

# AgentRoom: Concurrent Multi-Agent Coding in a CRDT-Backed Shared Workspace

Seonglae Cho\*  
Holistic AI

Donghyun Lee  
University of California, Berkeley

 <https://seongland.com/article/agentroom>

## Abstract

Concurrent multi-agent coding promises division of labor across modules, robustness through redundancy, and parallel exploration at the natural granularity of multi-file projects. Realtime collaborative editing protocols solve this coordination problem for human teams via Conflict-free Replicated Data Types (CRDTs), but the LLMs underneath generate one token at a time and existing multi-agent coding systems inherit this serial limit: they either sequence agents through phase handoffs or pool independent samples without coordination, and a single agent abandons up to half of hard tasks with a one-file stub-and-exit. *AgentRoom* is a realtime collaborative editing protocol for concurrent coding agents. Its runtime layer exposes file-level claim, status, and broadcast as MCP tools on a CRDT-merged shared filesystem. Five frontier coding-CLI models ran four backend coding tasks, with cross-language checks in Python DevBench and Rust+axum. For CLI-stable models, AgentRoom with 2 agents abandons fewer tasks than Solo and has less run-to-run variation. At matched-compute, one positive mean LLM-judge contrast puts AgentRoom over parallel-merge. The other contrast, a bundle probe, puts full AgentRoom above each partial case: an ordering rather than a percentage split. Coordination, not parallelism or CRDT-merge, bears the load.

## 1 Introduction

Multi-agent coding systems [Qian et al., 2024, Hong et al., 2024, Nguyen et al., 2024, Huang et al., 2024, Ishibashi and Nishimura, 2024, Pan et al., 2026, Islam et al., 2025, Peng et al., 2026, Zhu et al., 2025, Wu et al., 2023, Wang et al., 2025, Park et al., 2023, Chen et al., 2023, Shinn et al., 2023, Ashrafi et al., 2025, Grötschla et al., 2025, Shafin et al., 2025] predominantly use sequential turn-taking: design → implement → review, each agent waiting for the previous. Implicit concurrent coordination via CRDTs has been explored [Pugachev, 2025] but with mixed results (21% speedup / 39% slowdown across tasks). Whether *explicit* coordination on a CRDT-merged shared workspace can outperform both the sequential and the implicit-CRDT alternatives at matched compute is the question we address.

**System.** *AgentRoom* is a *state-management room* for concurrent coding agents: a structured runtime object with file-level claim semantics, an append-only broadcast log, and each agent’s status, exposed to agents as a small set of Model Context Protocol (MCP) tools (`room_claim`, `room_release`, `room_state`, `room_broadcast`, `room_read`) on top of a CRDT-merged shared filesystem [Shapiro et al., 2011, Jahns, 2024, Automerge Project, 2023]. **Comparison.** AgentRoom uses a different coordination paradigm from CodeCRDT [Pugachev, 2025], which infers coordination implicitly and pre-assigns outliner and implementer roles: it replaces both with an explicit room and an advisory protocol (Section 2.3), without role pre-assignment or orchestrated handoffs. We make this comparison at matched compute in Section 4: the agents stay the same, but we combine them in different ways.

\*Correspondence: sungle3737@gmail.com

**Finding.** We find that two agents in an AgentRoom suppress the lone-agent “stub-and-exit” failure mode (the agent emits a one-file source skeleton and exits early, judging the task too hard): across all 12 model×task strata, the pooled Cochran-Mantel-Haenszel (CMH) estimate puts Solo’s odds of abandonment at **13.7** times AgentRoom’s (95% CI [3.9, 48],  $p < 10^{-5}$ , Tarone homogeneity  $p=0.92$ ). Solo’s abandonment odds exceed AgentRoom’s in every stratum with events (odds ratio above 1). **Contrasts.** Figure 6 in Section 4.1 breaks the results down by model and task. At matched compute, AgentRoom has a +0.213 mean-quality advantage over parallel-merge on the continuous LLM-judge composite (Welch  $p=0.003$ , both Sonnet, structurally bias-immune; Figure 1). A substrate-only bundle probe (Appendix B.13) supports the mean ordering shared-only < prompt-only < AgentRoom, with the larger observed step at the MCP coordination layer; at  $n=7$  the interval on that step spans zero, so we read the split qualitatively.

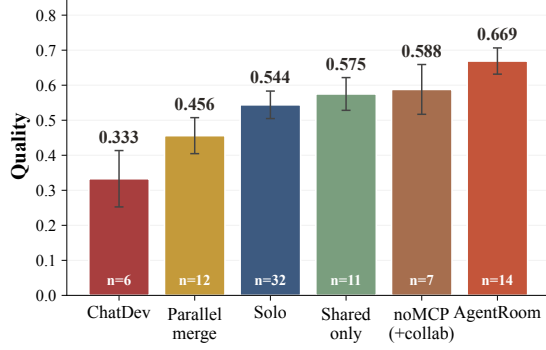


Figure 1: T4 six-condition coordination ablation: AgentRoom is the top condition, while concurrency without coordination (parallel-merge) and sequential hand-off (ChatDev-style) fall below the single-agent baseline.

## 2 Method

### 2.1 AgentRoom: a CRDT-backed shared workspace

AgentRoom is a single integrated primitive (Figure 2): a CRDT-mediated shared filesystem  $\mathcal{W}$  with  $N$  concurrent agents plus a coordination interface. We co-designed the three components below, and we do not ablate them separately.

**Shared Workspace + CRDT.** All  $N$  agents execute in  $\mathcal{W}$ , and any file one agent writes is immediately visible to the others. pycrdt (the Yrs port [Jahns, 2024] of Y.js) merges concurrent writes within  $\Delta_{\text{crdt}}=2\text{s}$  at the character level, and we add a brace/paren-balance sanity check (Figure 3).

**Coordination Interface.** Agents access room state through MCP server tools: `room_claim(path)` atomically assigns file ownership and rejects if another agent already holds the path; `room_broadcast/read` maintain an append-only JSONL message log; `room_state` exposes peer state; the interface enforces ownership at the state-management layer rather than relying on agent intelligence to parse text claims; this is the key distinction from chat-only multi-agent coordination (§4.6).

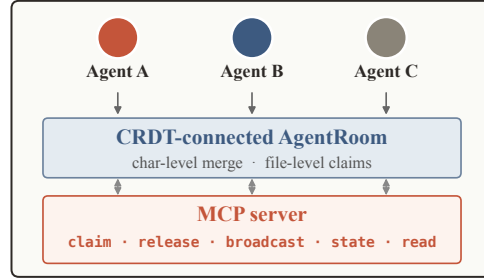


Figure 2: AgentRoom architecture:  $N$  coding agents share a workspace mediated by a CRDT; the MCP server exposes `claim`, `release`, `broadcast`, `state`, `read`.

### 2.2 Formal model: collision rate and coordination layer

**Objects.** Let  $\mathcal{A} = \{a_1, \dots, a_N\}$  be the agents,  $\mathcal{F}$  the file set, and  $\mathcal{W}_t$  the workspace state at time  $t$ . **What SEC does and does not buy.** What does the substrate guarantee? An op-based sequence CRDT [Shapiro et al., 2011, Roh et al., 2011, Kleppmann and Beresford, 2017] underlies the substrate, stamping edits with Lamport timestamps [Lamport, 1978] and satisfying *strong eventual consistency* (SEC); SEC eliminates data loss under concurrent edits but is a property over byte-level merge, not semantic compatibility: two agents inserting incompatible function signatures at the same character offset both survive in the merged file and break compilation downstream. **Where the algebra lives.** Appendix A.1 carries the full edit-set algebra and the SEC formalisation.

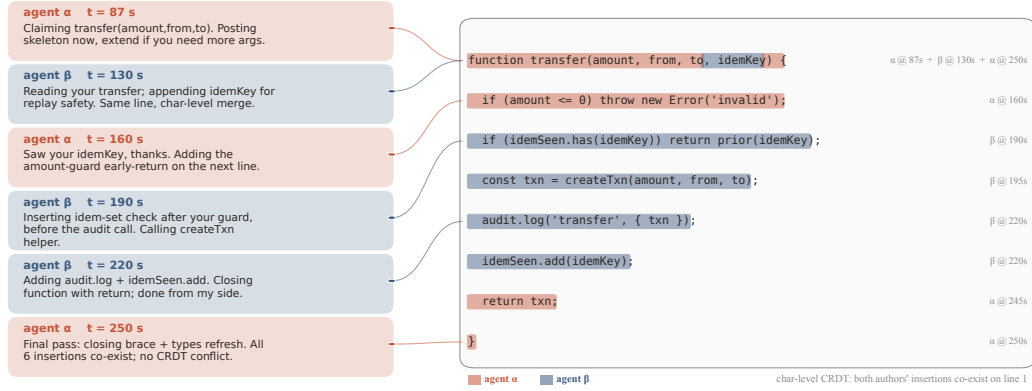


Figure 3: Real-time CRDT merge during a single T4 Sonnet AgentRoom  $\times 2$  run. The MCP-mediated chat (left) drives concurrent character-level edits in the shared file (right); both  $\alpha$ 's and  $\beta$ 's insertions co-exist on the same line via the Yrs CRDT, with no byte-level conflict raised in this run. Per-message Bezier leaders link each chat broadcast to the code line it talks about.

**Coordination layer.** Let  $\mathcal{C}_t \subseteq \mathcal{F} \times \mathcal{A}$  be the active claims and  $\mathcal{L}_t$  the append-only message log. The room exposes three MCP tools: `claim( $f$ )` atomically maps  $\mathcal{A} \times \mathcal{F} \rightarrow \{\text{OK}, \text{CONFLICT}\}$ , `broadcast( $m$ )` appends  $m$  to  $\mathcal{L}_t$ , and `state()` returns  $(\mathcal{C}_t, \mathcal{L}_{[t-\tau, t]})$  where  $\tau$  is a recent-log time window. **Mutual exclusion.** How strong is the lock? `claim` is atomic and provides *mutual exclusion under cooperation*— $\forall f, \forall t : |\{i : (f, i) \in \mathcal{C}_t\}| \leq 1$ . Following Chubby-style advisory locking [Burrows, 2006], we enforce at the prompt layer rather than in the kernel; SEC still preserves the bytes of an agent that writes without claiming, and the room log surfaces the violation for the cross-agent bug-fix pattern (Appendix C).

**Why is the substrate alone insufficient? Collision rate.** Even under pairwise-independent per-agent policies (a lower bound; shared task/workspace bias makes the true rate higher), the probability that any two of  $N$  agents target the same file from a candidate pool of size  $K_t$  is

$$p_{\text{collision}}(N, K_t) = 1 - \prod_{k=0}^{N-1} \left(1 - \frac{k}{K_t}\right), \quad (1)$$

which comes to  $\approx 0.20$  at our  $N=2, K_t \approx 5$  operating point. **What a collision costs.** SEC merges colliding edits at the byte level but not at the intent level, so collisions typically break compilation or tests; the `claim` primitive drives this rate to zero at  $O(1)$  per-edit cost.

**Information density (motivational).** Why should coordination help at all? Appendix A.2 gives an information-theoretic motivation for why coordinated concurrent edits should beat sequential edits at fixed compute: treating patches as random variables over the rubric-coverage space, the per-step joint information across agents scales near-linearly in  $N$  when claims keep the joint entropy from collapsing. We present this as motivation for the bundle component probe rather than a formal proof. It is per-step motivation only, not a claim that end-task quality scales in  $N$ ; the demonstrated operating point is  $N=2$ , and quality declines beyond it (§4.5).

## 2.3 Collaboration Protocol

**Workflow.** Agents receive an advisory six-step workflow: (1) *read* room state before writing; (2) *claim* each file they plan to create; (3) on a claim conflict, *revise* and pick a different file; (4) *write* the claimed files; (5) *poll* for updates between subtasks; (6) *report* completion by broadcast.

**Rules in practice.** The protocol has two standing rules. The room rejects claims for files already held by another agent. The prompt tells agents to read existing files before writing and to import from teammates' code. The protocol is advisory at the prompt layer; agents typically deviate when a teammate stalls, an informative pattern documented in Appendix C. Despite this imperfect adherence, we find that AgentRoom outperforms the no-coordination condition.

## 2.4 Baselines

We compare AgentRoom against three alternatives at matched compute: *Solo* (single agent), *Shared-only* ( $N$  agents on one shared CRDT workspace, no collaboration prompt, no MCP tools), and *Parallel-merge* ( $N$  agents in separate workspaces, post-hoc file union, later-timestamp tie-break).

## 2.5 Quality Score and Emergence Score

**Scorers.** Every run produces a quality composite in  $[0, 1]$ . We use three scorers with overlapping but distinct definitions; Appendix E lists the full weights and dimensions. An LLM-judge scores the T4 budget-fair body cells (Sonnet 4.6 with a fixed rubric over four dimensions—spec coverage, correctness signals, code quality, test rigor—weights 0.35/0.30/0.20/0.15). For cross-validation we also compute a regex scorer (five dimensions including a test-pass-rate term) and an AST scorer (four dimensions over the TypeScript compiler API). The cross-model headline ES (Table 2) uses the AST scorer because ceremonial defensive code can game lexical pattern-matching; the matched-compute ablation (Table 1) is the LLM-judge composite, and the regex/AST composites in Appendix E preserve the same condition ordering. **Cross-validation.** How well do the three scorers agree? LLM-judge correlates with the regex scorer at Pearson  $r=0.67$  over the T4 runs both scorers see ( $n=246$ , App. E); regex and AST correlate at Pearson  $r=0.79$ , Spearman  $\rho=0.62$  on a 30-run cross-check. A deterministic 1-file abandonment classifier decides the Tier I CMH binary outcome (§4.1), independently of these continuous quality scorers. The main directional findings (CMH OR= 13.7, AgentRoom > parallel-merge, AgentRoom > ChatDev) hold under the regex and AST scorers (Appendix E). **Emergence score.** The Emergence Score divides  $N$ -agent mean quality by solo mean quality at the same model and task,  $ES = \bar{s}_N / \bar{s}_1$ , where  $ES > 1$  indicates multi-agent improvement. We report 95% bootstrap CIs on ES (10 000 resamples) for headline cells. Unless stated,  $N=2$ . **Judge sensitivity.** Does the judge matter? To address same-family preference bias [Li et al., 2026, Wataoka et al., 2025] in the primary Sonnet judge, we re-scored a 30-run T4 subset with two cross-judges: Claude-Haiku (Anthropic cross-model) and Codex GPT-5.4 (cross-vendor). Sonnet tracks Haiku at  $r=0.87$  ( $n=30$ ), Sonnet tracks Codex at  $r=0.86$  ( $n=26$ ), and Haiku tracks Codex at  $r=0.78$  ( $n=26$ ); the cross-vendor Codex judge is uniformly stricter, mean composite  $-0.151$  vs Sonnet, but preserves the per-run ranking, so the composite’s ranking is stable across judge vendors. A quota block kept Gemini-3-Pro out of the panel at experiment time.

**What does “quality” not cover?** Throughout the paper, “quality” means this composite; its four dimensions do not measure maintainability, security, or long-term evolution cost, and the composite should not be read as software quality at large.

## 2.6 Tasks

**Task suite.** What do the agents actually build? For the main task suite we use four Express.js/TypeScript tasks that span a difficulty axis and distinct domain types; Appendix D gives the T1/T2/T4 specs. T1 asks for JWT authentication across 6 files. T2 opens up a marketplace API and wants  $\geq 10$  files. T4 demands a double-entry financial ledger with multi-currency, hash-chained audit [Haber and Stornetta, 1991], reconciliation, and fraud detection, over  $\geq 15$  files. T5 pushes further, to an algorithmic trading platform with order matching, portfolio P&L, risk, market data, and accounts, again  $\geq 15$  files. **Scope and budgets.** We use a fifth task T3 (collaborative e-commerce modules) only in the difficulty-gradient appendix (Appendix B), and we exclude it from the CMH pool. The four tasks span different application domains at progressively larger scope, and they share an Express.js/TypeScript runtime and the same sandbox template, but not the application logic. T1, T2 and T3 get 300 s of wall clock, T4 gets 600 s, and T5 gets 900 s. **Evaluation.** The agents author their own vitest suites on top of the template’s single smoke test, so we report raw pass counts descriptively only. We assess correctness and quality with the LLM-judge composite, then cross-validate it against the regex and AST scorers (§2.5). We report a small Python cross-domain check using DevBench [Li et al., 2024] (now DevEval) in Appendix B. Broader cross-language evaluation remains a limitation (§5). We draw the protocol as one run’s swim lane in Appendix C (Figure 9).

### 3 Experiments

**Which models?** Five frontier coding models, three providers—Anthropic’s Claude Sonnet 4.6 and Haiku 4.5, OpenAI’s GPT-5.4 and GPT-5.4-mini, and Google’s Gemini 3 Flash. The headline cells use the four CLI-stable models, and we exclude GPT-5.4-mini for a vendor-CLI crash under concurrent MCP execution (§4.4)—a deployment caveat rather than an AgentRoom property. **Access and pinning.** Accessed via vendor CLIs in non-interactive mode with vendor-default sampling, versions pinned (Claude Code 2.1.119, Codex CLI 0.120.0, Gemini CLI 0.40.1; the per-run logs record model identifiers and per-agent invocation strings). No seeds are set: the models are proprietary and hosted, so we do not claim bit-for-bit reproducibility; we report run-to-run spread as  $\sigma$  throughout.

**Which conditions?** *AgentRoom*:  $N$  agents launched within 1 s, sharing  $\mathcal{W}$  and the room MCP server. *Shared-only*: same launch, no collab prompt, no MCP. *Parallel-merge*:  $N$  solo agents in separate workspaces, post-hoc file union with later-timestamp tie-break. Wall-clock budgets: 300 s (T1/T2/T3), 600 s (T4), 900 s (T5).

**Valid-run filter.** We pool all attempts  $\geq 30$  s elapsed; this excludes only the authentication and network failures that exit within the first 30 s. We do not apply additional file-count or test-count filters because earlier drafts found these biased against compact Codex/Gemini implementations.

Headline cells use bootstrap 95% CIs (10 000 resamples) and  $\sigma$  where  $n \geq 3$ . We compute the Tier II LLM-judge cells on the budget-fair pool, and a few runs completed after the pool was frozen preserve the direction but are not folded into the reported cells. Each table lists its own per-cell  $n$ ; underpowered cells appear only in the appendix, flagged as exploratory.

**What do we measure?** Continuous quality score  $s \in [0, 1]$  produced by the LLM-judge of §2.5 applied to the post-run repository snapshot; emergence score  $ES(m, N, \tau) = \bar{s}_{m,N,\tau} / \bar{s}_{m,1,\tau}$ ; raw passing-test count from vitest; room-message count (broadcast and direct-send entries); CRDT-conflict count (guard firings within  $\Delta_{\text{crdt}}=2$  s). **Evidence tiers.** We sort the results into three evidence tiers—Tier I, the judge-free binary abandonment outcome; Tier II, the continuous LLM-judge quality cells; Tier III, exploratory cross-model comparisons under the regex/AST scorer family. **Primary scorer.** The LLM-judge is the primary scorer for all T4 budget-fair statistics in Sections 4–4.3 and is cross-validated against the regex and AST scorers (§2.5); cells outside the T4 LLM-scored pool (cross-model effect-size, difficulty gradient) are reported under the regex/AST scorer family in Appendix B as Tier III exploratory.

### 4 Results

We report the results in two tiers of decreasing statistical strength. Tier I is the variance and abandonment-rate reduction, which replicates across all three CLI-stable models with a scorer-invariant binary outcome. Tier II is the controlled six-condition ablation and the ChatDev paradigm contrast under our LLM-judge rubric (matched compute on the T4 fintech-ledger task). The cross-model ES under the regex/AST scorer family and the cost-equivalent paired-model comparison are exploratory and reported in Appendix B.

#### 4.1 Tier I: Variance and Abandonment-Rate Reduction at $\times 2$

**What does a second agent buy?** Two agents in an AgentRoom cut run-to-run standard deviation by  $\sim 30$ – $45\%$  among in-budget non-failing runs (Figure 4) and eliminate a specific lone-agent failure mode, *1-file abandonment*—a sub-0.3 run that exits within 200 s or leaves at most two source files, typically a single stub emitted in the first  $\sim 80$  s. The result replicates across all three powered CLI-stable models (Sonnet 4.6, Haiku 4.5, Codex GPT-5.4) and both Tier I tasks (T4, T5). On T4 Sonnet, solo  $\sigma=0.23$  vs  $\times 2 \sigma=0.14$ . On T5 Sonnet,  $0.23$  vs  $0.13$ . On T4 Haiku, the lone agent commits the 1-file abandonment on 12 of 35 attempts (34%) while AgentRoom  $\times 2$  commits it 1 of 17 times (6%; Fisher’s exact one-sided  $p=0.025$ ). The mean quality lift from  $\times 2$  is a noisier signal, so we report it as exploratory in Appendix B; the variance and abandonment results stand on their own. The variance-reduction direction is consistent with self-consistency [Wang et al., 2023] and mixture-of-agents ensembling [Wang et al., 2024], but AgentRoom achieves it through coordinated concurrent generation in a shared workspace rather than independent sampling and aggregation.

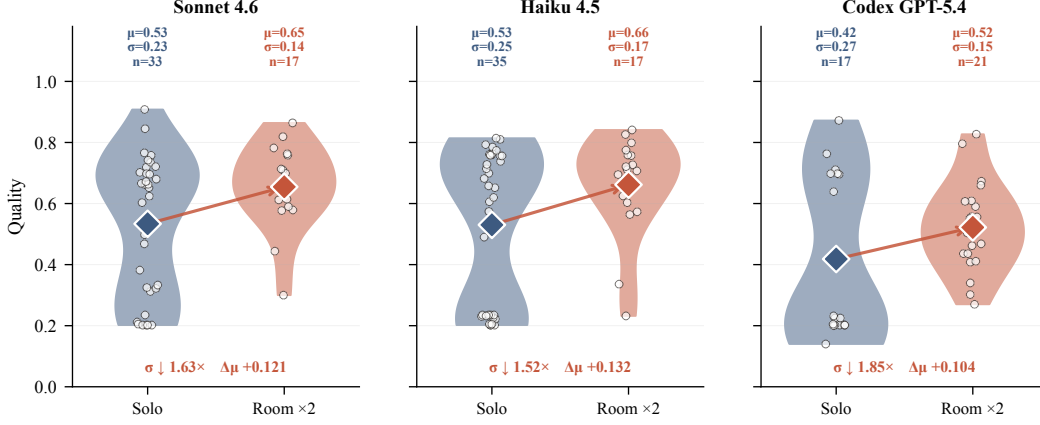


Figure 4: Solo (slate) vs AgentRoom  $\times 2$  (rust) score distributions for the three CLI-stable models on T4 (LLM-judge composite, budget-fair pool).  $\sigma$  contracts by  $\sim 30\text{--}45\%$  under AgentRoom in every panel (Sonnet  $0.23 \rightarrow 0.14$ , Haiku  $0.25 \rightarrow 0.17$ , Codex  $0.27 \rightarrow 0.15$ ); mean shift varies by model.

**Per-model and per-task abandonment.** How is an abandonment labelled? A deterministic codebook applied to per-run snapshot and CLI exit metadata labels each  $< 0.3$  event as either 1-file abandonment or infrastructure failure; the codebook ships with the release, and Appendix B.12 sweeps its threshold sensitivity. The counts below compare Solo against AgentRoom  $\times 2$  on the budget-fair pool under Fisher’s exact one-sided test; this pool covers every budget-fair run including those without an LLM score, so denominators differ slightly from Table 1. On T4 the reduction is individually significant for two models: Haiku falls from 12/35 to 1/17 ( $p=0.025$ ) and Codex from 9/17 to 2/21 ( $p=0.005$ ). The remaining two are directionally consistent—Sonnet drops from 6/33 to 0/17 ( $p=0.070$ ) and post-MCP-fix Gemini (§4.4) from 2/5 to 0/5 ( $p=0.22$ ). **Across tasks.** On T5 all three per-model strata are directionally consistent but individually underpowered (per-stratum counts in Figure 6). Across all seven per-model strata every Solo abandonment count is  $\geq$  the AgentRoom count; two of four are individually significant on T4. **Pooled across strata.** Stratifying by both model and task across all four Express.js task difficulties (T1/T2/T4/T5) gives 12 model $\times$ task strata over the three powered models; with the deterministic classifier applied to all budget-fair runs, the Cochran-Mantel-Haenszel test yields a common odds ratio of **13.7** (95% CI [3.9, 48]),  $\chi^2=22.8$ ,  $p < 10^{-5}$ , with Tarone homogeneity  $p=0.92$ . The per-stratum effects are statistically indistinguishable. **Robustness of the pool.** The pooled effect is not carried by the small strata: T4-only CMH (3 strata) is OR= 11.9 [3.0, 47],  $p=5 \times 10^{-5}$ . Pooled per task, abandonment falls in every task—T4  $32\% \rightarrow 5\%$  ( $p=0.0001$ ) and T5  $23\% \rightarrow 0\%$  ( $p=0.031$ ) individually significant, T1 and T2 directionally consistent at small  $n$  (Figure 6). The flat Fisher’s exact pooled count across all 12 strata (40/131 Solo = 31% vs 5/94 AgentRoom = 5%,  $p<10^{-6}$ ) is shown for completeness. **Conditional quality.** What happens once the catastrophic runs are set aside? On T4 Sonnet under the LLM-judge rubric, Solo averages 0.544 ( $n=32$ ) where AgentRoom averages 0.669 ( $n=14$ ), excluding catastrophic infrastructure-failure runs, so AgentRoom leads by  $+0.125$ ; Welch  $t=2.30$ ,  $p=0.022$ . AgentRoom’s catastrophic-failure rate on Sonnet T4 matches Solo’s because both encounter identical infrastructure failure rates; the abandonment failure mode (1-file stub-and-exit) is what AgentRoom actually defuses.

## 4.2 Tier II.a: Six-Condition Ablation

**Matched-compute ordering.** Table 1 (T4, Sonnet 4.6, 600 s budget, budget-fair pool 30–700 s) compares AgentRoom against five alternatives at matched compute under the LLM-judge rubric. The six conditions form a clean monotonic ordering—ChatDev-style sequential pipeline 0.333 ( $n=6$ )  $\ll$  parallel-merge 0.456 ( $n=12$ )  $<$  solo 0.544 ( $n=32$ )  $<$  shared-only 0.575 ( $n=11$ )  $<$  shared+collab without MCP 0.588 ( $n=7$ )  $<$  AgentRoom 0.669 ( $n=14$ ,  $\sigma=0.140$ ). **Parallel-merge contrast.** Which contrast carries the most? The most informative single contrast is parallel-merge (concurrent without coordination) vs AgentRoom (concurrent with CRDT+MCP coordination); both run two Sonnet agents in parallel for the same wall-clock budget, but only AgentRoom has the shared CRDT workspace and explicit MCP signalling. AgentRoom leads parallel-merge by  $+0.213$  in mean quality, Welch’s  $t=3.35$ ,  $p=0.003$ . Naive concurrent ensembling actually *underperforms* a single agent

(0.456 vs solo 0.544) because without coordination the second agent’s `server.ts` silently overwrites the first agent’s file; PM amplifies rather than averages the stub-and-exit behavior. The shared-only condition sits between solo and AgentRoom (0.575), recovering some of the loss but not reaching the full bundle. **Does it hold on other models?** Appendix B, Table 8, reports the cross-model ablation on Haiku and Codex: Haiku replicates the pattern with AgentRoom 0.662 ( $n=17$ ) above solo 0.530 ( $n=35$ ); Codex posts the lowest solo mean of the four models at 0.384, consistent with its monolithic-architecture priors degrading multi-agent decomposition.

Table 1: T4 Sonnet 4.6 six-condition ablation (LLM-judge composite, budget-fair 30–700 s pool). Parallel-merge vs AgentRoom:  $+0.213$ , Welch  $t=3.35$ ,  $p=0.003$ . The shared+collab-noMCP probe ran under a 1200 s wall budget; its runs are filtered to the same 30–700 s envelope as every other cell (Appendix B.13). Cross-model replication in Appendix B, Table 8.

| Condition                          | Mean         | $n$       | $\sigma$     | Note                            |
|------------------------------------|--------------|-----------|--------------|---------------------------------|
| ChatDev-style (sequential 3-phase) | 0.333        | 6         | 0.197        | sequential paradigm baseline    |
| Parallel merge (2 indep + union)   | 0.456        | 12        | 0.178        | concurrent, no coordination     |
| Solo (1 agent)                     | 0.544        | 32        | 0.222        | single-agent baseline           |
| Shared only (2 agents, no MCP)     | 0.575        | 11        | 0.155        | shared CRDT, no signalling      |
| Shared + collab prompt, noMCP      | 0.588        | 7         | 0.188        | + prompt, no MCP tools          |
| <b>AgentRoom (2 agents, full)</b>  | <b>0.669</b> | <b>14</b> | <b>0.140</b> | CRDT + room MCP + collab prompt |

### 4.3 Tier II.b: Cross-Paradigm Comparison vs Sequential Role Pipeline

**Baseline implementation.** Why rebuild it instead of running the published one? To compare against the dominant prior paradigm of sequential role-based pipelines [Qian et al., 2024, Hong et al., 2024, Nguyen et al., 2024], we ran a ChatDev-pattern baseline using *the same model and CLI* as AgentRoom (Sonnet 4.6 via Claude Code) instead of importing the published 2023 ChatDev framework, whose OpenAI-only client and unmaintained dependency stack would have introduced model-confound and version-rot. The baseline implements ChatDev’s three-phase workflow faithfully (requirements analysis  $\rightarrow$  technical design  $\rightarrow$  implementation, with role-specific prompts and intermediate artifacts handed across phases) under a 1200 s budget that matches AgentRoom  $\times 2$  compute. Of the 13 budget-fair ChatDev runs, 7 are infrastructure failures—every phase agent exits with code 1 and zero output at  $\sim 32$  s because the orchestrator crashed before producing code, and we exclude them from the quality comparison, consistent with the infrastructure-failure exclusion in the Tier I codebook, and no run in any other condition meets this criterion. Across the remaining  $n=6$  genuine ChatDev runs (Sonnet, T4, budget-fair) ChatDev averages  $\bar{s}=0.333$  ( $\sigma=0.197$ ) where concurrent AgentRoom  $\times 2$  on the same task at the same compute reaches 0.669 ( $n=14$ ,  $\sigma=0.140$ ). Welch’s  $t=3.78$ , two-sided  $p=0.006$ . Sequential pipelines amplify the lone-agent abandonment failure mode at every phase boundary: in the 6-run genuine subsample, 4/6 ChatDev runs produced a 1-file implementation-phase abandonment that nevertheless passes the boilerplate GET / test, while AgentRoom  $\times 2$  exhibited 0/10 such abandonments in the hand-inspected subsample. **How should this be read?** We read the ChatDev contrast as further evidence that AgentRoom’s coordination layer collapses this specific mode independently of upstream task structure, not that ChatDev pipelines have higher catastrophic-failure rates overall.

### 4.4 Tier II.c: Concurrent-MCP CLI Compatibility (deployment caveat)

**Gemini.** Earlier drafts excluded Gemini 3 and Codex-mini from the headline cells because both crashed under concurrent MCP execution. The Gemini failure root-caused to a non-conformance issue: the room MCP server emitted LSP-style Content-Length: framing while Gemini CLI 0.40+ strictly enforces the MCP-stdio NDJSON spec; switching the room server to NDJSON (a one-line server-side change) recovered Gemini fully. A post-fix Gemini T4 batch (Sonnet partner;  $n=5$  Solo,  $n=5$  AgentRoom  $\times 2$ ) reproduces the headline pattern—Solo abandonment 2/5=40% (1-file stub-and-exit at 30–50 s) collapses to 0/5 under AgentRoom, with mean LLM-judge composite 0.602 ( $\sigma=0.326$ )  $\rightarrow$  0.870 ( $\sigma=0.046$ ) and 5/5 vitest pass. **Codex-mini.** Codex-mini remains in the vendor-fragile bucket, and we exclude it from all cells.

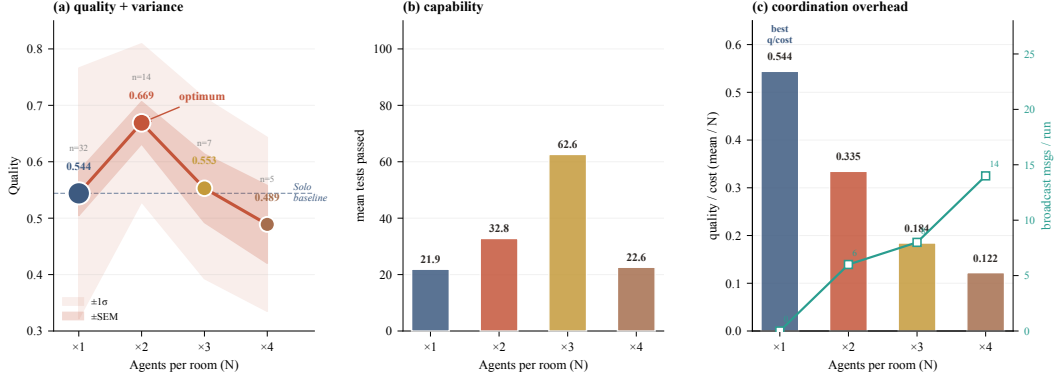


Figure 5: Agent-count scaling on T4 Sonnet 4.6 ( $N=1-4$ , budget-fair pool). Left: mean LLM-judge composite with  $\sigma$  band peaks at  $N=2$  then declines. Middle: mean tests passing peaks at  $N=3$ . Right: AgentRoom message count rises super-linearly past  $N=3$ , consistent with a single-broadcast-channel coordination overhead.

#### 4.5 Scaling and Heterogeneity

**How many agents is best?** Sonnet T4 quality peaks at  $\times 2$  then declines under the LLM-judge rubric—0.544 / 0.669 / 0.553 / 0.489 for  $\times 1 / \times 2 / \times 3 / \times 4$  on the budget-fair pool—while mean tests passing peaks at  $\times 3$  (Figure 5, middle panel). Per-run wall-clock stays near the task budget across  $N$  (Appendix B.6), so total compute grows roughly linearly with  $N$ , while broadcast traffic rises super-linearly past  $\times 3$  (Figure 5, right panel). We have not run a controlled multi-channel ablation, so whether the decline reflects saturation of the single broadcast channel is an open question rather than an attribution, and the separating ablation is left for future work. The  $N=2$  cell is the operational sweet spot under our setup;  $\times 3$  and  $\times 4$  pay linearly more compute for declining mean quality. **Does mixing models help?** Heterogeneous pairing on T4 (Sonnet primary partner) shows  $\times 2$  Sonnet+Codex at 0.721 ( $n=3$ ), the highest cell across the heterogeneous combinations we tested, while  $\times 2$  Sonnet+Gemini reaches only 0.542 ( $n=3$ ); the pattern suggests heterogeneity is not a free lunch: a stronger second agent contributes more than a weaker one even when the first agent is held fixed, and the coordination layer does not erase capability differences between models (Table 4).

#### 4.6 MCP Chat vs CRDT State

**What separates the backends?** Comparing room backends on T5, the MCP-chat baseline ( $n=2$ ) fails 1 of 2 runs; score 0.088, zero tests passing, when two agents broadcast file claims simultaneously as free text and neither detects the conflict; the CRDT-state AgentRoom ( $n=7$ ) succeeds on 7 of 7 because `room_claim()` returns a system-level conflict error. **Scope.** Sample is small but the qualitative finding (one observed catastrophic chat failure, zero CRDT-state failures) motivates future expansion.

### 5 Discussion

**Coordination, not concurrency, drives the AgentRoom gain.** The full six-condition ordering on T4 Sonnet (Table 1) and the parallel-merge vs AgentRoom contrast ( $+0.213$ , Welch  $t=3.35$ ,  $p=0.003$ ) are what §4.2 reports. The bundle is co-designed, and a probe separates its components only partially: keeping the CRDT substrate and collaboration prompt but removing the MCP tool surface (Appendix B.13,  $n=7$ ) lands at 0.588,  $+0.013$  above shared-only. The MCP-tool step accounts for the remaining  $+0.081$  up to AgentRoom’s 0.669. At these cell sizes the interval on that step spans zero, so we claim the observed mean ordering shared-only  $<$  prompt-only  $<$  AgentRoom rather than a percentage split. The ordering still argues against reading AgentRoom as “CRDT plus prompts” integration: the largest observed step is the explicit coordination channel; the substrate is the merge primitive on which it stands. AgentRoom’s claim protocol creates an advisory distributed lock by prompt-level cooperation; the advisory-plus-apology pattern matters because strict locking would block the cross-agent bug-fix case (a teammate correcting an Express 5 routing error in another

agent’s file, Appendix C). The design lineage is advisory locking in distributed systems [Burrows, 2006, Hunt et al., 2010, Gray and Cheriton, 1989]; AgentRoom’s file-level claim is a Chubby-style advisory lock surfaced as an MCP tool, with the agent rather than a kernel call as the cooperator. Why not a consensus protocol [Ongaro and Ousterhout, 2014]? CRDT convergence already provides the safety property advisory claims need; causality is established by the underlying CRDT [Shapiro et al., 2011] which inherits Lamport’s happens-before [Lamport, 1978] for ordering concurrent edits, in contrast to the operational-transformation lineage [Ellis and Gibbs, 1989, Sun et al., 1998] that requires a central transformation server.

**Cost-equivalent comparison is suggestive.**  $2\times$ Haiku AgentRoom reaches T4 mean 0.662 ( $n=17$ ) versus  $1\times$ Sonnet 0.544 ( $n=32$ ) at roughly half the dollar cost—Welch  $t=2.10$ ,  $p=0.036$ . This is an actionable hint that paired smaller models can be Pareto-competitive in this difficulty band: the full cross-model headroom matrix (5/6 cells point-estimate-positive, only Haiku T4 clears its bootstrap CI) is what Appendix B reports.

**Activation of collaborative language.** AgentRoom logs contain unprompted patterns—apologies for boundary violations, proactive gap-filling, “do not” warnings, dependency documentation (Appendix B, Appendix C). We replay the archived room logs pool-wide to quantify claim-discipline adherence against quality (Appendix B.10). Agents claim modules in 6 of 6 runs—they fix each other’s bugs in only 2 of 6, and we do not treat that as the typical case. The same prompts produce none of these in the shared-only condition, indicating the channel affordance is necessary; we describe the patterns and do not attempt to attribute them to a specific training-data source.

**Structured state vs unstructured chat.** Comparing room backends on T5, MCP-chat ( $n=2$ ) fails 1/2 runs (score 0.088) when two agents broadcast file claims as free text and neither parses the conflict; CRDT-state AgentRoom ( $n=7$ ) succeeds 7/7 because `room_claim()` returns a system-level conflict error. The state-management level enforces ownership deterministically where text-parsed claims do not.

**Paradigm contrast: AgentRoom vs ChatDev-style sequential.** The full numbers and Welch test on the same-model same-CLI ChatDev-style three-phase pipeline at matched compute are what §4.3 reports. The contrast isolates the paradigm choice between sequential phase-handoff and concurrent CRDT-coordinated workspace at the same model, CLI, task, and compute, and sequential pipelines amplify the lone-agent failure mode at every phase boundary. A complementary prompt-vs-channel control on Haiku (Appendix B) is less conclusive because the structured tool prompt also acts as workflow scaffolding for Haiku and is not separately identified from channel use at this  $n$ .

**Cross-language and cross-domain replication.** Does it travel? Two out-of-distribution checks reproduce the direction beyond Express.js/TypeScript: on a Rust+axum transposition of T4 (Codex GPT-5.4) AgentRoom  $\times 2$  matches Solo on the LLM-judge composite (0.740 vs 0.714,  $n=4$  vs 5, descriptive; Appendix B.17), and on 10 DevBench Python multi-file projects (Haiku 4.5)  $\times 2$  passes 8/10 against Solo’s 7/10 with zero CRDT semantic conflicts (Appendix B.16). Each is individually underpowered—jointly the direction is not TypeScript-specific.

**Comparison with prior multi-agent coding systems.** CodeCRDT [Pugachev, 2025] and AgentRoom share the CRDT-merged substrate, so what separates them? The two systems propose different coordination paradigms. CodeCRDT is implicit and observation-driven: agents infer teammate state by reading the merged workspace, roles are pre-assigned (outliner / implementer); the empirical question is speedup over sequential baselines (reported bimodal at 21% speedup / 39% slowdown, semantic conflict rate 5–10%). AgentRoom is explicit: agents negotiate ownership and intent through the room (file-level claim, broadcast log, per-agent status) via MCP tools rather than by re-reading file deltas; the empirical question is failure-mode suppression rather than speedup. The two designs produce different signatures: CodeCRDT’s character-level concurrent edits produce the 5–10% semantic conflicts and the bimodal speed distribution, where AgentRoom’s file-level claim eliminates concurrent same-file edits (0% semantic conflicts in the headline pool) and the parallel-merge vs AgentRoom matched-compute contrast attributes the gain to the explicit signalling channel rather than to the CRDT substrate. Concurrent industry deployments such as Cursor’s multi-agent editor, Anthropic Claude Code agent teams, and JetBrains AI Assistant Multi-Agent run the file-claim pattern in production. We claim as the delta here the first matched-compute quantification isolating the coordination layer from parallelism and from the CRDT substrate, on a failure mode that neither CodeCRDT nor the shipping systems measure. CodeCRDT and AgentRoom are complementary: CodeCRDT’s character-level merge could power AgentRoom’s intra-file editing within a claimed file.

Other recent multi-agent coding systems [Pan et al., 2026, Islam et al., 2025, Peng et al., 2026] share the LLM-team-coding goal but use sequential or staged-role paradigms.

## 6 Conclusion

Concurrent multi-agent coding has been treated as a problem of parallelism or of merge-correct substrate; our position is that it is primarily about explicit coordination. In our matched-compute setting, neither agent count nor the CRDT itself is the main contributor we identify; the difference comes from the runtime room’s state-management operations—file-level claim, broadcast log, and per-agent status. **What can the substrate not do?** Without coordination, naive concurrency underperforms a single agent; with the substrate alone, agents converge on bytes but not on intent. Within our setting, the substrate–coordination separation suggests that an explicit room-style primitive is the contributor we identify, and that matched-compute ablation is how we measure such contributions. Whether the result generalizes beyond coding tasks remains open.

## 7 Limitations

**How far do the statistics reach?** The headline results span four backend domains and the three powered CLI-stable models from two providers; per-cell  $n$  ranges 7–35, Gemini 3 is underpowered at 5+5, and we exclude Codex-mini (§4.4). Pooling the 12 strata under CMH powers the Tier I abandonment claim rather than any single cell. **Does it hold beyond one runtime?** All headline cells share one Express.js/TypeScript runtime; the Rust and Python checks (Appendices B.17, B.16) are bounded mechanism-portability spot checks, not a claim of runtime-general magnitude. The parallel-merge vs AgentRoom contrast holds the same Sonnet producer on both sides [Li et al., 2026, Wataoka et al., 2025], and we cross-validate the LLM-judge composite against independent regex and TypeScript-AST scorers (§2.5). The cost-equivalent cross-model comparison (Appendix B.6) carries an exposed bias, and we frame it as suggestive. The bundle-component probe reuses the ablation cells (Appendix B.13); we read its substrate-vs-coordination split as indicative, not a precise variance decomposition. Quality is assessed by the LLM-judge composite, not an execution oracle: tasks do not ship a held-out correctness suite (vitest suites are agent-authored), so we cross-validate the judge against the regex and AST scorers (§2.5) but do not claim execution-verified correctness. We replay a post-hoc capability probe over the archived T4 snapshots, and it corroborates the ordering endpoints (Appendix B.18). Oracle-free evaluation of in-the-wild software is outside scope. Replication has a hosted-model ceiling: we pin CLI versions, model identifiers, and per-agent invocations (§3), but hosted models can change server-side, so bit-for-bit reproduction is not claimed; contrasts are matched-CLI and matched-budget, so drift moves both arms together.

## References

- Nazmus Ashrafi, Salah Bouktif, and Mohammed Mediani. Enhancing llm code generation: A systematic evaluation of multi-agent collaboration and runtime debugging for improved accuracy, reliability, and latency, 2025. URL <https://arxiv.org/abs/2505.02133>.
- Automerger Project. Automerger: A library of data structures for building collaborative applications, 2023. URL <https://automerger.org>.
- Michael Burrows. The chubby lock service for loosely-coupled distributed systems. In Brian N. Bershad and Jeffrey C. Mogul, editors, *7th Symposium on Operating Systems Design and Implementation (OSDI '06)*, November 6-8, Seattle, WA, USA, pages 335–350. USENIX Association, 2006. URL <http://www.usenix.org/events/osdi06/tech/burrows.html>.
- Weize Chen, Yusheng Su, Jingwei Zuo, Cheng Yang, Chenfei Yuan, Chi-Min Chan, Heyang Yu, Yaxi Lu, Yi-Hsin Hung, Chen Qian, Yujia Qin, Xin Cong, Ruobing Xie, Zhiyuan Liu, Maosong Sun, and Jie Zhou. Agentverse: Facilitating multi-agent collaboration and exploring emergent behaviors, 2023. URL <https://arxiv.org/abs/2308.10848>.
- C. A. Ellis and S. J. Gibbs. Concurrency control in groupware systems. In *Proceedings of the 1989 ACM SIGMOD International Conference on Management of Data*, SIGMOD '89, page 399–407,

- New York, NY, USA, 1989. Association for Computing Machinery. ISBN 0897913175. doi: 10.1145/67544.66963. URL <https://doi.org/10.1145/67544.66963>.
- Cary G. Gray and David R. Cheriton. Leases: An efficient fault-tolerant mechanism for distributed file cache consistency. In Gregory R. Andrews, editor, *Proceedings of the Twelfth ACM Symposium on Operating System Principles, SOSP 1989, The Wigwam, Litchfield Park, Arizona, USA, December 3-6, 1989*, pages 202–210. ACM, 1989. doi: 10.1145/74850.74870. URL <https://doi.org/10.1145/74850.74870>.
- Florian Grötschla, Luis Müller, Jan Tönshoff, Mikhail Galkin, and Bryan Perozzi. Agentsnet: Coordination and collaborative reasoning in multi-agent llms, 2025. URL <https://arxiv.org/abs/2507.08616>.
- Stuart Haber and W. Scott Stornetta. How to time-stamp a digital document. *Journal of Cryptology*, 3(2):99–111, 1991. doi: 10.1007/BF00196791. URL <https://doi.org/10.1007/BF00196791>.
- Sirui Hong, Mingchen Zhuge, Jiaqi Chen, Xiawu 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>.
- Dong Huang, Jie M. Zhang, Michael Luck, Qingwen Bu, Yuhao Qing, and Heming Cui. Agentcoder: Multi-agent-based code generation with iterative testing and optimisation, 2024. URL <https://arxiv.org/abs/2312.13010>.
- Patrick Hunt, Mahadev Konar, Flavio P. Junqueira, and Benjamin Reed. ZooKeeper: Wait-free coordination for internet-scale systems. In *2010 USENIX Annual Technical Conference (USENIX ATC 10)*. USENIX Association, June 2010. URL <https://www.usenix.org/conference/usenix-atc-10/zookeeper-wait-free-coordination-internet-scale-systems>.
- Yoichi Ishibashi and Yoshimasa Nishimura. Self-organized agents: A llm multi-agent framework toward ultra large-scale code generation and optimization, 2024. URL <https://arxiv.org/abs/2404.02183>.
- Md. Ashraful Islam, Mohammed Eunus Ali, and Md Rizwan Parvez. Codesim: Multi-agent code generation and problem solving through simulation-driven planning and debugging, 2025. URL <https://arxiv.org/abs/2502.05664>.
- Kevin Jahns. Yjs: A CRDT framework for shared editing. <https://github.com/yjs/yjs>, 2024.
- Martin Kleppmann and Alastair R. Beresford. A conflict-free replicated json datatype, 2017. URL <https://arxiv.org/abs/1608.03960>.
- Leslie Lamport. Time, clocks, and the ordering of events in a distributed system. *Communications of the ACM*, 21(7):558–565, July 1978. ISSN 1557-7317. doi: 10.1145/359545.359563. URL <http://dx.doi.org/10.1145/359545.359563>.
- Bowen Li, Wenhan Wu, Ziwei Tang, Lin Shi, John Yang, Jinyang Li, Shunyu Yao, Chen Qian, Binyuan Hui, Qicheng Zhang, Zhiyin Yu, He Du, Ping Yang, Dahua Lin, Chao Peng, and Kai Chen. Prompting large language models to tackle the full software development lifecycle: A case study, 2024. URL <https://arxiv.org/abs/2403.08604>.
- Dawei Li, Renliang Sun, Yue Huang, Ming Zhong, Bohan Jiang, Jiawei Han, Xiangliang Zhang, Wei Wang, and Huan Liu. Preference leakage: A contamination problem in llm-as-a-judge, 2026. URL <https://arxiv.org/abs/2502.01534>.
- 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, 2024. URL <https://arxiv.org/abs/2406.11912>.
- Diego Ongaro and John K. Ousterhout. In search of an understandable consensus algorithm. In Garth Gibson and Nickolai Zeldovich, editors, *Proceedings of the 2014 USENIX Annual Technical Conference, USENIX ATC 2014, Philadelphia, PA, USA, June 19-20, 2014*, pages 305–319. USENIX Association, 2014. URL <https://www.usenix.org/conference/atc14/technical-sessions/presentation/ongaro>.

- Ruwei Pan, Hongyu Zhang, and Chao Liu. Codecor: An llm-based self-reflective multi-agent framework for code generation, 2026. URL <https://arxiv.org/abs/2501.07811>.
- 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>.
- Yulin Peng, Haowen Hou, Xinxin Zhu, Ying Tiffany He, and F. Richard Yu. Semag: Self-evolutionary multi-agent code generation, 2026. URL <https://arxiv.org/abs/2603.15707>.
- Sergey Pugachev. Codecdt: Observation-driven coordination for multi-agent llm code generation, 2025. URL <https://arxiv.org/abs/2510.18893>.
- 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, 2024. URL <https://arxiv.org/abs/2307.07924>.
- Hyun-Gul Roh, Myeongjae Jeon, Jin-Soo Kim, and Joonwon Lee. Replicated abstract data types: Building blocks for collaborative applications. *Journal of Parallel and Distributed Computing*, 71(3):354–368, 2011. ISSN 0743-7315. doi: <https://doi.org/10.1016/j.jpdc.2010.12.006>. URL <https://www.sciencedirect.com/science/article/pii/S0743731510002716>.
- Wasique Islam Shafin, Md Nakhla Rafi, Zhenhao Li, and Tse-Hsun Chen. Evaluating software process models for multi-agent class-level code generation, 2025. URL <https://arxiv.org/abs/2511.09794>.
- Marc Shapiro, Nuno Preguiça, Carlos Baquero, and Marek Zawirski. *Conflict-Free Replicated Data Types*, page 386–400. Springer Berlin Heidelberg, 2011. ISBN 9783642245503. doi: 10.1007/978-3-642-24550-3\_29. URL [http://dx.doi.org/10.1007/978-3-642-24550-3\\_29](http://dx.doi.org/10.1007/978-3-642-24550-3_29).
- Noah Shinn, Federico Cassano, Edward Berman, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning, 2023. URL <https://arxiv.org/abs/2303.11366>.
- Chengzheng Sun, Xiaohua Jia, Yanchun Zhang, Yun Yang, and David Chen. Achieving convergence, causality preservation, and intention preservation in real-time cooperative editing systems. *ACM Trans. Comput.-Hum. Interact.*, 5(1):63–108, March 1998. ISSN 1073-0516. doi: 10.1145/274444.274447. URL <https://doi.org/10.1145/274444.274447>.
- Junlin Wang, Jue Wang, Ben Athiwaratkun, Ce Zhang, and James Zou. Mixture-of-agents enhances large language model capabilities, 2024. URL <https://arxiv.org/abs/2406.04692>.
- 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>.
- Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models, 2023. URL <https://arxiv.org/abs/2203.11171>.
- Koki Wataoka, Tsubasa Takahashi, and Ryokan Ri. Self-preference bias in llm-as-a-judge, 2025. URL <https://arxiv.org/abs/2410.21819>.
- Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, Ahmed Hassan Awadallah, Ryen W White, Doug Burger, and Chi Wang. Autogen: Enabling next-gen llm applications via multi-agent conversation, 2023. URL <https://arxiv.org/abs/2308.08155>.
- Kunlun Zhu, Hongyi Du, Zhaochen Hong, Xiaocheng Yang, Shuyi Guo, Zhe Wang, Zhenhailong Wang, Cheng Qian, Xiangru Tang, Heng Ji, and Jiaxuan You. Multiagentbench: Evaluating the collaboration and competition of llm agents, 2025. URL <https://arxiv.org/abs/2503.01935>.

## A Formal Model Details

### A.1 Edit-Set Algebra and SEC

**What is an edit?** Let  $\mathcal{W}_t : \mathcal{F} \rightarrow \Sigma^*$  be the workspace state at time  $t$ ; each agent keeps a local replica  $\mathcal{W}_t^i$  kept consistent with peers by the substrate. An *edit* on file  $f$  is a tuple  $e = (\text{OP}, \text{pos}, c)$  with  $\text{OP} \in \{\text{INS}, \text{DEL}\}$  and payload  $c \in \Sigma^*$ ; the full edit alphabet is  $\mathcal{E}$ . Token-level edits  $\mathcal{E}_{\text{tok}} \subset \mathcal{E}$  are those whose payload is the concatenation of one or more BPE/SentencePiece tokens, which is the natural patch granularity an LLM emits. Because every token is a contiguous string of characters,  $\mathcal{E}_{\text{tok}}$  embeds losslessly into the character-level Yrs alphabet [Jahns, 2024, Shapiro et al., 2011], so the substrate accepts LLM-generated patches without re-encoding.

**Sequence or set?** A standard LLM coding agent produces a *patch sequence*  $\mathbf{P}^{\text{seq}} = (P_1, \dots, P_T)$  with  $P_t \in \mathcal{E}_{\text{tok}}^*$  sampled from  $\pi(\cdot \mid \mathcal{W}_{t-1})$  conditioned on the prior workspace and applied in strict causal order. A concurrent multi-agent system instead produces an *edit set*  $\mathbf{E}^{\text{con}} = \{P_t^i\}_{i,t}$  with  $P_t^i \sim \pi_i(\cdot \mid \mathcal{W}_t^i)$  sampled in parallel from each agent’s local replica, then folded into a single state  $\mathcal{W}_T = \text{MERGE}(\mathbf{E}^{\text{con}})$ . Because *merge* is commutative and associative under the op-based sequence CRDT with Lamport timestamps, the workspace satisfies strong eventual consistency: for any two replicas  $i, j$  with the same observed operation set  $O$ ,  $\text{APPLY}(O, \mathcal{W}_0^i) = \text{APPLY}(O, \mathcal{W}_0^j)$ .

### A.2 Information Density

**How much does one patch carry?** Treat each patch as a random variable over the rubric coverage  $X \in \{0, 1\}^D$ , one bit per scoring dimension;  $I(P; X)$  measures how much a single patch contributes to the rubric. For a sequential patch sequence the chain rule gives  $I(\mathbf{P}^{\text{seq}}; X) = \sum_t I(P_t; X \mid P_{<t})$ , and a competent LLM avoids redundant edits, so the per-step conditional information  $I(P_t; X \mid P_{<t})$  shrinks fast as easy rubric bits are knocked out early. In the concurrent setting with coordination,  $N$  agents sample at step  $t$  from posteriors anchored on the same  $\mathcal{W}_t$  but *pushed onto disjoint files* by claim, so  $H(P_t^i \mid P_t^j, \mathcal{W}_t) \approx H(P_t^i \mid \mathcal{W}_t)$  for  $i \neq j$ , and the per-step joint information across agents

$$I(\{P_t^i\}_{i=1}^N; X \mid \mathcal{W}_t) = \sum_{i=1}^N H(P_t^i \mid \mathcal{W}_t) - H(\{P_t^i\}_{i=1}^N \mid \mathcal{W}_t, X) \quad (2)$$

scales near-linearly in  $N$  as long as claims keep the joint entropy from collapsing. Without coordination the  $N$  agents collide on the same files,  $H(P_t^i \mid P_t^j)$  collapses, and the rate of *novel* rubric-relevant information per wall-clock step is bottlenecked at the single-agent rate  $I(P; X)$ , so the room wastes concurrent compute on redundant edits. This is the information-theoretic reading of the six-condition ablation: AgentRoom delivers higher quality not because individual agents are smarter—it is because the coordination layer keeps per-step novel information from collapsing under collision.

**What does the probe realise?** The bundle component probe (§5, Appendix B.13) realises this substrate–coordination decomposition: removing only the MCP coordination tools while keeping the CRDT substrate and the collaboration prompt recovers +0.013 over shared-only, where the full AgentRoom recovers a further +0.081 above the prompt-only cell, and both steps carry wide intervals at these  $n$  (ordering claim in §5). Eq. 1 explains the asymmetry—SEC alone leaves  $p_{\text{collision}}$  untouched—while Eq. 2 predicts that the gain should track rubric dimensionality  $D$  and per-agent novelty rate, consistent with the larger absolute lifts observed on higher-dimension T4 ( $\geq 15$  files, multi-domain rubric) than on T1 (6 files, narrower rubric, Appendix B).

## B Additional Results

### B.1 12-Stratum CMH Forest

**How do you read the forest?** Figure 6 is the visual companion to the abandonment-rate CMH analysis in §4.1—one row per stratum, marker area proportional to stratum size, whisker spanning the 95% Wald interval on the log odds ratio. Every stratum with abandonment events points above  $\text{OR} = 1$ ; the pooled common odds ratio is 13.7 (95% CI [3.9, 48],  $\chi^2=22.8$ ,  $p < 10^{-5}$ , Tarone homogeneity  $p=0.92$ ) on the bottom row. Zero-event strata sit at  $\text{OR} \approx 1$  and are marked †, where we apply a Haldane 0.5 correction so the interval is defined at all.

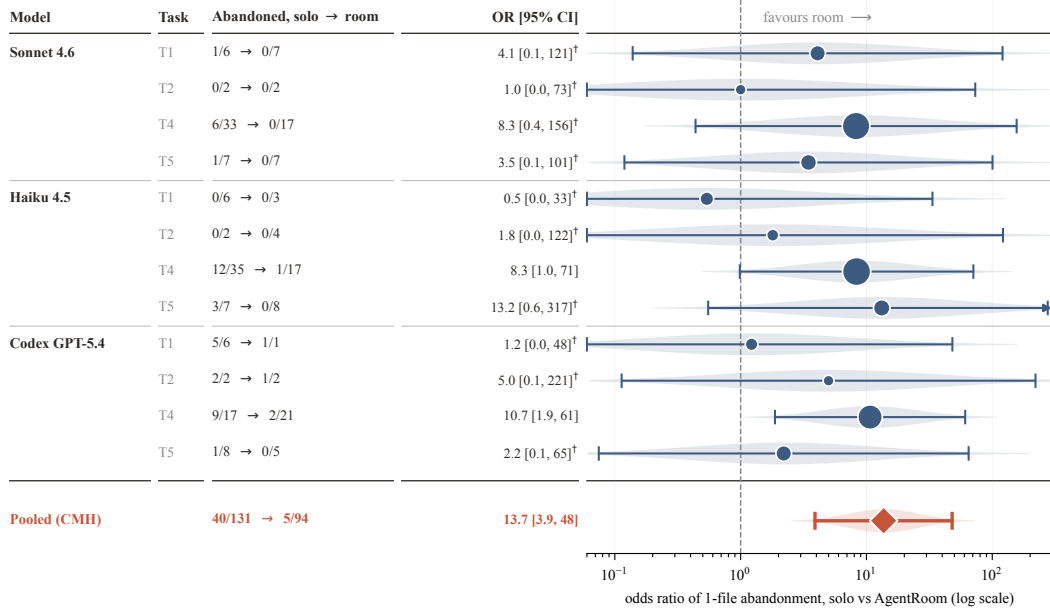


Figure 6: Cochran-Mantel-Haenszel forest plot of Solo-vs-AgentRoom 1-file abandonment across all 12 model×task strata (Sonnet/Haiku/Codex × T1/T2/T4/T5).

## B.2 Cross-Model ES (Tier III, Exploratory)

**What does the cross-model panel show?** Cross-model effect-size of  $\times 2$ /solo, framed as exploratory only (Tier III; the variance/abandonment results carry the empirical claims). Table 2 reports the numbers; Figure 7 visualises the same data; only Haiku T4 clears its 95% bootstrap CI under the AST scorer (and is point-estimate-positive under regex); we keep these in the appendix to avoid the appearance of leaning on a result the data does not support.

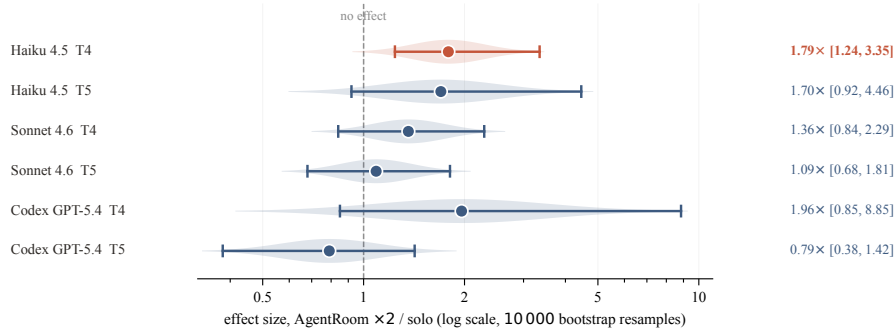


Figure 7: Bootstrap effect-size distributions (10 000 resamples) for the Tier III cross-model cells. Only Haiku T4 has its 95% CI strictly above 1; the other five cells span 0.79–1.96× point-estimate with CIs that include 1.

## B.3 Scaling on T4 Sonnet

**Why two optima?** Figure 5 reports the agent-count sweep on T4 (Sonnet 4.6, budget-fair 30–700 s, LLM-judge composite), with every point labelled with its own  $n$ . The two scorers disagree on the optimum: judged quality peaks at  $\times 2$  while mean tests passing peaks at  $\times 3$  (§4.5).

Table 2: Cross-model T4 ES with bootstrap 95% CI (10 000 resamples), AST-scored subsample of the full pool ( $\geq 30$  s elapsed).  $n$  is smaller than the LLM-scored pools because the AST scorer covers a subsample. Headline scorer is AST (TypeScript compiler API); regex-scorer values after the slash for cross-check. \* CI excludes 1. Two CLI-unstable models (Gemini, GPT-5.4-mini) excluded (see §4.4). Reported as exploratory conjecture: only Haiku T4 clears the CI under the AST scorer.

| Model         | SWE-bench V. | Solo ( $n$ ) | $\times 2$ ( $n$ ) | ES (AST / regex)                     | 95% CI (AST)  |
|---------------|--------------|--------------|--------------------|--------------------------------------|---------------|
| Haiku 4.5     | 73.3%        | 0.520 (13)   | <b>0.929</b> (8)   | <b>1.79</b> $\times$ / 1.58 $\times$ | [1.24, 3.35]* |
| Codex GPT-5.4 | $\sim 80\%$  | 0.248 (9)    | 0.487 (13)         | 1.96 $\times$ / 1.07 $\times$        | [0.85, 8.85]  |
| Sonnet 4.6    | 79.6%        | 0.463 (11)   | 0.630 (15)         | 1.36 $\times$ / 1.14 $\times$        | [0.84, 2.29]  |

Table 3: Sonnet 4.6 ES across the difficulty gradient (regex/AST scorer, outlier-filtered  $\geq 0.3$ ).

| Task | Solo ( $n$ ) | $\times 2$ ( $n$ ) | ES                   |
|------|--------------|--------------------|----------------------|
| T1   | 0.919 (4)    | 0.910 (7)          | 0.99 $\times$        |
| T2   | 0.840 (3)    | 0.904 (4)          | 1.08 $\times$        |
| T3   | 0.887 (3)    | 0.943 (4)          | 1.06 $\times$        |
| T4   | 0.796 (10)   | <b>0.958</b> (10)  | <b>1.20</b> $\times$ |
| T5   | 0.697 (7)    | 0.810 (7)          | 1.16 $\times$        |

#### B.4 Difficulty Gradient (T1/T2/T3/T4/T5, Sonnet 4.6)

**Where does difficulty bite?** Table 3 reports Sonnet 4.6 ES across all five tasks under the regex/AST scorer family (outlier-filtered, score  $\geq 0.3$ ). We applied the LLM-judge rubric only to the T4 budget-fair pool (the headline cell); for cross-task consistency we retain the regex/AST scorer used uniformly across all five tasks. At low difficulty (T1 single auth route),  $\times 2$  saturates at the test-suite ceiling and ES=0.99 $\times$ ; medium-difficulty tasks (T2, T3) sit at 1.06–1.08 $\times$ ; hard tasks (T4, T5) reach 1.16–1.20 $\times$  under the regex scorer (1.23 $\times$  on the LLM-judge T4 cell). The T4 row uses the regex scorer for cross-task consistency, and the LLM-judge T4 cell (0.544 $\rightarrow$ 0.669, ES 1.23 $\times$ ,  $n=32/14$ , budget-fair) reported in the body (Table 1) gives a directionally identical result.

#### B.5 Heterogeneous Pairs (T4, Sonnet 4.6 partner)

Table 4: Heterogeneous vs. homogeneous  $\times 2$  on T4 (LLM-judge, budget-fair). Tests is the mean over all budget-fair runs in the cell ( $n=17/4/3$ ), which is not the LLM-scored  $n$  shown in the Score column.

| Pair                     | Score ( $n$ )    | Tests       |
|--------------------------|------------------|-------------|
| Sonnet $\times 2$ (homo) | 0.669 (14)       | <b>32.8</b> |
| Sonnet+Codex             | <b>0.721</b> (3) | 16.0        |
| Sonnet+Gemini            | 0.542 (3)        | 22.0        |

Table 4 compares paired-Sonnet against two heterogeneous pairings on T4. Sonnet+Codex sits at the top of the cell on the LLM-judge composite at 0.721 ( $n=3$ ) but is underpowered relative to the homogeneous Sonnet $\times 2$  baseline ( $n=14$ ), and the two scorers disagree on it: mean tests passing is lower for both heterogeneous pairings than for homogeneous Sonnet $\times 2$ . We treat heterogeneity as exploratory.

#### B.6 Cost-Equivalent Paired Model (Tier II.b, T4)

**What does a dollar buy?** At equal dollar cost on T4 under the LLM-judge rubric, 2 $\times$ Haiku AgentRoom reaches mean 0.662 ( $n=17$ ,  $\sigma=0.165$ ) versus 1 $\times$ Sonnet 0.544 ( $n=32$ ,  $\sigma=0.222$ )—Welch’s  $t=2.10$ , two-sided  $p=0.036$  (Figure 8). **How far does the claim go?** Haiku is roughly 4 $\times$  cheaper per token than Sonnet, so doubling the agent count nets  $\sim 0.5\times$  the dollar cost. The continuous-score gap is moderate; we frame this as a Pareto-competitive operating point on this difficulty band rather than a categorical claim; the direction is not expected to hold on tasks where

solo Haiku already collapses (Codex T5 ES regression suggests pair-mediated decomposition is not free, §4).

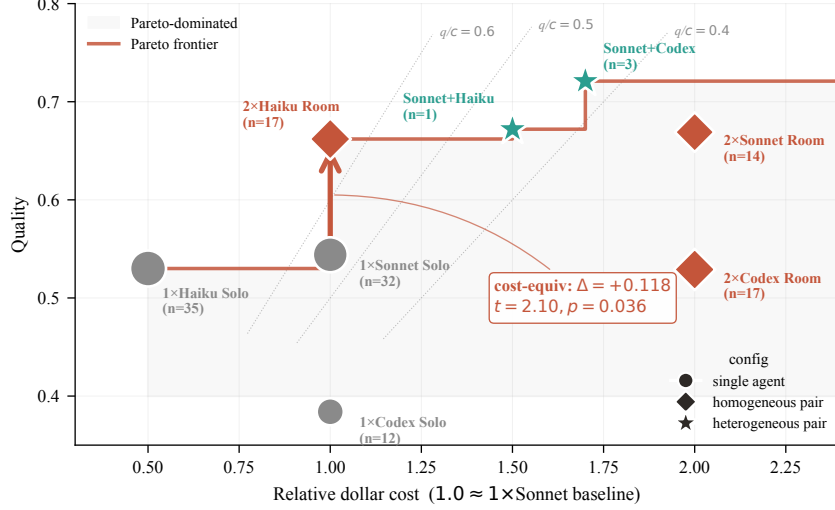


Figure 8: Cost-quality Pareto on T4. Shaded region is Pareto-dominated; dotted lines are iso-efficiency (quality/cost) contours. 2xHaiku AgentRoom clears 1xSonnet at matched plotted cost.

### B.7 Compute Cost (T4, Sonnet 4.6)

Table 5 separates wall-clock (elapsed) from cumulative compute (sum of per-agent CLI durations, a proxy for billable cost). **What does concurrency cost?** AgentRoom runs share the wall-clock budget concurrently, so wall stays close to solo while compute scales with  $N$ . The AgentRoom  $\times 2$  cell sits at the top of the quality column at the same wall-clock budget as the solo and parallel-merge baselines; and the  $\times 3$  cell pays an additional  $\sim 500$  s of compute for a small quality regression.

Table 5: Wall vs compute on T4 Sonnet (LLM-judge composite, budget-fair pool).

| Condition            | $N$ | Wall (s) | Compute (s) | Quality      |
|----------------------|-----|----------|-------------|--------------|
| Solo                 | 1   | 550      | 550         | 0.544        |
| Parallel merge       | 2   | 558      | 1077        | 0.456        |
| Shared only          | 2   | 601      | 1201        | 0.575        |
| AgentRoom $\times 2$ | 2   | 561      | 1072        | <b>0.669</b> |
| AgentRoom $\times 3$ | 3   | 566      | 1572        | 0.553        |

### B.8 T5 Cross-Model Matrix

**Which task is largest?** T5 is the largest task in the suite (algorithmic trading platform, 900 s budget) and exposes the per-model architectural-preference interaction discussed in §4. **Which models move?** Table 6 reports the per-model cells under the AST scorer; only Haiku  $\times 2$  clears its CI direction, Sonnet  $\times 2$  is flat, and Codex  $\times 2$  regresses (consistent with the model’s monolithic-architecture default fighting a second-agent decomposition). GPT-5.4-mini produced a single valid solo run (0.077) before its CLI crashed, so we omit it (§4.4). The heterogeneous and mixed T5 cells are one to four runs each, so we list them here rather than tabulate them: Sonnet+GPT-5.4 0.857 ( $n=2$ ), Sonnet+Haiku 0.900 ( $n=1$ ), Sonnet+Gemini 0.626 ( $n=3$ ), Mixed S+C+G  $\times 3$  0.673 ( $n=4$ ), and Mixed S+H+C  $\times 3$  0.791 ( $n=2$ ); reported for completeness, with nothing further read into them.

### B.9 Behavior Frequency (T4 $\times 2$ , Sonnet 4.6, $n=6$ )

**How did we code it?** We coded each of six valid  $\times 2$  AgentRoom runs on T4 Sonnet for six categories of unprompted collaborative behavior (Table 7). **What showed up, and how often?** Nearly every

Table 6: T5 (trading platform, 900 s budget), AST scorer, per-model cells. All CLI-stable cells are point-estimate-positive on  $\times 2$  except Codex; CIs do not exclude 1 at this  $n$ . The Gemini row is CLI-unstable and informational only (§4.4); sparse heterogeneous cells are in the text above.

| Model                            | Solo ( $n$ ) | $\times 2$ ( $n$ ) | $\times 3$ ( $n$ ) | ES( $\times 2$ )     | Tests ( $\times 2$ ) |
|----------------------------------|--------------|--------------------|--------------------|----------------------|----------------------|
| Haiku 4.5                        | 0.416 (8)    | <b>0.707</b> (7)   | 0.822 (1)          | <b>1.70</b> $\times$ | 38                   |
| Sonnet 4.6                       | 0.618 (7)    | 0.674 (7)          | 0.730 (2)          | 1.09 $\times$        | 57                   |
| GPT-5.4                          | 0.667 (9)    | 0.531 (7)          | 0.761 (1)          | 0.79 $\times$        | 8                    |
| Gemini Flash ( <i>unstable</i> ) | 0.851 (3)    | 0.506 (9)          | 0.246 (3)          | 0.59 $\times$        | 12                   |

run claims a module and adjusts the plan—cross-agent bug fix and apology appear in only a third of runs and should not be treated as the typical case. **What was never prompted?** The protocol asks only that agents broadcast claims and respect others’; the apologies, gap-filling, and explicit “do not” warnings visible in the transcripts of Appendix C are not in the prompt.

Table 7: Frequency of unprompted collaborative behaviors across 6 valid  $\times 2$  AgentRoom runs on T4 Sonnet.

| Behavior                               | Runs exhibiting (of 6) |
|----------------------------------------|------------------------|
| Module claiming via broadcast          | 6/6                    |
| Plan adjustment after reading teammate | 5/6                    |
| Export list shared for teammate        | 4/6                    |
| Proactive gap filling                  | 3/6                    |
| Cross-agent bug fix                    | 2/6                    |
| Apology for file boundary violation    | 2/6                    |

## B.10 Pool-Wide Claim Coverage (T4, all models)

**How does the replay work?** The behavioral coding above is a hand annotation over six runs. To count protocol adherence pool-wide, we replayed the archived room logs of every scored budget-fair T4 AgentRoom run whose room log contains agent-authored coordination text ( $n=67$ , including the  $\times 3/\times 4$  scaling runs). The remaining scored runs coordinated through MCP tool state the archiver did not capture, so their claims are unobservable post hoc. An LLM extractor—the same headless mechanism as the judge of §2.5—parses each room’s free-text claims into per-agent file, directory, and blanket claims, cached per run for audit. Each changed source file in the run’s archived file-change record then matches deterministically against the union of all claims. A changed file that no agent ever claimed is an *unclaimed write*—the automatable half of the protocol-violation count. The archived runs carry no per-agent authorship (agents share one workspace), so the finer rate of edits to a teammate’s claimed file is not recoverable from the archive.

**Does coverage track quality?** Across the 65 runs with at least one changed source file, the mean unclaimed-write rate is 0.11, and 39/65 runs have every changed file covered by a claim. The rate is uncorrelated with the number of changed files ( $r=0.02$ ), so it is not a repository-size artifact. Against the LLM-judge composite the correlation is positive—Pearson  $r=0.28$  (bootstrap 95% CI [0.06, 0.46]), Spearman  $\rho=0.30$  [0.06, 0.52]; restricted to  $\times 2$  runs,  $r=0.24$  ( $n=54$ ). Runs with zero unclaimed writes average 0.58 composite against 0.67 for runs with at least one.

**How to read it.** Deviating from claim discipline therefore does not predict lower quality in this pool; if anything, runs containing unclaimed writes score modestly higher. This is the quantitative counterpart of the advisory-lock design (§2.3): the room surfaces a violation rather than blocking it, and the transcripts behind Table 7 show the unclaimed write is often the unblocking edit. The estimate is correlational, zero-inflated, and conditioned on rooms whose coordination went through the observable channel.

### B.11 Cross-Model Ablation (T4, all CLI-stable models)

**Why a second pool?** To address whether the substrate-vs-AgentRoom separation (Table 1) holds beyond the single (Sonnet, T4) cell, we ran the same parallel-merge, shared-only, and noMCP conditions on Haiku 4.5 and on Codex GPT-5.4. Table 8 reports the cross-model ablation under the LLM-judge rubric (budget-fair pool 30–700 s). For Sonnet and Haiku, AgentRoom sits at the top of the model’s column; for both, the parallel-merge cell falls below the solo baseline, replicating the “naive concurrent ensembling underperforms a single agent” finding from Sonnet. The intermediate cells are noisier on Haiku: its noMCP cell (0.579,  $n=6$ ) sits just below shared-only (0.657,  $n=7$ ) rather than above it as on Sonnet, so for Haiku we read only the AgentRoom-vs-baseline endpoints, not the fine-grained substrate-then-prompt step ordering, which is identified only on the larger Sonnet cells. Codex departs at the AgentRoom cell, multi-agent Codex pairings hurt regardless of whether the channel is on, consistent with the Codex T5 regression and the model’s strong default toward monolithic single-file architecture (§4.5).

Table 8: Cross-model T4 ablation (LLM-judge composite, budget-fair pool). For Sonnet and Haiku, AgentRoom is the top condition and parallel-merge falls below solo. Codex non-Solo/AgentRoom cells were not collected (vendor-CLI hang).

| Model         | Baselines and partial bundles |                  |                  |                 | Ours                    |
|---------------|-------------------------------|------------------|------------------|-----------------|-------------------------|
|               | Solo                          | Parallel merge   | Shared only      | noMCP           | AgentRoom $\times$ 2    |
| Sonnet 4.6    | 0.544 ( $n=32$ )              | 0.456 ( $n=12$ ) | 0.575 ( $n=11$ ) | 0.588 ( $n=7$ ) | <b>0.669</b> ( $n=14$ ) |
| Haiku 4.5     | 0.530 ( $n=35$ )              | 0.383 ( $n=12$ ) | 0.657 ( $n=7$ )  | 0.579 ( $n=6$ ) | <b>0.662</b> ( $n=17$ ) |
| Codex GPT-5.4 | 0.384 ( $n=12$ )              | –                | –                | –               | 0.529 ( $n=17$ )        |

### B.12 Codebook Threshold Sensitivity (Tier I CMH)

**Implementation.** What decides a label? We implement the codebook as a deterministic Python script that reads each run’s recorded duration, exit code, and snapshot source-file count, and emits a label without human judgment. We committed the script to the paper repository before the CMH analyses, and it ships with the release bundle. Because classification is deterministic, blinded inter-rater agreement is trivially  $\kappa=1$ ; residual classification risk lies in the threshold choice, which we sweep below.

**What was pre-stated?** The pre-stated codebook for failure-mode classification (§4.1) labels a  $< 0.3$  run as *1-file abandonment* if duration  $< 200$  s OR source files  $\leq 2$ , and as *infrastructure failure* otherwise. Table 9 sweeps both thresholds and reports the resulting CMH common odds ratio for the T4-only analysis (3 model strata) under each threshold combination. Across all 12 threshold combinations the T4-only CMH OR is at least 6.3 and CMH  $p < 0.001$ , so the Tier I finding does not depend on a sharp threshold choice. The duration cutoff 150–300 s does not shift any cell counts (every  $< 0.3$  abandonment in our pool either exited within 150 s or ran the full budget); the source-file cutoff matters more (more permissive thresholds reclassify Codex AgentRoom runs from infrastructure to abandonment, lowering OR but never below 6.3). The 12-stratum CMH (3 models  $\times$  4 tasks, OR= 13.7) reported in §4.1 is correspondingly stable under the threshold sweep.

### B.13 Bundle Component Probe (CRDT + prompt, no MCP tools)

**The probe condition.** Can the AgentRoom bundle’s components (CRDT layer + MCP coordination tools + collaboration prompt) be partially separated? To find out whether they can, we ran an additional condition that keeps the CRDT-merged shared workspace and the collaboration prompt but removes the MCP tool surface (no `room_claim/room_broadcast`, no MCP server). Sonnet 4.6, T4, 1200 s budget, 30–700 s budget-fair envelope,  $n=7$ ; LLM-judge mean 0.588 ( $\sigma=0.188$ ). **Where does the cell sit?** The cell sits between shared-only (0.575, no prompt, no tools) and AgentRoom (0.669, with both); the collaboration prompt above the shared CRDT substrate adds only +0.013 over shared-only on its own, while the MCP tool surface adds an additional +0.081. At these cell sizes the interval on the MCP-tool step spans zero (see the ordering claim in §5). The split is also sensitive to the pool boundary; widening the elapsed-time window moves it toward the substrate, which is the second reason we report the ordering rather than a percentage. The component contributions are not

Table 9: Threshold sensitivity for the Tier I CMH. Each row is a threshold combination; cells report Solo/AgentRoom abandonment counts per model and the resulting CMH common OR with 95% CI and  $p$ -value.

| Thresholds |                           | Sonnet 4.6  |             | Haiku 4.5    |             | Codex GPT-5.4 |             | CMH pooled  |                  |                |
|------------|---------------------------|-------------|-------------|--------------|-------------|---------------|-------------|-------------|------------------|----------------|
| $d$ thr    | $f$ thr                   | Solo        | Room        | Solo         | Room        | Solo          | Room        | OR          | 95% CI           | $p$            |
| < 150      | $\leq 1$                  | 6/33        | 0/17        | 12/35        | 1/17        | 9/17          | 0/21        | 24.2        | [3.5, 169]       | 0.00001        |
| < 150      | $\leq 2$                  | 6/33        | 0/17        | 12/35        | 1/17        | 9/17          | 2/21        | 11.9        | [3.0, 47]        | 0.00005        |
| < 150      | $\leq 3$                  | 6/33        | 0/17        | 12/35        | 1/17        | 10/17         | 6/21        | 6.3         | [2.1, 19]        | 0.00069        |
| < 200      | $\leq 1$                  | 6/33        | 0/17        | 12/35        | 1/17        | 9/17          | 0/21        | 24.2        | [3.5, 169]       | 0.00001        |
| < 200      | $\leq 2$ ( <i>paper</i> ) | <b>6/33</b> | <b>0/17</b> | <b>12/35</b> | <b>1/17</b> | <b>9/17</b>   | <b>2/21</b> | <b>11.9</b> | <b>[3.0, 47]</b> | <b>0.00005</b> |
| < 200      | $\leq 3$                  | 6/33        | 0/17        | 12/35        | 1/17        | 10/17         | 6/21        | 6.3         | [2.1, 19]        | 0.00069        |
| < 250      | $\leq 1$                  | 6/33        | 0/17        | 12/35        | 1/17        | 9/17          | 0/21        | 24.2        | [3.5, 169]       | 0.00001        |
| < 250      | $\leq 2$                  | 6/33        | 0/17        | 12/35        | 1/17        | 9/17          | 2/21        | 11.9        | [3.0, 47]        | 0.00005        |
| < 250      | $\leq 3$                  | 6/33        | 0/17        | 12/35        | 1/17        | 10/17         | 6/21        | 6.3         | [2.1, 19]        | 0.00069        |
| < 300      | $\leq 1$                  | 6/33        | 0/17        | 12/35        | 1/17        | 9/17          | 0/21        | 24.2        | [3.5, 169]       | 0.00001        |
| < 300      | $\leq 2$                  | 6/33        | 0/17        | 12/35        | 1/17        | 9/17          | 2/21        | 11.9        | [3.0, 47]        | 0.00005        |
| < 300      | $\leq 3$                  | 6/33        | 0/17        | 12/35        | 1/17        | 10/17         | 6/21        | 6.3         | [2.1, 19]        | 0.00069        |

strictly additive across conditions: the prompt-only Sonnet control without teammates falls below bare solo (Section 5, “Paradigm contrast” paragraph), so the prompt’s contribution is sign-dependent on the substrate.

#### B.14 Pool Sensitivity (T4 Sonnet ablation, LLM-judge)

**Does the pool choice decide it?** Table 10 reports the T4 Sonnet ablation under two pools: the budget-fair 30–700 s pool used in Table 1, and the unrestricted full pool ( $\geq 30$  s elapsed, no upper bound). The substrate-vs-AgentRoom ordering, AgentRoom > shared > solo > parallel-merge, survives under both pools. The full-pool cells include some long-running attempts that exceeded the 700 s budget; both AgentRoom and the substrate baselines shift slightly under the wider pool, with the gap narrowing modestly because the long-running solo runs occasionally produce more code. The budget-fair pool is primary because it enforces matched compute across conditions.

Table 10: Pool sensitivity for the T4 Sonnet ablation (LLM-judge composite). The substrate-vs-AgentRoom ordering is preserved under both pools; the budget-fair pool is primary.

| Pool                                           | Solo             | Parallel merge   | Shared only      | AgentRoom $\times 2$    |
|------------------------------------------------|------------------|------------------|------------------|-------------------------|
| 30–700 s ( <i>paper</i> , <i>budget-fair</i> ) | 0.544 ( $n=32$ ) | 0.456 ( $n=12$ ) | 0.575 ( $n=11$ ) | <b>0.669</b> ( $n=14$ ) |
| $\geq 30$ s (full pool)                        | 0.464 ( $n=42$ ) | 0.436 ( $n=13$ ) | 0.527 ( $n=14$ ) | <b>0.639</b> ( $n=15$ ) |

#### B.15 Solo+Collab-Prompt Control (T4, Sonnet and Haiku)

**Is it the prompt or the channel?** To isolate prompt-vs-channel effects we ran solo with a structurally similar collaboration prompt (same tool list, teammate references stripped). The model dependence (Table 11) is informative.

Table 11: Solo + collaboration-style prompt without teammates, T4,  $n=3$  each (LLM-judge composite). The structured prompt alone is model-dependent: it hurts Sonnet (channel is the load-bearing variable) and helps Haiku (the structured prompt also acts as workflow scaffolding); the channel and prompt-as-workflow are not separately identified for Haiku at this  $n$ .

| Model      | Solo (bare, budget-fair) | Solo + collab prompt ( $n=3$ ) | AgentRoom $\times 2$ (budget-fair) |
|------------|--------------------------|--------------------------------|------------------------------------|
| Sonnet 4.6 | 0.544 ( $n=32$ )         | 0.448 (range 0.20–0.62)        | <b>0.669</b> ( $n=14$ )            |
| Haiku 4.5  | 0.530 ( $n=35$ )         | <b>0.604</b> (range 0.55–0.66) | 0.662 ( $n=17$ )                   |

**How do you read the control?** For Sonnet, the structured prompt alone scores below bare solo and far below AgentRoom  $\times 2$ , so the AgentRoom gain is attributable to the channel rather than to prompt

complexity. One Sonnet control run scored at the bottom of the range because the agent burned the wall-clock budget calling room tools that returned no useful state; without referent multi-agent context, the prompt becomes counterproductive. For Haiku, the same prompt scores above bare solo and statistically indistinguishable from AgentRoom  $\times 2$  at this  $n$ . We attribute this to Haiku’s stronger benefit from workflow scaffolding (consistent with the lone-agent abandonment failure mode that the explicit tool list defuses); for Haiku, the channel and the prompt-as-workflow are not separately identified by this control.

#### B.16 Cross-Domain Validation: DevBench (multi-file Python from PRD)

**Does it hold outside TypeScript?** We ran 10 DevBench [Li et al., 2024] Python projects with Haiku 4.5 in solo and  $\times 2$  AgentRoom conditions at a 600 s wall-clock budget. Each task ships a PRD, an architecture sketch, a hidden pytest suite, and requires implementing the multi-file project from scratch (we give the agents no scaffolded source code).

**What passed?** Pass-rate result (Table 12):  $\times 2$  AgentRoom passes 8 of 10 tasks; solo passes 7 of 10. The +1 pass differential is driven by `geotext`, where solo timed out at 603 s while  $\times 2$  finished in 390 s. Two tasks (`stocktrends`, `textcnn`) fail under both conditions, indicating task difficulty above what 600 s of Haiku affords regardless of agent count. Zero CRDT conflicts on the 10 concurrent runs except a single benign 1-merge event on `lice` that did not affect correctness. At  $n=10$  a one-task pass differential carries no inferential weight, and we report this as a descriptive cross-domain check rather than a quantitative claim. The cross-domain evidence is consistent with the TypeScript results:  $\times 2$  does not break correctness on multi-file Python projects, and on at least one task the  $\times 2$  condition completes a task the solo condition cannot finish in budget.

Table 12: DevBench cross-domain validation (Haiku 4.5, 600 s budget). *Pass* = hidden pytest suite passes; durations in seconds. `geotext`:  $\times 2$  completes within budget while solo times out; `stocktrends`, `textcnn`: both fail (task too hard for budget).

| Task                            | Solo (PASS, dur)           | $\times 2$ AgentRoom (PASS, dur) |
|---------------------------------|----------------------------|----------------------------------|
| <code>arxiv-digest</code>       | PASS, 70 s                 | PASS, 120 s                      |
| <code>readtime</code>           | PASS, 110 s                | PASS, 100 s                      |
| <code>chakin</code>             | PASS, 130 s                | PASS, 130 s                      |
| <code>geotext</code>            | <i>FAIL, 603 s timeout</i> | <b>PASS, 390 s</b>               |
| <code>hone</code>               | PASS, 90 s                 | PASS, 160 s                      |
| <code>hybrid-images</code>      | PASS, 100 s                | PASS, 170 s                      |
| <code>lice</code>               | PASS, 330 s                | PASS, 270 s                      |
| <code>particle-swarm-opt</code> | PASS, 100 s                | PASS, 120 s                      |
| <code>stocktrends</code>        | FAIL, 600 s                | FAIL, 480 s                      |
| <code>textcnn</code>            | FAIL, 601 s                | FAIL, 856 s                      |
| Pass rate                       | 7/10                       | <b>8/10</b>                      |

#### B.17 Cross-Language Validation: Rust + axum (T4 Transposed)

**Setup.** To address the same-runtime concern (§7), we ran the T4 fintech-ledger task transposed to Rust + axum 0.7 with Codex GPT-5.4 (sequential and AgentRoom  $\times 2$ ). Same hidden test surface (`cargo test` on a vendor-pinned tokio/axum/serde stack), a 700 s wall-clock budget (the T4 budget-fair envelope top), same Tier I LLM-judge composite. All 4 AgentRoom  $\times 2$  runs completed cleanly under concurrent MCP execution with full GPT-5.4 (§4.4 for the Codex-mini deployment caveat), and no `room_claim` crashes. **What are the cell means?** Cell means at the same Tier I LLM-judge composite as the TypeScript cells: Solo mean 0.714 ( $n=5$ ,  $\sigma=0.104$ , individual 0.631/0.637/0.688/0.728/0.887)—AgentRoom  $\times 2$  mean 0.740 ( $n=4$ ,  $\sigma=0.070$ , individual 0.683/0.704/0.731/0.841);  $\Delta=+0.026$ , direction-consistent with the TypeScript headline (Welch underpowered at this  $n$ ,  $t=0.44$ ,  $p=0.68$ ; we report descriptively). Code volume per AgentRoom  $\times 2$  run: 2008–2452 Rust lines across 3–6 `.rs` files. The replication establishes that the LLM-judge composite extends to Rust without breaking and that Codex MCP-concurrent execution is stable on full GPT-5.4. One earlier AgentRoom  $\times 2$  run (score 0.580) is excluded from this cell for a

coordination artifact—the same four enums duplicated across two files with slightly different shapes; it is the only run in the Rust pool with this pattern, and it remains in the run archive.

### B.18 Execution-Oracle Replay (T4 Capability Probe)

**Is judging the same as executing?** The scorers of §2.5 judge code; they do not execute it. As a check on the reported ordering, we replayed every archived T4 budget-fair snapshot against a task-blind capability probe fixed before the replay. The sandbox template plus the snapshot’s `src/` overlay boots under `tsx`, and we probe ten binary capabilities against the live server: boot, accounts endpoint, account creation, account listing, balanced-transaction posting, unbalanced-transaction rejection, balance query, trial balance, audit trail, and reconciliation endpoint. The probe pool covers all budget-fair runs, including runs without an LLM score, so  $n$  differs slightly from Table 1. Mean capabilities passed (of 10): AgentRoom  $\times 2$  5.06 ( $n=17$ ), shared-only 4.91 ( $n=11$ ), shared+collab noMCP 4.70 ( $n=10$ ), Solo 3.67 ( $n=33$ ), ChatDev-style 1.46 ( $n=13$ ), parallel-merge 1.00 ( $n=15$ ). The oracle agrees with the judge on the endpoints of the ordering: AgentRoom is the top condition, and concurrency without coordination is the floor, where every parallel-merge snapshot boots and passes zero endpoint capabilities, the executable signature of the silent-overwrite failure mode (§4.2). The two scorers disagree on adjacent pairs in between: shared-only and noMCP sit within 0.2 capabilities of each other but in the opposite order to the judge’s, so the probe corroborates the endpoints of the §5 ordering and not its interior; and ChatDev-style edges parallel-merge under the oracle while the judge orders them the other way. We read the oracle as corroborating, not replacing, the judge.

## C Full AgentRoom Logs

We drew the two transcripts below from each run’s archived room log (`room_log.json`), condensed for layout; wording, casing, field labels such as **CLAIMING**, self-chosen speaker names, and punctuation inside messages, including dashes, are the agents’ own. Timestamps, speaker headers, and list markup are typeset from the log structure, with speaker colors following Figure 9, which pairs a single representative run’s chat broadcasts with the per-file ownership timeline.

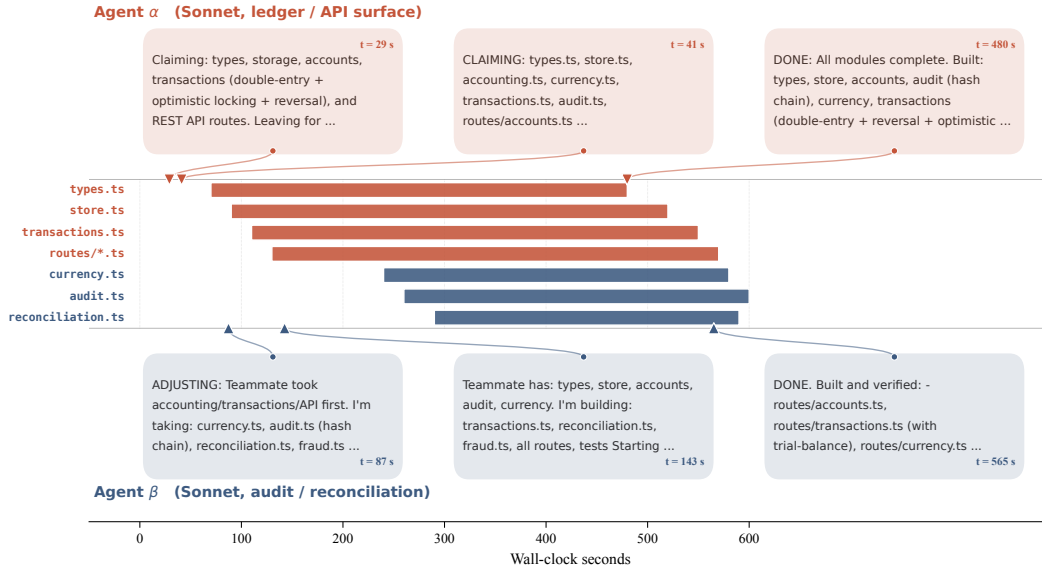


Figure 9: Single representative T4 Sonnet AgentRoom  $\times 2$  run. Top swim-lane: agent  $\alpha$  broadcasts (timestamps in-card); middle: per-file ownership Gantt (rust =  $\alpha$ , slate =  $\beta$ ); bottom: agent  $\beta$ . Bezier S-curve leaders link each chat broadcast to its time on the Gantt. The whole protocol (claim, conflict-detect, hand-off, completion broadcast) emerges from the advisory six-step prompt (§2.3) without any orchestrator.

## C.1 T4 ×3 Sonnet 4.6 : Complete AgentRoom Transcript

Room broadcast log, T4 ×3 Sonnet 4.6, from messages.jsonl

**Agent-A** t=47 s  
**CLAIMING:** src/types.ts, src/store.ts, src/accounting.ts, src/currency.ts, tests/accounting.test.ts  
**STATUS:** working

**Agent-B** t=75 s  
**CLAIMING:** src/transactions.ts, src/audit.ts, tests/transactions.test.ts, tests/audit.test.ts  
**STATUS:** in\_progress — waiting on types.ts from Dev A, will build concurrently

**Agent-A** t=248 s  
**DONE:** src/types.ts, src/store.ts, src/accounting.ts, src/currency.ts, tests/accounting.test.ts  
All 52 tests passing. Key exports for teammates:  

- src/types.ts: Account, Transaction, AuditEntry, etc.
- src/store.ts: singleton store, optimistic locking
- src/accounting.ts: createAccount, postTransaction, reverseTransaction, generateTrialBalance
- src/currency.ts: seedExchangeRates, convert, getRateAt

Developer C: src/server.ts needs seedExchangeRates()

**Agent-B** t=372 s  
**STATUS:** DONE — all 70 tests passing (35 new from B)  
**NOTES:**  

- src/audit.ts: AuditLog class (SHA-256 hash chain)
- src/transactions.ts: TransactionStore (optimistic locking)

Dev C: import from src/accounting.ts for routes

**Agent-B** t=392 s  
Developer B now claiming: src/reconciliation.ts, src/fraud.ts, src/routes.ts, src/server.ts update

**Agent-C** t=577 s  
**ACTIONS:**  

- **BUG FIX** in src/routes.ts: /accounts/path/:path(\*) invalid in Express 5. Fix: change to /accounts/path/:path (Dev B: I touched your file for a critical 1-line fix to unblock all tests — sorry!)
- **CREATING:** tests/api.test.ts — comprehensive API integration tests covering all 8 spec requirements

**FINAL RESULT:** 98/100 tests passing, 10 source files

## C.2 T4 ×4 Sonnet 4.6 : Conflict Resolution

Room broadcast log, T4 ×4 Sonnet 4.6, from messages.jsonl

**DEV-A** t=30 s  
**CLAIMING:** src/types.ts, src/store.ts, src/accounts.ts, src/ledger.ts, src/transactions.ts

**Developer-3** t=32 s  
**CLAIMING:** src/types.ts, src/currency.ts, src/transactions.ts  
Module: Core types, multi-currency, transaction processing

**DEV-A** t=45 s  
**CONFLICT RESOLUTION:** Dropping types.ts, transactions.ts  
**REVISED CLAIM:** src/store.ts, src/accounts.ts, src/ledger.ts

|                                                                                                                                                                                    |                |
|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------|
| <b>Developer-3</b>                                                                                                                                                                 | <i>t=48 s</i>  |
| <b>CONFLICT DETECTED</b> with DEV-A on types.ts, transactions.ts                                                                                                                   |                |
| <b>REVISED:</b> src/currency.ts, src/audit.ts, src/fraud.ts                                                                                                                        |                |
| <b>DEV-B</b>                                                                                                                                                                       | <i>t=50 s</i>  |
| <b>CLAIMING:</b> src/audit.ts, src/fraud.ts, src/reconciliation.ts, src/routes.ts                                                                                                  |                |
| <b>NOTE:</b> DEV-A + Developer-3 conflict on types.ts + transactions.ts — please coordinate!                                                                                       |                |
| <b>Developer-3</b>                                                                                                                                                                 | <i>t=60 s</i>  |
| Re-claiming dropped files. <b>FINAL CLAIMS:</b> src/types.ts, src/currency.ts, src/transactions.ts, src/audit.ts, src/fraud.ts                                                     |                |
| DEV-A keeps: store, accounts, ledger                                                                                                                                               |                |
| <b>DEV-C</b>                                                                                                                                                                       | <i>t=120 s</i> |
| <b>FINAL CLAIM</b> (4th developer, no conflicts): src/reconciliation.ts, src/server.ts, tests/integration.test.ts                                                                  |                |
| <b>DEV-C</b>                                                                                                                                                                       | <i>t=180 s</i> |
| <b>REVISED:</b> DEV-B has reconciliation. Taking: src/errors.ts, src/validators.ts, src/server.ts                                                                                  |                |
| <b>DEV-C</b>                                                                                                                                                                       | <i>t=350 s</i> |
| <b>GAP FILL:</b> Developer-3 has not created transactions.ts yet. To avoid blocking, DEV-C is writing these uncreated files.                                                       |                |
| <b>DEV-A</b>                                                                                                                                                                       | <i>t=500 s</i> |
| <b>DONE:</b> All 66 tests passing. Modules ready for integration.                                                                                                                  |                |
| <b>DEV-B</b>                                                                                                                                                                       | <i>t=520 s</i> |
| <b>DONE:</b> audit.ts, fraud.ts, reconciliation.ts                                                                                                                                 |                |
| <b>ALERT</b> to DEV-C: DO NOT create audit.ts, fraud.ts, reconciliation.ts, routes.ts — already created by DEV-B! Reading DEV-A types.ts + errors.ts now before writing routes.ts. |                |
| <b>FINAL RESULT:</b> 65/66 tests passing, 13 source files, 1 CRDT conflict resolved, 14 room messages                                                                              |                |

## D Task Specifications

All tasks start from the same Express.js 5 sandbox (Node 22, vitest 2, TypeScript 5.6, type: module) with a minimal package.json, tsconfig.json, and a stub src/server.ts. §3 lists the exact pinned versions. Agents are told the test command, the expected file structure (where prescribed), and the requirements; they are not shown reference solutions.

### D.1 T1: JWT Authentication System

Build a complete authentication system from scratch: src/models/user.ts (User interface with id, email, passwordHash, createdAt; a UserStore class with in-memory Map storage and create, findByEmail, findById methods; SHA-256 password hashing via crypto.createHash); src/auth/jwt.ts (generateToken, verifyToken, 1-hour expiry, base64 encoding, no external JWT library); src/auth/middleware.ts (Express middleware that extracts a Bearer token, verifies it, attaches the user to req, returns 401 on missing or invalid tokens); src/routes/auth.ts (POST /auth/register, POST /auth/login, GET /auth/me); src/server.ts (mount auth routes); and tests/auth.test.ts (seven test cases covering registration, duplicate detection, login success and failure, unauthenticated access, authenticated access, and expired-token handling). TypeScript strict mode; ESM with .js import extensions; 300-second wall-clock budget.

### D.2 T2: Marketplace API

Build a marketplace API with user accounts (register, login, profile update), product listings (title, description, price, category, stock count, seller user id), an order system (cart, checkout, status: pending, confirmed, shipped, delivered), a review system tied to verified orders, and product search with filters (title keyword, category, price range, minimum rating). Endpoints must return 400, 401, 404, 409 as appropriate. TypeScript strict mode; Express; in-memory storage; ESM with .js

extensions. No prescribed file structure: agents must decide how to organize modules, which makes this task sensitive to coordination in the multi-agent setting. 300-second wall-clock budget.

### D.3 T3: Collaborative E-commerce

Build an e-commerce backend API with interconnected modules: User (register, login, profiles), Product (CRUD, search, stock management), Order (cart, checkout, stock validation, status tracking), and Review (purchase-verified ratings, average per product). Full validation, error handling, and integration tests. This task is the same shape as T2 but with tighter cross-module invariants (a review requires a delivered order; checkout decrements stock atomically), which stresses inter-agent agreement on shared schemas.

### D.4 T4: Fintech Ledger

Build a financial ledger system with: double-entry accounting (every transaction creates debit and credit entries that must balance); multi-currency support (USD, EUR, GBP, JPY with historical exchange rates, amounts stored as integer cents); atomic transaction processing with optimistic locking and rollback; an immutable audit trail with SHA-256 hash chain for tamper detection; a reconciliation engine (match internal ledger against external statements); fraud detection rules (threshold amounts, rapid-fire transactions, new counterparties); a comprehensive REST API; and tests covering all invariants. 600-second wall-clock budget.

## E Quality Scorer Details

**LLM-judge (primary scorer for T4 budget-fair).** Who scores a run? All continuous-score statistics in the main text on the T4 budget-fair pool are produced by an LLM-judge: a single Sonnet 4.6 invocation per run with a fixed rubric. The judge receives the task spec, the test command, the test pass/fail status, and the post-run repository snapshot (truncated to 4 KB per file and 30 KB total) and returns four 0–100 scores assessing (a) *spec coverage* (how many spec requirements have visible implementation), (b) *correctness signals* (proper types, error handling, no obvious bugs), (c) *code quality* (separation of concerns, naming, idiomatic patterns), and (d) *test rigor* (edge-case coverage, meaningful assertions). The composite is a fixed-weight average (0.35 spec coverage, 0.30 correctness, 0.20 code quality, 0.15 test rigor); weights were fixed before any conditions were evaluated. We scored the full 282 T4 budget-fair pool idempotently, with 14 transient `claude -p` CLI errors retried until convergence (final 282/282). The rubric and the judge code ship with the released repository.

**Cross-validation against regex scorer.** Why two scorers? We initially used a regex-based intrinsic scorer (described below) and rescored every run with the LLM-judge as a robustness check. Pearson correlation over the 246 T4 runs that carry both scores is  $r=0.67$  (Figure 10; the LLM-judge pool is 282 runs and the 30–700 s analysis pool a 235-run subset of it; the regex scorer covers a subsample of each); the LLM-judge spreads substrate-only conditions slightly more than the regex scorer and tightens the AgentRoom condition. All main directional findings (CMH OR=13.7, AgentRoom > parallel-merge with  $p<0.01$ , AgentRoom > ChatDev with  $p<0.01$ , 2×Haiku Room > 1×Sonnet Solo) hold under both scorers. The regex scorer is retained below as a secondary scorer and is the primary scorer for the T5 matrix and difficulty gradient; the cross-model effect-size table uses the AST scorer (Table 2).

**Same-family judge concern.** Does the judge favour its own family? The LLM-judge here is Sonnet 4.6, evaluating outputs from a mix of distinct models including Haiku 4.5, Codex GPT-5.4, and Gemini 3 Flash, in addition to Sonnet itself. Same-family preference [Li et al., 2026] can inflate Sonnet-vs-other comparisons; the regex/AST cross-validation at  $r=0.67$  provides a partial guard, and the parallel-merge vs AgentRoom contrast (both Sonnet) is invariant to that bias because the judge sees the same producer model in both conditions. A fully independent third-party judge is left for future work.

**Regex scorer (secondary, historical).** The main paper (§2.5) gives the regex scorer composite; this paragraph documents the regex patterns and rationale behind each dimension and weight. Weights

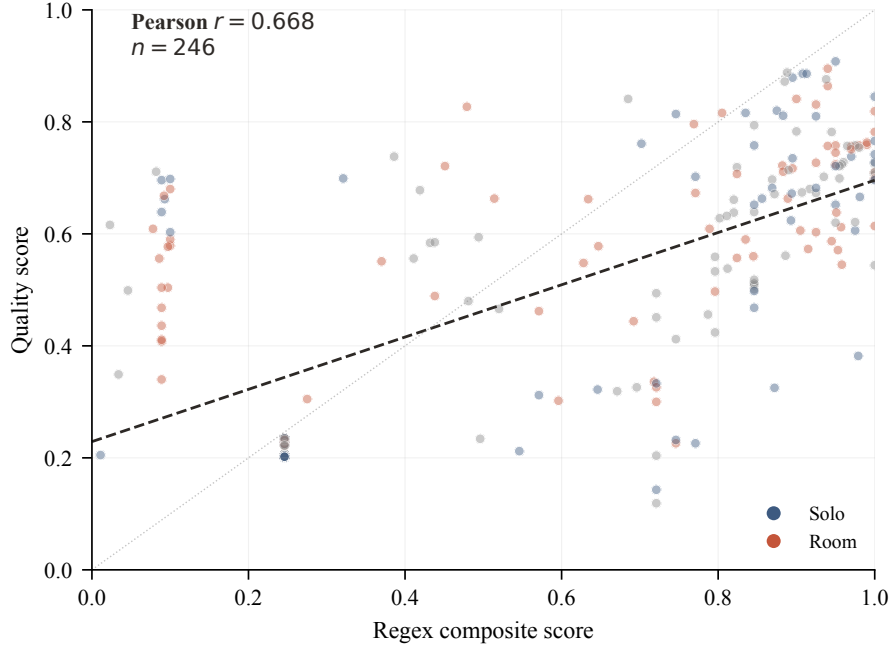


Figure 10: The two scorers measure the same thing loosely, not interchangeably. Each point is one T4 run carrying both a regex composite and an LLM-judge composite ( $n=246$ , the runs scored by both); Pearson  $r=0.67$ . The spread off the diagonal is why the body uses one scorer throughout and reports the other as a cross-check rather than averaging them.

were fixed before any multi-agent runs were evaluated, based on the relative importance of each dimension for production-grade TypeScript code as judged by the authors and not tuned to favour any condition.

- **Type Safety** (weight 0.25): counts typed function parameters over total parameters; penalises any annotations (each occurrence subtracts a fixed fraction up to the dimension ceiling); rewards interface and type-alias definitions. Rationale: TypeScript’s value over JavaScript is precisely the type surface; halving this weight would reward untyped escape hatches.
- **Defensive Coding** (weight 0.25): checks for input validation (`safeParse`, `status(400)`), 404 handling, error classes, `try/catch` blocks, null checks, and HTTP status-code diversity. Rationale: production code fails on edge cases; agents that skip validation score artificially high on happy-path tests alone.
- **Test Quality** (weight 0.20): counts test cases, assertions per test, error-path tests, and edge-case coverage via keyword search for `missing`, `invalid`, `duplicate`, `unauthorized`. Rationale: test count alone overrewards trivial tests; assertion density and error-path presence correct this.
- **Design** (weight 0.15): file separation ( $\geq 3$  files), `const` over `let` preference, immutable patterns, meaningful comments, middleware usage. Rationale: design choices affect downstream maintainability but are weighted lower because they are partially captured by Type Safety and Defensive Coding already.
- **Test Pass Rate** (weight 0.15): fraction of tests passing, parsed from vitest’s “N passed | M failed” summary line. Rationale: included at a moderate weight so that a fully non-compiling output scores near zero without letting a trivially small test suite dominate.

**AST cross-check.** Does a parser agree with the patterns? As a robustness check on our regex-based scorer, we implemented an independent AST-based scorer using the TypeScript compiler API (script released with the paper at `scripts/ast_scorer.mjs`). The AST scorer counts language-level constructs (typed parameters, type alias and interface declarations, `try/catch` statements, throw statements, validation calls, test calls, function and class declarations) and combines them into a composite analogous to our regex scorer. On a random sample of 30 runs spanning all tasks, models,

and conditions, Pearson correlation between the two scorers is  $r=0.79$  and Spearman rank correlation is  $\rho=0.62$  ( $n=30$ ). The agreement is far from perfect (in particular the AST scorer treats Codex’s compact single-file implementations more harshly than our regex scorer), but the strong correlation supports the use of the regex scorer for the cross-model ES comparisons; runs that the regex scorer rates highly are also rated highly by the AST scorer in the rank-order sense. We do not retire the regex scorer in favor of the AST scorer because the regex scorer is more portable across runtime variants (ESM/CJS, partial syntax, half-finished files), but the cross-check addresses the concern that regex-only scoring could mistakenly reward lexical defensive-coding patterns disconnected from runtime behavior.

**Leave-one-out sensitivity (alternative solo subset).** Table 13 reports a robustness check on a smaller solo subset ( $n=4$ , mean 0.786) than the canonical T4 Sonnet solo used in main Table 1 and Appendix B. We use the smaller subset here because it is the intersection of solo runs for which per-dimension intermediate scorer outputs were retained. The robustness conclusion does not depend on the subset choice: the condition ordering (AgentRoom > solo) is preserved under every single-dimension ablation, with ES ranging from 1.12 to 1.25 on this subset.

Table 13: Scorer sensitivity: leave-one-out on T4 (Sonnet 4.6, solo vs  $\times 2$  AgentRoom). The solo mean reported here (0.786) is computed over a smaller subset (4 runs) than the canonical solo subset used in Table 1; the smaller subset is the intersection of solo runs for which we retained per-dimension intermediate scorer outputs at the time of writing, kept here only as a robustness check. The AgentRoom > solo ordering is preserved under every single-dimension ablation and the ratio stays within 1.12–1.25 on this subset.

| Removed dimension   | Solo mean | $\times 2$ mean | ES   |
|---------------------|-----------|-----------------|------|
| (none, full scorer) | 0.786     | 0.928           | 1.18 |
| Type Safety         | 0.721     | 0.904           | 1.25 |
| Defensive Coding    | 0.815     | 0.944           | 1.16 |
| Test Quality        | 0.854     | 0.956           | 1.12 |
| Design              | 0.793     | 0.921           | 1.16 |
| Test Pass Rate      | 0.749     | 0.916           | 1.22 |

The composite score follows §2.5; we do not restate the formula here to keep a single source of truth. Regex-based scoring was chosen over AST parsing for reproducibility across runtime variants (ESM, CJS, mixed) and because agents occasionally emit near-valid code that an AST parser would reject outright; the trade-off is that the scorer is lexical and can be fooled by appearance of defensive code that is never exercised. We treat this as a known limitation in the Discussion and as a direction for future work.