately a three- to four-fold increase in active interference. Two pressure sources explain this trend. First, the probability that two agents touch the same file in the same step grows combinatorially with N : for $N=2$ there is a single peer pair, for $N=3$ three, for $N=5$ ten. Second, the peer-activity render must aggregate $N-1$ peers’ activity in each layer, so the routing signal each agent receives weakens as N grows even when the per-pair signal strength is constant. Sed-reversal, by contrast, remains rare at all N (0.0%, 2.7%, 3.4%). The cooperation mechanism documented in RQ5 thus continues to suppress targeted-line reversal at higher N , even as routing-level interference grows several-fold.

The present coordination layer is calibrated for $N=2$; the routing signal it carries weakens as N grows. At $N > 2$ the system continues to produce successful patches but accumulates routing-level interference several-fold, suggesting that larger agent populations would benefit from an additional coordination mechanism, with a cooperative claim layer the natural candidate (Section 7).

■ **Table 3** $N \in \{2, 3, 5\}$ comparison on a 76-instance subset under MiniMax-M2.7, averaged across two independent runs. Pass rate differences across N are not statistically detectable (Cochran’s Q $p \geq 0.28$ in both runs; pairwise McNemar all $p \geq 0.13$). Contention signatures follow the definitions of Section 4.5; per-run counts are in the replication package.

Signal	$N=2$	$N=3$	$N=5$
Pass rate (%)	59.1	63.8	61.1
Co-edit (%)	16.8	32.2	53.0
Any-destructive (%)	10.8	25.6	46.3
Sed-reversal (%)	0.0	2.7	3.4

6 Threats to Validity

The evaluation uses greedy decoding (temperature 0) with three independent runs per configuration. The consistency of the lift across MiniMax-M2.7 and DeepSeek-V4-Pro and across all three runs, together with the Breslow-Day rejection of effect heterogeneity and the negligible run-level random-effect variance (ICC 0.4% in the GLMM analysis), supports the robustness of the observed effect.

Generalisation of the empirical claims is bounded by benchmark scope. We chose SWE-Bench Pro precisely for its contamination-resistance properties (Section 4.2); its public Python subset still inherits the structural constraints of long-horizon multi-file Python tasks, and the relative performance of action-space coordination may differ on benchmarks whose tasks are shorter-horizon, single-file, or expressed in languages whose project layout yields a weaker structural signal through the file system. The 76-instance scaling subset of Section 5.6 is a stratified sample from the same pool as the full 266-instance corpus and inherits its scope. For resource reasons, this scaling analysis is restricted to a smaller subset and a single model (MiniMax-M2.7), and its results are intended as scaling observations rather than confirmatory measurements.

Outcome quality is operationalised by SWE-Bench Pro’s hidden test suite combining fail-to-pass and pass-to-pass tests, which we adopt as the field-standard proxy for issue resolution; the construct limitations of this proxy carry over from the underlying benchmark. The contention diagnostics of Section 5.5 are constructed from the agents’ issued bash commands, with the sed-reversal signal additionally referencing the shared git history to identify peer-written lines; they surface the failure modes that motivate industry guidance against shared workspaces, but are operational measures of interference rather than ground-truth semantic equivalence between peer edits.

The primary inference is the pooled stratified analysis on $n = 1596$, with agreement across McNemar, GEE, and GLMM reducing inferential artifact risk. The SILENT baseline’s agents share a git tree and could in principle observe one another through explicit git commands; if this backdoor channel produced coordination, the SILENT $\equiv$ SINGLE equivalence would be harder to obtain, not easier. The $n=76$ scaling subset of Section 5.6 admits detectable effects only above ± 10 pp, and the reported pass-rate invariance across $N \in \{2, 3, 5\}$ should be read with this minimum detectable effect in mind.

The 50-instance pilot corpus used to verify the three-layer feed (Section 3.5) overlaps with the 266-instance evaluation corpus. The pilot involved no algorithmic training and no parameter fitting; its verification targets were operational (regex coverage, injected-block token budget, L2 event shapes) rather than outcome-shaped, and at $n=50$ pass-count differences between consecutive layer additions fall within a 1–3 instance range – below the statistical resolution of any paired test – so no design choice could have been outcome-selected from the pilot.

7 Conclusion and Future Work

Homogeneous agents can share a single workspace and a single task without a central manager, an external selector, or a post-hoc merge. What makes this viable is coordination through a structured peer-activity feed over the shared git tree. Under two independently-developed open-weight LLMs, the resulting system resolves more instances than an isolated single-agent baseline, at a lower cost per resolved instance than a two-agent baseline without peer awareness, and with reduced contention on the shared workspace. Without peer awareness, two agents on the shared tree are statistically equivalent to a single agent: the gain comes from the observation, not the second agent. The same feed over isolated worktrees is outperformed under deployable patch selection and attains parity with the shared tree only when granted oracle selection, placing the remainder of the effect in the workspace topology rather than in the feed alone.

For software engineering practice, the implication is that a portion of what an autonomous agent fails to resolve in isolation is recoverable through the addition of a second agent able to observe its work, with the recovery attributable to the observation rather than to the additional agent. Where a single agent proves insufficient on the long-horizon, multi-file tasks that characterise this benchmark, coordination offers a route to extend what a given model resolves, without recourse to a stronger model or a larger step budget. Worktree isolation eliminates concurrent-edit collisions by construction, but substitutes for them the problem of reconciling two patches, whether by selection or by merge. The shared tree instead leaves a single patch, and the peer-activity feed lowers contention across all three signals relative to the SILENT baseline.

Several directions remain open. The present evaluation uses greedy decoding with three independent runs per configuration; while the GLMM analysis suggests that run-level variance contributes little to the method comparison at this scale, a wider sweep – more runs per configuration, alternative inference engines, and non-zero temperatures – would characterise the joint effect of provider-side stochasticity and decoding noise on the coordination lift, particularly on instances near the resolution boundary. The minimal design suffices at $N=2$. At $N > 2$ the system continues to produce successful patches but accumulates routing-level interference several-fold (Section 5.6), suggesting that larger agent populations would benefit from an additional coordination mechanism. A natural next step is an L4 cooperative claim layer evaluated as a $\{\text{no-L4}, \text{L4}\} \times \{N=3, N=5\}$ factorial, isolating its marginal contribution from the agent count. Beyond scaling, evaluating PASC across a wider range of model strengths would test whether the lift remains constant, scales with single-agent strength, or saturates as the single-agent ceiling rises. The coordination mechanism is not specific to MINI-SWE-AGENT; re-evaluating PASC atop richer scaffolds is a natural extension. Finally, the architecture already supports heterogeneity at the system-prompt level: agents can be assigned distinct roles or strategies within the same shared workspace without changes to the coordination layer. A follow-on study pairing heterogeneous models or role-specialised prompts in a single shared tree would extend PASC to combine model diversity [39] with action-space coordination.

Data Availability Statement

The implementation, per-instance outcome data, analysis scripts, and a README for reproducing the headline numbers are available at doi.org/10.5281/zenodo.20276750 and github.com/hasenoz/PASC. The release includes: (i) the implementation of PASC on top of MINI-SWE-AGENT v2.2.6; (ii) the SWE-Bench Pro Python subset configuration files; (iii)

per-instance pass/fail vectors for the four configurations (SINGLE, SILENT, FEDERATED, PASC) on the full 266-instance corpus under both LLMs, and for the $N \in \{2, 3, 5\}$ scaling subset under MiniMax-M2.7, with the 50-instance pilot diagnostics kept separate from the confirmatory data; (iv) the scripts that compute every McNemar, Breslow-Day, GEE, GLMM, and TOST contrast reported in the paper; and (v) the LLM-provider invocation scripts, cost computation, and pricing details. Model weights are not redistributed; reproducing LLM inference requires access to the respective providers' APIs or self-hosted weights.

References

- 1 Reem Aleithan, Haoran Xue, Mohammad Mahdi Mohajer, Elijah Nnorom, Gias Uddin, and Song Wang. SWE-Bench+: Enhanced coding benchmark for LLMs. arXiv:2410.06992, 2024. doi:10.48550/arXiv.2410.06992.
- 2 Antonis Antoniadis, Albert Örwall, Kexun Zhang, Yuxi Xie, Anirudh Goyal, and William Wang. SWE-Search: Enhancing software agents with Monte Carlo tree search and iterative refinement. In *The Thirteenth International Conference on Learning Representations (ICLR 2025)*, 2025. doi:10.48550/arXiv.2410.20285.
- 3 Berk Atıl, Sarp Aykent, Alexa Chittams, Lisheng Fu, Rebecca J. Passonneau, Evan Radcliffe, Guru Rajan Rajagopal, Adam Sloan, Tomasz Tudrej, Ferhan Ture, Zhe Wu, Lixinyu Xu, and Breck Baldwin. Non-determinism of “deterministic” LLM system settings in hosted environments. In *Proceedings of the 5th Workshop on Evaluation and Comparison of NLP Systems (Eval4NLP 2025)*, pages 135–148. Association for Computational Linguistics, 2025. doi:10.18653/v1/2025.eval4nlp-1.12.
- 4 Augment Code. How to run a multi-agent coding workspace. Vendor guide, 2026. Industry practitioner guide, accessed 2026-05-17. URL: <https://www.augmentcode.com/guides/how-to-run-a-multi-agent-coding-workspace>.
- 5 Augment Code. How to use git worktrees for parallel AI agent execution. Vendor guide, 2026. Industry practitioner guide, accessed 2026-05-17. URL: <https://www.augmentcode.com/guides/git-worktrees-parallel-ai-agent-execution>.
- 6 N. E. Breslow and N. E. Day. *Statistical Methods in Cancer Research, Volume I: The Analysis of Case-Control Studies*. International Agency for Research on Cancer, Lyon, France, 1980. IARC Scientific Publications No. 32.
- 7 Yuriy Brun, Reid Holmes, Michael D. Ernst, and David Notkin. Proactive detection of collaboration conflicts. In *Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering (ESEC/FSE '11)*, pages 168–178. ACM, 2011. doi:10.1145/2025113.2025139.
- 8 Silin Chen, Shaoxin Lin, Yuling Shi, Heng Lian, Xiaodong Gu, Longfei Yun, Dong Chen, Lin Cao, Jiyang Liu, Nu Xia, and Qianxiang Wang. SWE-Exp: Experience-driven software issue resolution. arXiv:2507.23361, 2025. doi:10.48550/arXiv.2507.23361.
- 9 Cursor. Cursor 2.0: New coding model and agent interface. Vendor release notes, October 2025. Product release notes, accessed 2026-05-17. URL: <https://cursor.com/changelog/2-0>.
- 10 DeepSeek-AI. DeepSeek-V4: Towards highly efficient million-token context intelligence. Technical report, DeepSeek-AI, April 2026. URL: <https://huggingface.co/deepseek-ai/DeepSeek-V4-Pro>.
- 11 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? arXiv:2509.16941, 2025. doi:10.48550/arXiv.2509.16941.

- 12 Thomas G. Dietterich. Approximate statistical tests for comparing supervised classification learning algorithms. *Neural Computation*, 10(7):1895–1923, 1998. doi:10.1162/089976698300017197.
- 13 Paul Dourish and Victoria Bellotti. Awareness and coordination in shared workspaces. In *Proceedings of the 1992 ACM Conference on Computer-Supported Cooperative Work (CSCW '92)*, pages 107–114. ACM, 1992. doi:10.1145/143457.143468.
- 14 Paul Gauthier. Aider: AI pair programming in your terminal. Software repository, 2024. URL: <https://github.com/Aider-AI/aider>.
- 15 Jiayi Geng and Graham Neubig. Effective strategies for asynchronous software engineering agents. arXiv:2603.21489, 2026. doi:10.48550/arXiv.2603.21489.
- 16 Sirui Hong, Mingchen Zhuge, Jiaqi 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. In *The Twelfth International Conference on Learning Representations (ICLR 2024)*, 2024. doi:10.48550/arXiv.2308.00352.
- 17 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? In *The Twelfth International Conference on Learning Representations (ICLR 2024)*, 2024. doi:10.48550/arXiv.2310.06770.
- 18 Daniël Lakens. Equivalence tests: A practical primer for *t* tests, correlations, and meta-analyses. *Social Psychological and Personality Science*, 8(4):355–362, 2017. doi:10.1177/1948550617697177.
- 19 Jiaye Lin, Yifu Guo, Yuzhen Han, Sen Hu, Ziyi Ni, Licheng Wang, Mingguang Chen, Hongzhang Liu, Ronghao Chen, Yangfan He, Daxin Jiang, Binxing Jiao, Chen Hu, and Huacan Wang. SE-Agent: Self-evolution trajectory optimization in multi-step reasoning with LLM-based agents. In *Advances in Neural Information Processing Systems 38 (NeurIPS 2025)*, 2025. doi:10.48550/arXiv.2508.02085.
- 20 Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegrefe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. Self-refine: Iterative refinement with self-feedback. In *Advances in Neural Information Processing Systems 36 (NeurIPS 2023)*, pages 46534–46594, 2023. doi:10.48550/arXiv.2303.17651.
- 21 Rian Melo, Pedro Simões, Rohit Gheyi, Marcelo d’Amorim, Márcio Ribeiro, Gustavo Soares, Eduardo Santana de Almeida, and Elvys Soares. Agentic LMs: Hunting down test smells. *IEEE Software*, 43(1):32–40, 2026. doi:10.1109/MS.2025.3621356.
- 22 MiniMax-AI. MiniMax-M2.7: Early echoes of self-evolution. Technical report, MiniMax-AI, March 2026. URL: <https://www.minimax.io/news/minimax-m27-en>.
- 23 Martin Mirchev, Ridwan Shariffdeen, Haifeng Ruan, Yuntong Zhang, and Abhik Roychoudhury. AutoCodeRover: Agentic program repair for SonarQube issues. In *Proceedings of the 34th ACM International Conference on the Foundations of Software Engineering (FSE Companion '26)*, pages 351–361. ACM, 2026. doi:10.1145/3803437.3805209.
- 24 Minh Huynh Nguyen, Thang Phan Chau, Phong X. Nguyen, and Nghi D. Q. Bui. AgileCoder: Dynamic collaborative agents for software development based on agile methodology. In *2025 IEEE/ACM Second International Conference on AI Foundation Models and Software Engineering (Forge)*, pages 156–167. IEEE Press, 2025. doi:10.1109/Forge66646.2025.00026.
- 25 OpenAI. Introducing SWE-bench Verified. Technical report, August 2024. Vendor announcement, accessed 2026-05-17. URL: <https://openai.com/index/introducing-swe-bench-verified/>.
- 26 OpenAI. Why SWE-bench Verified no longer measures frontier coding capabilities. OpenAI technical note, February 2026. Vendor announcement, accessed 2026-05-17. URL: <https://openai.com/index/why-we-no-longer-evaluate-swe-bench-verified/>.

- 27 Albert Örwall. Moatless Tools: A modular toolkit for AI-driven software development. Software repository, 2024. URL: <https://github.com/aorwall/moatless-tools>.
- 28 Shuyin Ouyang, Jie M. Zhang, Mark Harman, and Meng Wang. An empirical study of the non-determinism of ChatGPT in code generation. *ACM Transactions on Software Engineering and Methodology*, 34(2):42:1–42:28, 2025. doi:10.1145/3697010.
- 29 Hasen Özyatürk and Feza Buzluca. PASC – Peer-Aware Action-Space Coordination. Software, version 1.0.0., swbId: swb:1:dir:f44739f72961a5fb3ed2c5f4c8b2d88faf946f81 (visited on 2026-09-18). URL: <https://github.com/hasenoz/PASC>, doi:10.4230/artifacts.28067.
- 30 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. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 15174–15186. Association for Computational Linguistics, 2024. doi:10.18653/v1/2024.acl-long.810.
- 31 Noah Shinn, Federico Cassano, Edward Berman, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems 36 (NeurIPS 2023)*, pages 8634–8652, 2023. doi:10.48550/arXiv.2303.11366.
- 32 The SWE-agent Project (Princeton NLP and Stanford NLP). mini-SWE-agent: A 100-line software engineering agent. Software repository, 2025. Open-source software, accessed 2026-05-17. URL: <https://github.com/SWE-agent/mini-swe-agent>.
- 33 The SWE-agent Project (Princeton NLP and Stanford NLP). SWE-agent: Repository notice on successor project. Software repository README, 2025. Open-source software, accessed 2026-05-17. URL: <https://github.com/SWE-agent/SWE-agent>.
- 34 Verdent AI. SWE-bench Verified technical report. Vendor technical report, 2025. Non-peer-reviewed vendor report, accessed 2026-05-17. URL: <https://www.verdent.ai/blog/swe-bench-verified-technical-report>.
- 35 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. In *The Thirteenth International Conference on Learning Representations (ICLR 2025)*, 2025. doi:10.48550/arXiv.2407.16741.
- 36 Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Ahmed Awadallah, Ryen W. White, Doug Burger, and Chi Wang. AutoGen: Enabling next-gen LLM applications via multi-agent conversation. In *The First Conference on Language Modeling (COLM 2024)*, 2024. doi:10.48550/arXiv.2308.08155.
- 37 Chunqiu Steven Xia, Yinlin Deng, Soren Dunn, and Lingming Zhang. Agentless: Demystifying LLM-based software engineering agents. In *Proceedings of the ACM on Software Engineering (FSE 2025)*, volume 2, pages 801–824. ACM, 2025. doi:10.1145/3715754.
- 38 John Yang, Carlos E. Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. SWE-agent: Agent-computer interfaces enable automated software engineering. In *Advances in Neural Information Processing Systems 37 (NeurIPS 2024)*, pages 50528–50652, 2024. doi:10.48550/arXiv.2405.15793.
- 39 Kexun Zhang, Weiran Yao, Zuxin Liu, Yihao Feng, Zhiwei Liu, Rithesh Murthy, Tian Lan, Lei Li, Renze Lou, Jiacheng Xu, Bo Pang, Yingbo Zhou, Shelby Heinecke, Silvio Savarese, Huan Wang, and Caiming Xiong. Diversity empowers intelligence: Integrating expertise of software engineering agents. arXiv:2408.07060, 2024. doi:10.48550/arXiv.2408.07060.
- 40 Yuntong Zhang, Haifeng Ruan, Zhiyu Fan, and Abhik Roychoudhury. AutoCodeRover: Autonomous program improvement. In *Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA 2024)*, pages 1592–1604. ACM, 2024. doi:10.1145/3650212.3680384.