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

Anonymous Authors[1]

Abstract

Concurrent multi-agent coding promises division of labor across modules and robustness through redundancy. Realtime collaborative editing protocols solve this coordination problem 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-task attempts with a one-file stub-and-exit. *AgentRoom* is a realtime collaborative editing protocol for concurrent coding agents: a runtime layer exposing file-level claim, status, and broadcast as MCP tools on top of a CRDT-merged shared filesystem. Across five frontier coding-CLI models on four backend coding tasks, AgentRoom $\times 2$ suppresses Solo abandonment on the CLI-stable models and tightens run-to-run variance. At matched compute, AgentRoom outperforms naive parallel ensembling, and ablating the coordination tools while keeping the CRDT substrate forfeits most of the coordination layer's contribution over the substrate alone. Coordination, not parallelism or merge correctness, is what carries the improvement.

1. Introduction

Multi-agent coding systems (Qian et al., 2024; Hong et al., 2024; Huang et al., 2024) predominantly use role-based pipelines (design $\rightarrow$ implement $\rightarrow$ review or sprint-style role rotation (Nguyen et al., 2024)), each role waiting for the previous artifact (further references in §5). 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.

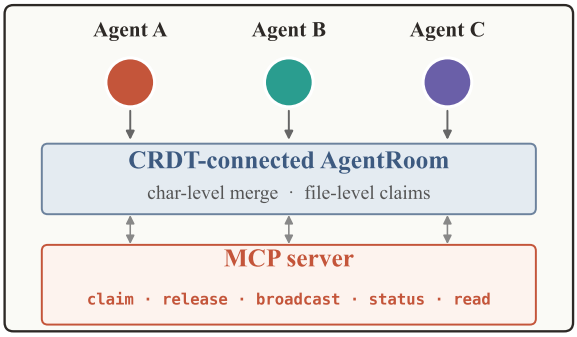

Figure 1. AgentRoom architecture: N coding agents share a CRDT-merged workspace; the MCP server exposes `claim/release/broadcast/read/state`.

We introduce **AgentRoom**, a *room as a state-management primitive* for concurrent coding agents: a runtime object with file-level claim semantics, an append-only broadcast log, and per-agent status, exposed as a small set of Model Context Protocol (MCP) tools (`room_claim`, `room_release`, `room_state`, `room_broadcast`, `room_read`) (Anthropic, 2024) on top of a CRDT-merged shared filesystem (Shapiro et al., 2011; Jahns, 2024; Automerge Project, 2023). This is a different coordination paradigm from CodeCRDT (Pugachev, 2025) (implicit inference, pre-assigned outliner/implementer roles): AgentRoom replaces both the implicit inference and the role pre-assignment with an explicit room and an advisory protocol (Section 2.3) that does not pre-assign roles or orchestrate handoffs. We compare AgentRoom against alternatives that combine the same agents in different ways at matched compute (Section 4).

Finding. Two agents in an AgentRoom *suppress the lone-agent "stub-and-exit" failure mode* (the agent emits a one-file skeleton and exits within $\sim$80 s, judging the task too hard): pooling across all 12 model$\times$task strata, the

[1]Anonymous Institution, Anonymous City, Anonymous Region, Anonymous Country. Correspondence to: Anonymous Author <anon.email@domain.com>.

Preliminary work. Under review at the Workshop on Failure Modes in Agentic AI (FAGEN) at ICML 2026. Do not distribute.

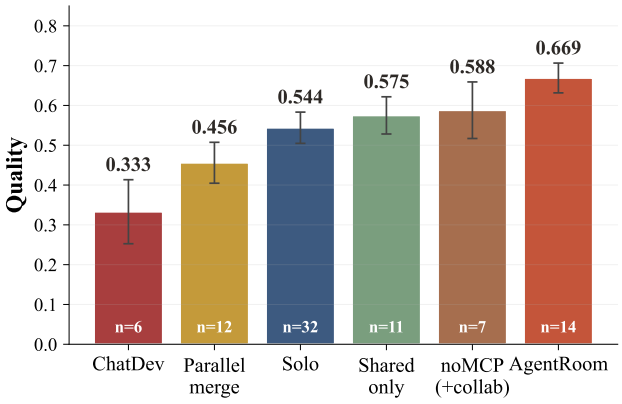

Figure 2. T4 six-condition coordination ablation (LLM-judge composite, Table 1): AgentRoom is the highest condition; the parallel-merge concurrent baseline is the only matched-compute condition that underperforms solo.

Cochran-Mantel-Haenszel common odds ratio for Solo-vs-AgentRoom abandonment is 13.7 (95% CI [3.9, 48], $p < 10^{-5}$); every stratum with abandonment events has an odds ratio above 1. Per-model and per-task breakdowns are reported in Section 4.1 (Figure 7). On the LLM-judge composite, the matched-compute parallel-merge vs AgentRoom contrast adds $+0.213$ (Welch p=0.003); a bundle-component probe (Appendix A.14) attributes the gain to the MCP coordination layer rather than the CRDT substrate, with the prompt-only step within noise at n=7, so we read the split qualitatively.

2. Method

2.1. AgentRoom: a CRDT-backed shared workspace

AgentRoom (Figure 1) is a single integrated primitive: a CRDT-mediated shared filesystem $\mathcal{W}$ with N concurrent agents and a coordination interface. The three components below are co-designed and not separately ablated.

Shared Workspace + CRDT. All N agents execute in $\mathcal{W}$; files written by any agent are immediately visible to all others. Concurrent writes within Δ_{crdt}=2 s are merged at the character level using `pycrdt` (Yrs port (Jahns, 2024) of Y.js), with a brace/paren-balance sanity check.

Coordination Interface. Agents access room state through MCP server (Anthropic, 2024) tools: `room_claim(path)` (atomic file ownership; rejects if another agent holds the path), `room_broadcast/read` (append-only message log), `room_state` (peer state). The interface enforces ownership at the state-management layer rather than asking agents to parse text claims; this distinguishes the design from chat-only multi-agent coordination (§4.6).

2.2. Formal model: collision rate and coordination layer

Let $\mathcal{A} = \{a_1, \ldots, a_N\}$ be the agents, $\mathcal{F}$ the file set, and $\mathcal{W}_t$ the workspace state at time t. The substrate is an op-based sequence CRDT (Shapiro et al., 2011; Roh et al., 2011; Kleppmann & Beresford, 2017) with edits stamped by Lamport timestamps (Lamport, 1978) 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. Full edit-set algebra and SEC formalisation in Appendix A.1.

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 $\mathcal{A}\times\mathcal{F}\to\{\text{OK}, \text{CONFLICT}\}$, `broadcast`$(m)$ appends m to $\mathcal{L}_t$, and `state`$()$ returns $(\mathcal{C}_t, \mathcal{L}_{[t-\tau,t]})$ where τ is a recent-log time window. `claim` is atomic and provides *mutual exclusion under cooperation*: $\forall f, \forall t: \left|\{i : (f,i) \in \mathcal{C}_t\}\right| \leq 1$. Following Chubby-style advisory locking (Burrows, 2006), enforcement is at the prompt layer rather than the kernel; an agent that writes without claiming still has its bytes preserved by SEC, and the room log surfaces the violation for the cross-agent bug-fix pattern (Appendix B).

Why the substrate alone is 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

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

giving ≈ 0.20 at the headline N=2, K_t≈5 regime. SEC merges colliding edits at the byte level but not at the intent level, so collisions break compilation or tests; the `claim` primitive drives this rate to zero at $O(1)$ per-edit cost.

Information density (motivational). 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 and showing the per-step joint information across agents scales near-linearly in N when claims keep the joint entropy from collapsing) is given in Appendix A.2 (Eq. 2); we present this as motivation for the bundle component probe rather than a formal proof.

2.3. Collaboration Protocol

Agents receive an advisory seven-step prompt: (1) *read* room state before writing; (2) *claim* target file paths; (3) *re-*

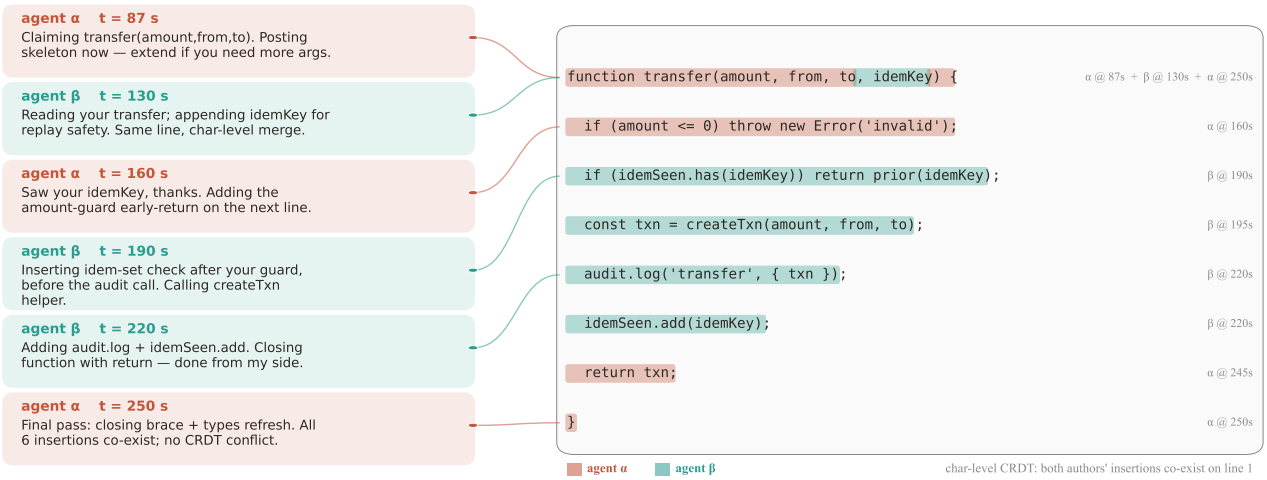

Figure 3. Real-time CRDT merge during a single T4 Sonnet AgentRoom ×2 run. The MCP-mediated chat (left) drives concurrent character-level edits in the shared file (right); both α's and β's insertions co-exist on the same line via the Yrs CRDT, with no byte-level conflict raised in this run. Per-message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

solve overlapping claims by revising; (4) *write* only claimed files; (5) *respect* files claimed by others (request edits via `room_broadcast`); (6) *poll* between subtasks; (7) *report* on completion. The protocol is advisory at the prompt layer; agents deviate (typically when a teammate stalls), and deviations are themselves informative (Appendix B). Imperfect adherence outperforms no coordination.

2.4. Baselines

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

2.5. Quality Score and Emergence Score

Each run produces a quality composite in $[0, 1]$. We use three scorers; full weights and dimension lists are in Appendix D. The headline body cells (T4 budget-fair) are scored by an LLM-judge (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 to address the concern that lexical pattern-matching can be gamed by ceremonial defensive code; the matched-compute ablation (Table 1) uses the LLM-judge composite; the regex/AST composites (Appendix D) preserve the same ordering. LLM-judge correlates with the regex scorer at Pearson r=0.67 on the full T4 budget-fair pool (n=282, App. D); regex and AST correlate at Pearson r=0.79, Spearman ρ=0.62 on a 30-run cross-check. The deterministic 1-file abandonment classifier used for the Tier I CMH binary outcome (§4.1) is independent of these continuous quality scorers; the headline directional findings (CMH OR= 13.7, AgentRoom > parallel-merge, AgentRoom > ChatDev) hold under both the regex and AST scorers (Appendix D). 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). Cross-vendor agreement is strong: even the weakest pair (Haiku vs Codex) reaches Pearson r=0.78, and Sonnet agrees with each cross-judge at r=0.86–0.87. The cross-vendor Codex judge is uniformly stricter (mean composite -0.151 vs Sonnet) but preserves the per-run ranking, so the LLM-judge composite is a robust ranking signal under judge-vendor variation. Gemini-3-Pro was quota-blocked at submission time and is omitted from the panel.

The *Emergence Score* is the ratio of N-agent mean quality to solo mean quality at the same model and task, $\mathrm{ES} = \bar{s}_N/\bar{s}_1$; ES>1 indicates multi-agent improvement. We report 95% bootstrap CIs on ES (10 000 resamples) for headline cells. Unless stated, N=2.

2.6. Tasks

The headline study uses four Express.js/TypeScript tasks spanning a difficulty axis and distinct domain types (specs in Appendix C): T1 (JWT authentication, 6 files), T2 (open-ended marketplace API, ≥10 files), T4 (double-entry finan-

cial ledger with multi-currency, hash-chained audit (Haber & Stornetta, 1991), reconciliation, fraud detection, $\geq$15 files), and T5 (algorithmic trading platform with order matching, portfolio P&L, risk, market data, accounts, $\geq$15 files). A fifth task T3 (collaborative e-commerce modules) is used only in the difficulty-gradient appendix (Appendix A) and is excluded from the CMH headline. The four headline tasks are different application domains (auth, marketplace, fintech, trading) at progressively larger scope; they share an Express.js/TypeScript runtime and the same sandbox template, not the application logic. Wall-clock budgets: 300 s (T1/T2/T3), 600 s (T4), 900 s (T5). Vitest suites are agent-authored on top of the template's single smoke test, so raw pass counts are reported descriptively only; correctness and quality are assessed by the LLM-judge composite cross-validated against the regex and AST scorers (§2.5). A small Python cross-domain check using DevBench (Li et al., 2024) is reported in Appendix A; broader cross-language evaluation is acknowledged as a limitation (§5). A representative single-run swim-lane visualisation of the protocol in action is shown in Appendix B (Figure 9).

3. Experiments

Models. Five frontier coding models, three providers: Claude Sonnet 4.6 and Haiku 4.5 (Anthropic), GPT-5.4 and GPT-5.4-mini (OpenAI), Gemini 3 Flash (Google). The headline cells use the four CLI-stable models (Sonnet, Haiku, GPT-5.4, Gemini 3); GPT-5.4-mini is excluded due to a vendor-CLI crash under concurrent MCP execution (§4.4, treated as a deployment caveat rather than an AgentRoom property). Accessed via vendor CLIs in non-interactive mode with vendor-default sampling; CLI versions pinned and listed in Appendix E.

Conditions and budgets. *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).

Reporting. Runs surviving the per-task wall-clock window with $\geq$ 30 s elapsed (the *budget-fair pool*) carry the headline statistics. The headline T4 pool is 30–700 s (the 600 s task budget plus a 100 s queue/network overhead allowance); the bundle-component probe (Appendix A.14) is a re-cut of the same budget-fair cells rather than a separate experiment. The ChatDev paradigm contrast uses a 1200 s subpool that matches AgentRoom $\times 2$ wall-clock compute (§4.3). Bootstrap 95% CIs (10 000 resamples) and per-cell n accompany every reported cell. The Tier II LLM-judge cells are computed from runs completed before the submission snapshot (2026-04-30); a few later runs preserve the direction but are not folded into the reported cells.

Metrics. Continuous quality score $s \in [0, 1]$ produced by an LLM-judge (Sonnet 4.6 with a fixed rubric assessing spec coverage, correctness signals, code quality, and test rigor; Appendix D) applied to the post-run repository snapshot; emergence score $\mathrm{ES} = \bar{s}_N/\bar{s}_1$ (§2.5); raw passing-test count; room-message count (broadcast and direct-send entries); CRDT-conflict count (guard firings within Δ_{crdt}=2 s). The LLM-judge is the primary scorer for all T4 budget-fair statistics in Sections 4–4.3 and is cross-validated against a regex-based intrinsic scorer at Pearson r=0.67 (n=282); cells outside the T4 LLM-scored pool (cross-model effect-size, difficulty gradient) are reported under the regex/AST scorer family in Appendix A as Tier III exploratory.

4. Results

Results are reported in two tiers of decreasing strength. Tier I is the variance and abandonment-rate reduction, which replicates across the 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 A.

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

Two agents in an AgentRoom cut run-to-run standard deviation by $\sim$ 30–45% among in-budget non-failing runs and eliminate a 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 the three CLI-stable models (Sonnet 4.6, Haiku 4.5, Codex GPT-5.4) and both headline tasks. On T4 Sonnet the standard deviation contracts from 0.22 to 0.14; T5 replicates (0.23 $\rightarrow$ 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 and is reported as exploratory in Appendix A; 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., 2025a), but AgentRoom achieves it through coordinated concurrent generation in a shared workspace rather than independent sampling and aggregation.

Abandonment is suppressed across models and tasks. Solo abandonment rates vary by model and tier; Agent-

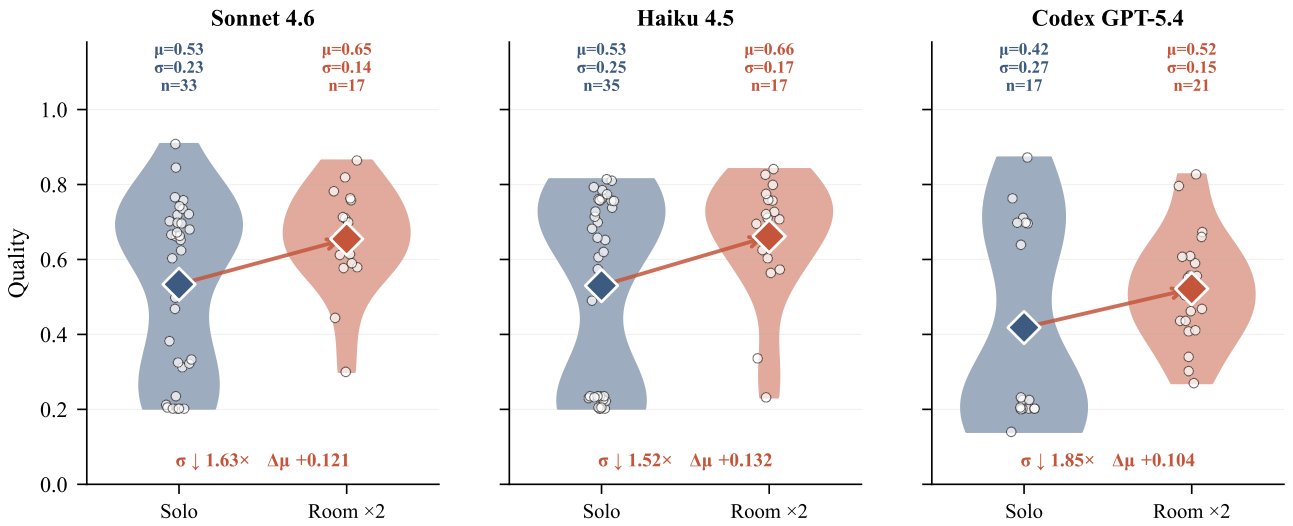

Figure 4. Solo (orange) vs AgentRoom ×2 (teal) score distributions for the three CLI-stable models on T4 (LLM-judge composite, budget-fair pool). σ contracts by $\sim$ 30–45% under AgentRoom in every panel (Sonnet 0.22→0.14, Haiku 0.25→0.17, Codex 0.26→0.14); mean shift varies by model.

Room drives them down across CLI-stable models and tiers. We classify each < 0.3 outcome by a deterministic codebook (a run is labelled 1-*file stub-and-exit abandonment* when the score is below 0.3 *and* either wall-clock duration is below 200 s or the source-file count is at most 2, otherwise *infrastructure failure*; Appendix A.13). We then test Solo against AgentRoom ×2 within each model×task stratum (Fisher's exact one-sided; per-stratum counts in Figure 7 and Appendix A.3). On T4, Haiku and Codex are individually significant while Sonnet and Gemini are directional in the same sign (Gemini underpowered, §4.4); T5 replicates the direction in the three CLI-stable models. Pooling the 12 strata, the Cochran-Mantel-Haenszel common odds ratio is 13.7 (95% CI $[3.9, 48]$, $p < 10^{-5}$, Tarone homogeneity p=0.92); no stratum reverses direction. The T4-only sub-pool yields OR= 11.9 ([3.0, 47]), robust across all 12 threshold combinations (Table 10); pooled abandonment rates fall from 32% to 5% on T4 and from 23% to 0% on T5. The LLM-judge composite tracks the same direction: on T4 Sonnet, excluding one infrastructure-failure run that distorts the unconditional mean, AgentRoom ×2 adds +0.125 over Solo (Welch p=0.022).

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

Table 1 (T4, Sonnet 4.6, budget-fair pool 30–700 s) compares AgentRoom against five alternatives at matched compute under the LLM-judge rubric. The six conditions are monotonically ordered: ChatDev-style sequential (0.333) $\ll$ parallel-merge (0.456) $<$ solo (0.544) $<$ shared-only (0.575) $<$ noMCP (0.588) $<$ AgentRoom (0.669); cell sizes are in Table 1. The headline 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. The gap is +0.213 in mean quality with Welch's t=3.35, p=0.003 (n=12 parallel-merge vs n=14 AgentRoom). On T4 Sonnet at this matched compute, naive concurrent ensembling *underperforms* a single agent (0.456 vs solo 0.544) because without coordination the second agent silently overwrites the first agent's entrypoint file; PM amplifies rather than averages the lone-agent failure mode. By contrast, the shared-only condition (concurrent agents on the same CRDT workspace but without the collaboration prompt or MCP tools) sits between solo and AgentRoom (0.575), recovering some of the loss but not reaching the full bundle. The cross-model ablation on Haiku and Codex is reported in Appendix A, Table 9: Haiku replicates the pattern, AgentRoom 0.662 above solo 0.530; Codex's solo

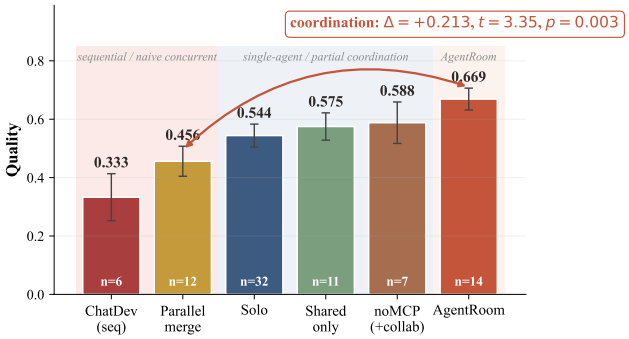

Figure 5. Same ablation as Table 1 with paradigm bands; arrow marks the parallel-merge → AgentRoom contrast (+0.213, p=0.003).

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 A.14). Cross-model replication in Appendix A, Table 9.

Condition	Mean	n	σ	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
AgentRoom (2 agents, full)	**0.669**	**14**	**0.140**	CRDT + room MCP + collab prompt

mean (0.384) is the lowest in the headline, consistent with its monolithic-architecture priors degrading multi-agent decomposition.

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

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 adapts ChatDev's three-phase workflow (the original is design → coding → testing; we use requirements analysis → technical design → implementation, with role-specific prompts and intermediate design artifacts handed off between phases) under a 1200 s budget that matches AgentRoom $\times 2$ compute. The phase boundary is the only structural difference from AgentRoom under the same model and CLI: the same agent runs each phase sequentially with the prior phase's artifact as input, rather than two agents writing concurrently to a shared room. 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; 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; no run in any other condition meets this criterion. Across the remaining n=6 genuine ChatDev runs (Sonnet, T4, budget-fair) the LLM-judge mean quality is $\bar{s}$=0.333 (σ=0.197); the matched AgentRoom $\times 2$ pool reaches 0.669 (n=14, σ=0.140). The ChatDev baseline runs under a 1200 s wall budget so its wall-clock compute aligns with two concurrent agents. Welch's t=3.78, two-sided p=0.006. The contrast isolates the paradigm choice: with model, CLI, task, and compute matched, the only difference is whether agents are coordinated through sequential phase-handoff (ChatDev-style) or concurrent CRDT-coordinated workspace (AgentRoom). Sequential pipelines amplify the lone-agent abandonment failure mode at every phase boundary: in a matched 6-run hand-labelled subsample, $4/6$ ChatDev runs produced a 1-file implementation-phase abandonment that nevertheless passes the boilerplate health-check test, while the 10 AgentRoom $\times 2$ runs inspected exhibited 0 such abandonments. 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)

Vendor coding CLIs differ in how strictly they enforce the MCP transport contract. Our early room server passed handshake with permissive clients (Claude Code, Codex CLI) but failed with strict-spec clients (Gemini CLI 0.40+); a single transport-framing fix unblocks Gemini and is the only CLI-side change required across the four CLI-stable models. GPT-5.4-mini still exhibits a client-side crash under concurrent MCP execution (Codex CLI 0.119 bug) and is excluded from the headline as a deployment caveat rather than an AgentRoom property. The Gemini 3 cell on T4 (n=5 Solo, n=5 AgentRoom $\times 2$, Sonnet partner, underpowered) is reported descriptively and reproduces the headline direction: Solo abandonment ($2/5$; 1-file stub-and-exit at 30–50 s) collapses to $0/5$ under AgentRoom, mean quality rising from 0.602 to 0.870 with 5/5 vitest pass.

4.5. Scaling and Heterogeneity

Sonnet T4 quality peaks at $\times 2$ then declines under the LLM-judge rubric (0.544 at $\times 1$, 0.669 at $\times 2$, declining to 0.489 at $\times 4$; full table in Appendix A); maximum test coverage peaks at $\times 3$. Per-run wall-clock cost grows linearly with N while broadcast traffic rises super-linearly past $\times 3$ (Figure 6, right panel), consistent with informational saturation of the single broadcast channel rather than a quality ceiling. 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.

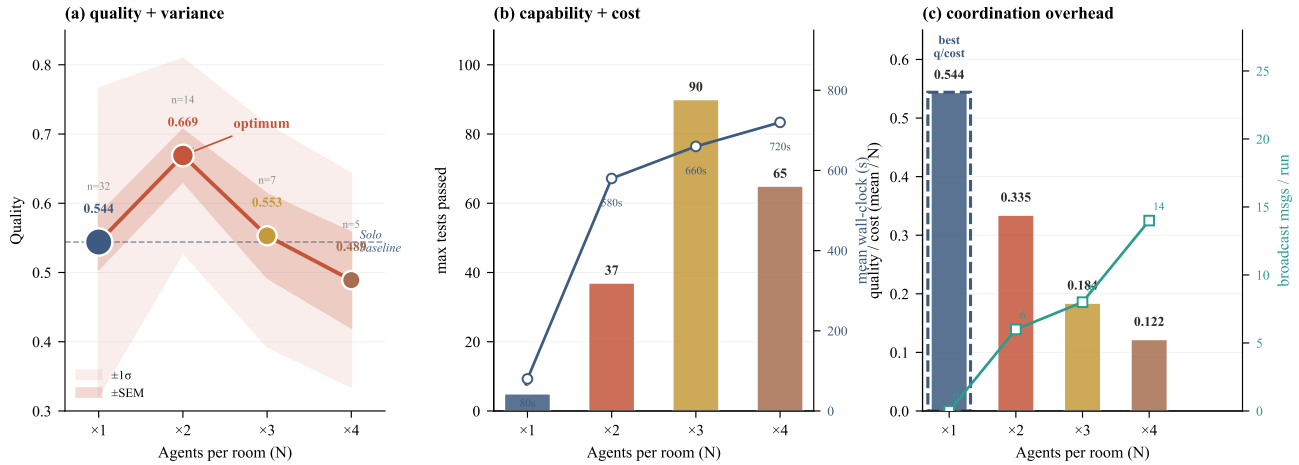

Figure 6. Agent-count scaling on T4 Sonnet 4.6 (N=1–4, budget-fair pool). Left: mean LLM-judge composite with σ band peaks at N=2 then declines. Middle: maximum tests passing peaks at N=3 while per-run wall-clock cost grows linearly. Right: message count rises super-linearly past N=3, consistent with single-broadcast-channel overhead.

Heterogeneous pairing on T4 (Sonnet primary partner) shows ×2 Sonnet+Codex at 0.721 (n=3), the highest cell across the heterogeneous combinations we tested, while ×2 Sonnet+Gemini is 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 AgentRoom's coordination layer does not erase capability differences between models (Table 5).

4.6. MCP Chat vs CRDT State

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. 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 (Table 1) and the parallel-merge vs AgentRoom contrast (+0.213, p=0.003) are in §4.2. The bundle is co-designed but its components are not equally responsible for the gain; the decomposition draws on the same Table 1 conditions (shared-only, noMCP, and AgentRoom) rather than new runs. Keeping the CRDT substrate and collaboration prompt but removing the MCP tool surface (Appendix A.14, n=7) lands at 0.588, only +0.013 above shared-only; the additional +0.081 to AgentRoom's 0.669 is contributed by the MCP coordination layer, carrying most of the substrate-to-full-bundle gain. The +0.013 prompt-only step is within noise at n=7, so we read this as a qualitative ordering (coordination layer > substrate) rather than a precise 86% split. AgentRoom is thus not merely "CRDT plus prompts": the load-bearing piece is the explicit coordination channel, with the substrate as the merge primitive beneath it. AgentRoom's claim protocol is a Chubby-style advisory lock (Burrows, 2006; Hunt et al., 2010; Gray & Cheriton, 1989) surfaced as an MCP tool with the agent as cooperator; consensus (Ongaro & Ousterhout, 2014) is unnecessary because CRDT convergence (Shapiro et al., 2011) with Lamport-style ordering (Lamport, 1978) already provides the safety property advisory claims need (in contrast to the operational-transformation lineage (Ellis & Gibbs, 1989; Sun et al., 1998) that requires a central transformation server). The advisory-plus-apology pattern matters because strict locking would block the cross-agent bug-fix case (Appendix B).

Activation of collaborative language. AgentRoom logs contain unprompted patterns the seven-step protocol does not specify: apologies for boundary violations, proactive gap-filling, "do not" warnings, dependency documentation (Appendix A, Appendix B). Across n=6 T4 Sonnet AgentRoom ×2 runs we observe module claiming in 6/6, plan-adjustment in 5/6, and cross-agent bug-fix in 2/6 (author labelling of room logs); the last is rare and should not be treated as the typical case. A representative bug-fix message (drawn from a ×3 Sonnet run where the pattern is more frequent) reads "*Dev B: I touched your file for a critical 1-line fix to unblock all tests; sorry!*". This apology-plus-edit pattern is absent from the prompt yet emergent in the trace. The same prompts produce none of these patterns in the shared-only condition, so the broadcast 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. 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. Enforcing ownership at the state-management level is deterministic where text-parsed claims are not.

Paradigm contrast: AgentRoom vs ChatDev-style sequential. At the same model, CLI, task, and compute (§4.3), sequential phase-handoff amplifies the lone-agent failure mode at every phase boundary; the prompt-vs-channel Haiku control (Appendix A) is less conclusive because the structured tool prompt also acts as workflow scaffolding at this n.

Cross-language and cross-domain replication. Two out-of-distribution checks show the AgentRoom framework ports beyond Express.js/TypeScript: a Rust+axum transposition of T4 (Codex GPT-5.4) runs cleanly under concurrent MCP execution, and 10 DevBench Python multi-file projects (Haiku 4.5) run with zero CRDT semantic conflicts (Appendices A, A.18). These establish that the substrate and the coordination tools operate unmodified in a second language and domain; the cell sizes are far below the power needed for a quality comparison, which we leave to future work.

Comparison with prior multi-agent coding systems. CodeCRDT (Pugachev, 2025) and AgentRoom share the CRDT-merged substrate but 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), and the empirical question is speedup over sequential baselines (reported task-dependent 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, and the empirical question is failure-mode suppression rather than speedup. AgentRoom's file-level claim eliminates concurrent same-file edits (0% semantic conflicts in the headline pool), whereas CodeCRDT's character-level concurrent edits produce the 5–10% rate. Concurrent industry deployments (Cursor's multi-agent editor, Anthropic Claude Code agent teams, JetBrains Air (Cursor, 2026; Anthropic, 2025; JetBrains, 2026)) use the file-claim/lock pattern in production; the rooms-as-state-primitives framing here names and quantifies that pattern with the matched-compute ablation the deployments do not report. Other recent multi-agent coding systems (Pan et al., 2025; Islam et al., 2025; Peng et al., 2026; Zhu et al., 2025; Ishibashi & Nishimura, 2024; Wang et al., 2025b; Wu et al., 2024) use sequential or staged-role paradigms; broader infrastructure work spans coordination benchmarks (Grötschla et al., 2026; Chen et al., 2024; Park et al., 2023), runtime debugging (Ashrafi et al., 2025), process modelling (Shafin et al., 2025), and self-reflection loops (Shinn et al., 2023). None target concurrent in-place editing on a CRDT-merged workspace.

6. Conclusion

Concurrent multi-agent coding has been treated as a problem of parallelism or of merge-correct substrate; we argue it is primarily about explicit coordination. In our matched-compute setting the main contributor is neither agent count nor the CRDT itself but a small set of state-management operations (file-level claim, broadcast log, per-agent status) exposed as a runtime room. Without coordination, naive concurrency underperforms a single agent; with the substrate alone, agents converge on bytes but not on intent. Within our setting (Express.js/TypeScript backends, N=2, four CLI-stable models), the substrate–coordination separation suggests that an explicit room-style primitive is the contributor we identify, and that matched-compute ablation is the methodological lens for measuring such contributions; whether this generalizes beyond coding tasks remains open.

7. Limitations

The headline spans four backend domains and the three powered CLI-stable models from two providers (n=7–35 per cell; Gemini 3, the fourth CLI-stable model, is underpowered at n=5+5 and Codex-mini is excluded, §4.4); the Tier I abandonment claim is powered by pooling the 12 strata under CMH rather than any single cell. The parallel-merge vs AgentRoom contrast holds the same Sonnet producer on both sides (Li et al., 2026; Wataoka et al., 2025); the LLM-judge composite is cross-validated against an independent regex scorer at Pearson r=0.67 (n=282) and a TypeScript-AST scorer (regex–AST r=0.79). The cost-equivalent cross-model comparison (Appendix A.8) is bias-exposed and framed as suggestive. The bundle-component probe reuses the ablation cells (Appendix A.14); 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; oracle-free evaluation of in-the-wild software is outside scope.

References

Anthropic. Introducing the Model Context Protocol. https://www.anthropic.com/news/model-

context-protocol, 2024.

Anthropic. Create custom subagents: Claude Code documentation. https://code.claude.com/docs/en/sub-agents, 2025.

Ashrafi, N., Bouktif, S., and Mediani, M. 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.

Automerge Project. Automerge: A library of data structures for building collaborative applications. https://automerge.org, 2023.

Burrows, M. The chubby lock service for loosely-coupled distributed systems. In *7th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2006. URL https://www.usenix.org/conference/osdi-06/chubby-lock-service-loosely-coupled-distributed-systems.

Chen, W., Su, Y., Zuo, J., Yang, C., Yuan, C., Chan, C.-M., Yu, H., Lu, Y., Hung, Y.-H., Qian, C., Qin, Y., Cong, X., Xie, R., Liu, Z., Sun, M., and Zhou, J. Agentverse: Facilitating multi-agent collaboration and exploring emergent behaviors. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=EHg5GDnyq1.

Cursor. New Cursor interface (Cursor 3.0): Run many agents in parallel across repos and environments. https://cursor.com/changelog/3-0, April 2026.

Ellis, C. A. and Gibbs, S. J. Concurrency control in groupware systems. In *Proceedings of the 1989 ACM SIGMOD International Conference on Management of Data*, pp. 399–407, 1989. doi: 10.1145/67544.66963. URL https://doi.org/10.1145/67544.66963.

Gray, C. G. and Cheriton, D. R. Leases: An efficient fault-tolerant mechanism for distributed file cache consistency. In *Proceedings of the Twelfth ACM Symposium on Operating Systems Principles (SOSP)*, pp. 202–210, 1989. doi: 10.1145/74850.74870. URL https://doi.org/10.1145/74850.74870.

Grötschla, F., Müller, L., Tönshoff, J., Galkin, M., and Perozzi, B. Agentsnet: Coordination and collaborative reasoning in multi-agent LLMs, 2026. URL https://openreview.net/forum?id=gsSIH0mZ0Y.

Haber, S. and Stornetta, W. S. 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.

Hong, S., Zhuge, M., Chen, J., Zheng, X., Cheng, Y., Zhang, C., Wang, J., Wang, Z., Yau, S. K. S., Lin, Z., Zhou, L., Ran, C., Xiao, L., Wu, C., and Schmidhuber, J. MetaGPT: Meta programming for a multi-agent collaborative framework. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=VtmBAGCN7o.

Huang, D., Zhang, J. M., Luck, M., Bu, Q., Qing, Y., and Cui, H. Agentcoder: Multi-agent-based code generation with iterative testing and optimisation, 2024. URL https://arxiv.org/abs/2312.13010.

Hunt, P., Konar, M., Junqueira, F. P., and Reed, B. 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.

Ishibashi, Y. and Nishimura, Y. Self-organized agents: A llm multi-agent framework toward ultra large-scale code generation and optimization. *CoRR*, abs/2404.02183, 2024. URL https://doi.org/10.48550/arXiv.2404.02183.

Islam, M. A., Ali, M. E., and Parvez, M. R. CodeSim: Multi-agent code generation and problem solving through simulation-driven planning and debugging. In Chiruzzo, L., Ritter, A., and Wang, L. (eds.), *Findings of the Association for Computational Linguistics: NAACL 2025*, pp. 5128–5154, Albuquerque, New Mexico, April 2025. Association for Computational Linguistics. ISBN 979-8-89176-195-7. doi: 10.18653/v1/2025.findings-naacl.285. URL https://aclanthology.org/2025.findings-naacl.285/.

Jahns, K. Yjs: A CRDT framework for shared editing. https://github.com/yjs/yjs, 2024.

JetBrains. Air launches as public preview: A new wave of dev tooling. https://blog.jetbrains.com/air/2026/03/air-launches-as-public-preview-a-new-wave-of-dev-tooling-built-on-26-years-of-experience/, March 2026.

Kleppmann, M. and Beresford, A. R. A conflict-free replicated json datatype. *IEEE Transactions on Parallel and Distributed Systems*, 28(10):2733–2746, October 2017. ISSN 1045-9219. doi: 10.1109/tpds.2017.2697382. URL http://dx.doi.org/10.1109/TPDS.2017.2697382.

Lamport, L. Time, clocks, and the ordering of events in a distributed system. *Communications of the ACM*, 21(7):

558–565, 1978. doi: 10.1145/359545.359563. URL https://doi.org/10.1145/359545.359563.

Li, B., Wu, W., Tang, Z., Shi, L., Yang, J., Li, J., Yao, S., Qian, C., Hui, B., Zhang, Q., Yu, Z., Du, H., Yang, P., Lin, D., Peng, C., and Chen, K. Prompting large language models to tackle the full software development lifecycle: A case study, 2024. URL https://arxiv.org/abs/2403.08604.

Li, D., Sun, R., Huang, Y., Zhong, M., Jiang, B., Han, J., Zhang, X., Wang, W., and huan liu. Preference leakage: A contamination problem in LLM-as-a-judge. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=grIvSXVJ65.

Nguyen, M. H., Chau, T. P., Nguyen, P. X., and Bui, N. D. Q. Agilecoder: Dynamic collaborative agents for software development based on agile methodology, 2024. URL https://arxiv.org/abs/2406.11912.

Ongaro, D. and Ousterhout, J. In search of an understandable consensus algorithm. In *2014 USENIX Annual Technical Conference (USENIX ATC 14)*, pp. 305–319. USENIX Association, 2014. URL https://www.usenix.org/conference/atc14/technical-sessions/presentation/ongaro.

Pan, R., Zhang, H., and Liu, C. Codecor: An llm-based self-reflective multi-agent framework for code generation, 2025. URL https://arxiv.org/abs/2501.07811.

Park, J. S., O'Brien, J. C., Cai, C. J., Morris, M. R., Liang, P., and Bernstein, M. S. Generative agents: Interactive simulacra of human behavior, 2023. URL https://arxiv.org/abs/2304.03442.

Peng, Y., Hou, H., Zhu, X., He, Y. T., and Yu, F. R. Semag: Self-evolutionary multi-agent code generation, 2026. URL https://arxiv.org/abs/2603.15707.

Pugachev, S. Codecrdt: Observation-driven coordination for multi-agent llm code generation, 2025. URL https://arxiv.org/abs/2510.18893.

Qian, C., Liu, W., Liu, H., Chen, N., Dang, Y., Li, J., Yang, C., Chen, W., Su, Y., Cong, X., Xu, J., Li, D., Liu, Z., and Sun, M. ChatDev: Communicative agents for software development. In Ku, L.-W., Martins, A., and Srikumar, V. (eds.), *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 15174–15186, Bangkok, Thailand, August 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.acl-long.810. URL https://aclanthology.org/2024.acl-long.810/.

Roh, H.-G., Jeon, M., Kim, J.-S., and Lee, J. Replicated abstract data types: Building blocks for collaborative applications. *Journal of Parallel and Distributed Computing*, 71(3):354–368, 2011. doi: 10.1016/j.jpdc.2010.12.006. URL https://doi.org/10.1016/j.jpdc.2010.12.006.

Shafin, W. I., Rafi, M. N., Li, Z., and Chen, T.-H. Evaluating software process models for multi-agent class-level code generation, 2025. URL https://arxiv.org/abs/2511.09794.

Shapiro, M., Preguiça, N., Baquero, C., and Zawirski, M. Conflict-free replicated data types. In *Symposium on Self-Stabilizing Systems (SSS)*, pp. 386–400, 2011. doi: 10.1007/978-3-642-24550-3_29. URL https://doi.org/10.1007/978-3-642-24550-3_29.

Shinn, N., Cassano, F., Berman, E., Gopinath, A., Narasimhan, K., and Yao, S. Reflexion: language agents with verbal reinforcement learning, 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/1b44b878bb782e6954cd888628510e90-Paper-Conference.pdf.

Sun, C., Jia, X., Zhang, Y., Yang, Y., and Chen, D. Achieving convergence, causality preservation, and intention preservation in real-time cooperative editing systems. *ACM Transactions on Computer-Human Interaction*, 5 (1):63–108, 1998. doi: 10.1145/274444.274447. URL https://doi.org/10.1145/274444.274447.

Wang, J., WANG, J., Athiwaratkun, B., Zhang, C., and Zou, J. Mixture-of-agents enhances large language model capabilities. In *The Thirteenth International Conference on Learning Representations*, 2025a. URL https://openreview.net/forum?id=h0ZfDIrj7T.

Wang, X., Wei, J., Schuurmans, D., Le, Q. V., Chi, E. H., Narang, S., Chowdhery, A., and Zhou, D. Self-consistency improves chain of thought reasoning in language models. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=1PL1NIMMrw.

Wang, X., Li, B., Song, Y., Xu, F. F., Tang, X., Zhuge, M., Pan, J., Song, Y., Li, B., Singh, J., Tran, H. H., Li, F., Ma, R., Zheng, M., Qian, B., Shao, Y., Muennighoff, N., Zhang, Y., Hui, B., Lin, J., Brennan, R., Peng, H., Ji, H., and Neubig, G. Openhands: An open platform for AI software developers as generalist agents. In *The Thirteenth International Conference on Learning Representations*, 2025b. URL https://openreview.net/forum?id=OJd3ayDDoF.

Wataoka, K., Takahashi, T., and Ri, R. Self-preference bias in llm-as-a-judge, 2025. URL https://arxiv.org/abs/2410.21819.

Wu, Q., Bansal, G., Zhang, J., Wu, Y., Li, B., Zhu, E., Jiang, L., Zhang, X., Zhang, S., Liu, J., Awadallah, A. H., White, R. W., Burger, D., and Wang, C. Autogen: Enabling next-gen LLM applications via multi-agent conversations. In *First Conference on Language Modeling*, 2024. URL https://openreview.net/forum?id=BAakY1hNKS.

Zhu, K., Du, H., Hong, Z., Yang, X., Guo, S., Wang, Z., Wang, Z., Qian, C., Tang, X., Ji, H., and You, J. MultiAgentBench : Evaluating the collaboration and competition of LLM agents. In Che, W., Nabende, J., Shutova, E., and Pilehvar, M. T. (eds.), *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 8580–8622, Vienna, Austria, July 2025. Association for Computational Linguistics. ISBN 979-8-89176-251-0. doi: 10.18653/v1/2025.acl-long.421. URL https://aclanthology.org/2025.acl-long.421/.

A. Additional Results

A.1. Formal Model: Edit-Set Algebra and CRDT Convergence

Edit-set algebra. An *edit* on file f is a tuple $e = (\text{OP},\ pos,\ c)$ 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.

Sequential vs concurrent edit production. A standard LLM coding agent produces a *patch sequence* $\mathbf{P}^{\text{seq}} = (P_1, \ldots, P_T)$ 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 $\mathcal{W}_T = \text{MERGE}(\mathbf{E}^{\text{con}})$. The substrate is an op-based sequence CRDT (Shapiro et al., 2011; Roh et al., 2011; Kleppmann & Beresford, 2017) with edits stamped by Lamport timestamps (Lamport, 1978), so *merge* is commutative and associative and the workspace satisfies *strong eventual consistency* (SEC): 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 Argument (motivational, not a tight bound)

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

$$I\big(\{P_t^i\}_{i=1}^N;\ X \mid \mathcal{W}_t\big) \ \approx\ \sum_{i=1}^{N} H(P_t^i \mid \mathcal{W}_t) \ -\ H\big(\{P_t^i\}_{i=1}^N \mid \mathcal{W}_t,\ X\big), \quad \text{under claim-enforced } P_t^i \perp\!\!\!\perp P_t^j \mid \mathcal{W}_t. \tag{2}$$

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 concurrent compute is wasted on redundant edits. This is the information-theoretic reading of the matched-compute ablation: AgentRoom delivers higher quality not because individual agents are smarter, but because the coordination layer keeps per-step novel information from collapsing under collision. We present this as motivation for the empirical bundle component probe (§5) rather than a formal proof; the per-agent independence assumption is approximate in practice (agents do read each other's broadcasts; Appendix B), and Eq. 2 grounds the decomposition rather than bounding a tight effect size.

A.3. 12-Stratum CMH Forest (Tier I headline visual)

Figure 7 is the visual companion to the abandonment-rate CMH analysis reported in §4.1: per-stratum density-wave width $\propto$ inverse standard error, circle area $\propto$ stratum sample size, pooled common odds ratio 13.7 (95% CI $[3.9,\ 48]$, χ^2=22.8, $p < 10^{-5}$, Tarone homogeneity p=0.92). Every stratum with abandonment events points above OR$=1$; zero-event strata sit at OR≈ 1.

A.4. Cross-Model ES (Tier III, Exploratory)

Cross-model effect-size of $\times$2/solo, framed as exploratory only (Tier III; the variance/abandonment headline carries the empirical claims). Table 2 reports the headline numbers; Figure 8 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.

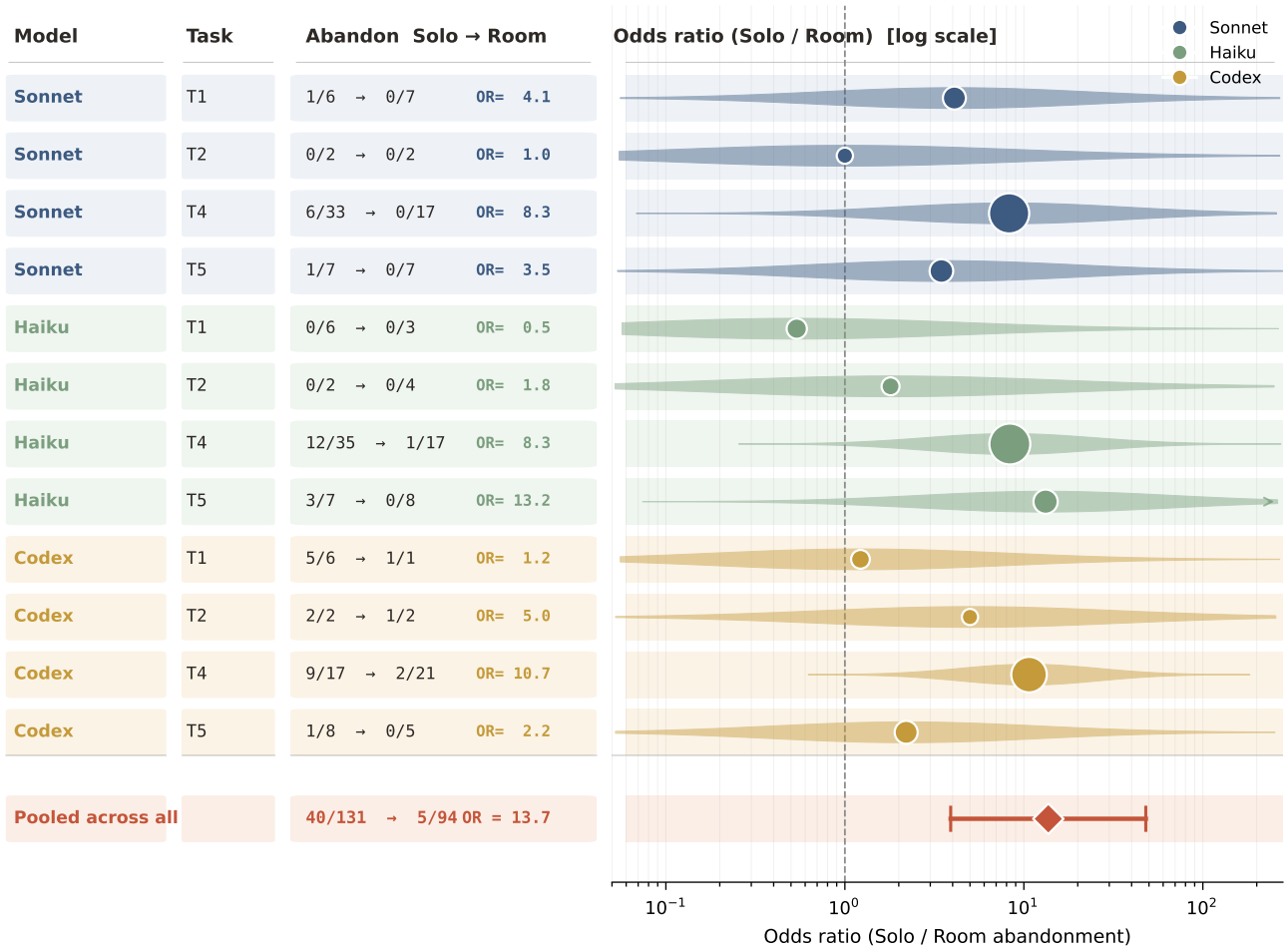

Figure 7. Cochran-Mantel-Haenszel forest plot of Solo-vs-AgentRoom 1-file abandonment across all 12 model×task strata (Sonnet/Haiku/-Codex × T1/T2/T4/T5).

A.5. Scaling on T4 Sonnet

A.6. Difficulty Gradient (T1/T2/T3/T4/T5, Sonnet 4.6)

Table 4 reports Sonnet 4.6 ES across all five tasks under the regex/AST scorer family (outlier-filtered, score ≥ 0.3). The LLM-judge rubric was applied 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; 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 headline.

A.7. Heterogeneous Pairs (T4, Sonnet 4.6 partner)

Table 5 compares paired-Sonnet against two heterogeneous pairings on T4. Sonnet+Codex sits at the top of the cell at 0.721 ($n=3$) but is underpowered relative to the homogeneous Sonnet$\times 2$ baseline ($n=14$); we treat heterogeneity as exploratory.

A.8. Cost-Equivalent Paired Model (Tier II.b, T4)

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 $p=0.036$. Haiku is roughly $4\times$ cheaper per token than

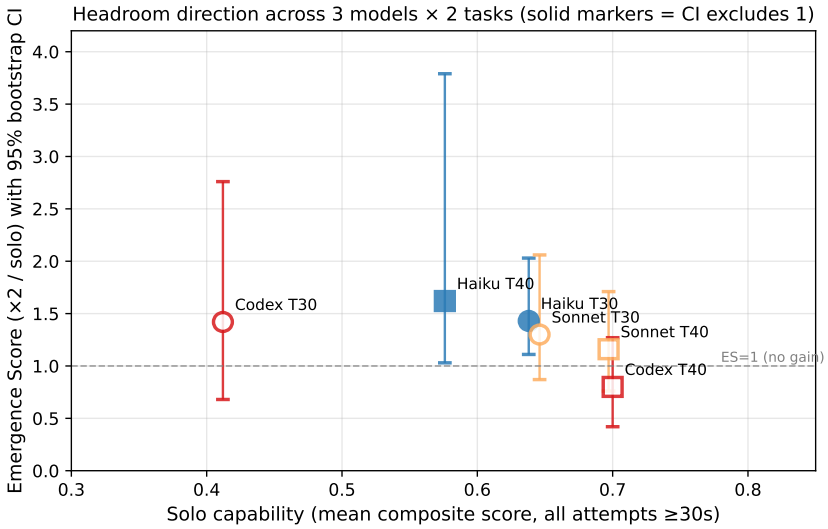

Figure 8. Exploratory cross-model headroom direction across 3 models × 2 tasks. Filled markers = CI excludes 1. Only Haiku T4 clears the CI; the other five cells span 0.79–1.96× point-estimate with CIs that include 1.

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; `scripts/ast_scorer.mjs`); regex-scorer values in parentheses 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)	×2 (n)	ES (AST / regex)	95% CI (AST)
Haiku 4.5	73.3%	0.520 (13)	**0.929** (8)	**1.79**× / 1.58×	[1.24, 3.35]*
Codex GPT-5.4	∼80%	0.248 (9)	0.487 (13)	1.96× / 1.07×	[0.85, 8.85]
Sonnet 4.6	79.6%	0.463 (11)	0.630 (15)	1.36× / 1.14×	[0.84, 2.29]

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, and 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).

A.9. Compute Cost (T4, Sonnet 4.6)

Table 6 separates wall-clock (elapsed) from cumulative compute (sum of per-agent CLI durations, a proxy for billable 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; the $\times 3$ cell pays an additional $\sim$ 500 s of compute for a small quality regression.

A.10. T5 Cross-Model Matrix

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. Table 7 reports every cell we collected on T5 under the AST scorer; only Haiku $\times 2$ point-estimate-positive cleanly 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). The CLI-unstable rows (Gemini, GPT-5.4-mini) are reported for completeness but excluded from the headline (§4.4).

A.11. Behavior Frequency (T4 ×2, Sonnet 4.6, n=6)

We coded each of six valid $\times 2$ AgentRoom runs on T4 Sonnet for six categories of unprompted collaborative behavior (module claiming via broadcast, plan adjustment after reading the teammate, export-list sharing, proactive gap filling,

Table 3. Scaling on T4 (Sonnet 4.6), budget-fair pool (30–700 s), LLM-judge composite. Quality peaks at $\times 2$ then declines; tests = max across runs at each N.

Agents	Score	Tests Passed	AgentRoom Messages	n
$\times 1$ (solo)	0.544	5	—	32
$\times 2$	**0.669**	37	4–7	14
$\times 3$	0.553	90	7–8	7
$\times 4$	0.489	65	12–14	5

Table 4. Sonnet 4.6 ES across the difficulty gradient (regex/AST scorer, outlier-filtered ≥ 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)	**0.958** (10)	**1.20**$\times$
T5	0.697 (7)	0.810 (7)	1.16$\times$

cross-agent bug fix, and apology for boundary violation). Module claiming and plan adjustment are nearly universal, while cross-agent bug fix and apology appear in only a third of runs and should not be treated as the typical case.

A.12. Cross-Model Ablation (T4, all CLI-stable models)

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 9 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). Codex non-Solo/AgentRoom cells were not collected in budget due to a CLI hang we did not unblock in time for submission.

A.13. Codebook Threshold Sensitivity (Tier I CMH)

Implementation. The codebook is implemented 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. Because classification is deterministic, blinded inter-rater agreement is trivially κ=1; residual classification risk lies in the threshold choice, which we sweep below.

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 ≤ 2, and as *infrastructure failure* otherwise. Table 10 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 headline 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 headline 12-stratum CMH (3 models $\times$ 4 tasks, OR= 13.7) reported in §4.1 is correspondingly stable under the threshold sweep.

A.14. Bundle Component Probe (CRDT + prompt, no MCP tools)

Reviewers asked whether the AgentRoom bundle's components (CRDT layer + MCP coordination tools + collaboration prompt) can be partially separated. 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 (σ=0.188). The cell sits between

Table 5. Heterogeneous vs. homogeneous ×2 on T4 (LLM-judge, budget-fair).

Pair	Score (n)	Tests
Sonnet ×2 (homo)	0.669 (14)	37
Sonnet+Codex	**0.721** (3)	**71**
Sonnet+Gemini	0.542 (3)	65

Table 6. 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 ×2	2	561	1072	**0.669**
AgentRoom ×3	3	566	1572	0.553

shared-only (0.575, no prompt, no tools, n=11) and AgentRoom (0.669, with both, n=14); 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, so the coordination layer accounts for 86% of the substrate-to-full-bundle gain. The percentage split is the point estimate; at n=7 the variance is substantial and the qualitative ordering MCP-layer-contribution > CRDT-substrate-contribution is the load-bearing claim. The component contributions are not 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.

A.15. Pool Sensitivity (T4 Sonnet ablation, LLM-judge)

Table 11 reports the T4 Sonnet ablation under two pools: the budget-fair 30–700 s pool used in Table 1, and the unrestricted full pool (≥ 30 s elapsed, no upper bound). The substrate-vs-AgentRoom ordering, AgentRoom > shared > solo > parallel-merge, is preserved 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 the headline because it enforces matched compute across conditions.

A.16. Solo+Collab-Prompt Control (T4, Sonnet and Haiku)

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 12) is informative.

For Sonnet, the structured prompt alone scores below bare solo and far below AgentRoom ×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: the prompt becomes counterproductive without a referent multi-agent context. For Haiku, the same prompt scores above bare solo and statistically indistinguishable from AgentRoom ×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.

A.17. Cross-Domain Validation: DevBench (multi-file Python from PRD)

We ran 10 DevBench (Li et al., 2024) Python projects with Haiku 4.5 in solo and ×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 (no scaffolded source code is provided to the agents).

Pass-rate result (Table 13): ×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 ×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. The cross-domain

Table 7. T5 cross-model matrix (AST scorer, 900 s budget). Haiku ×2 is the only cell whose CI cleanly clears 1; Sonnet ×2 is flat, Codex regresses.

Model	Solo (n)	×2 (n)	×3 (n)	ES(×2)	Tests (×2)
Haiku 4.5	0.416 (8)	**0.707** (7)	0.822 (1)	**1.70**×	38
Sonnet 4.6	0.618 (7)	0.674 (7)	0.730 (2)	1.09×	57
GPT-5.4	0.667 (9)	0.531 (7)	0.761 (1)	0.79×	8
CLI-unstable (informational only):					
Gemini Flash	0.851 (3)	0.506 (9)	0.246 (3)	0.59×	12
GPT-5.4-mini	0.077 (1)	—	—	—	—
Heterogeneous and mixed:					
Sonnet+GPT-5.4	—	0.857 (2)	—	—	52
Sonnet+Haiku	—	0.900 (1)	—	—	—
Sonnet+Gemini	—	0.626 (3)	—	—	42
Mixed S+C+G ×3	—	—	0.673 (4)	—	—
Mixed S+H+C ×3	—	—	0.791 (2)	—	—

Table 8. Frequency of unprompted collaborative behaviors across 6 valid ×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

evidence is consistent with the TypeScript headline: ×2 does not break correctness on multi-file Python projects, and on at least one task the ×2 condition completes a task the solo condition cannot finish in budget.

A.18. Cross-Language Validation: Rust + axum (T4 fintech ledger transposed)

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 ×2) on a vendor-pinned tokio/axum/serde stack with a 700 s wall-clock budget. All AgentRoom ×2 runs completed cleanly under concurrent MCP execution with full GPT-5.4 (§4.4 for the Codex-mini deployment caveat), with no `room_claim` crashes; each run produced a compiling, test-bearing service (2008–2452 Rust lines across 3–6 `.rs` files).

The check establishes infrastructure portability: the CRDT substrate, the room MCP tools, and the Codex CLI operate unmodified in a second language under concurrent execution. The cell sizes (Solo n=5, AgentRoom ×2 n=4) are far below the power needed for a quality comparison, so we make no quality claim here and defer a powered cross-language evaluation to future work.

B. Full AgentRoom Logs

The two transcripts below are reproduced byte-exact from the `messages.jsonl` log of each run; whitespace, casing, speaker labels, and punctuation (including any dashes inside agent messages) are preserved as evidence rather than edited for style. Figure 9 pairs a single representative run's chat broadcasts with the per-file ownership timeline.

Table 9. Cross-model T4 ablation (LLM-judge composite, budget-fair pool).

Model	Solo	Parallel merge	Shared only	noMCP	AgentRoom ×2
Sonnet 4.6	0.544 (n=32)	0.456 (n=12)	0.575 (n=11)	0.588 (n=7)	**0.669** (n=14)
Haiku 4.5	0.530 (n=35)	0.383 (n=12)	0.657 (n=7)	0.579 (n=6)	**0.662** (n=17)
Codex GPT-5.4	0.384 (n=12)	n/a	n/a	n/a	0.529 (n=17)

Table 10. Threshold sensitivity for the Tier I CMH headline. 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.

d thr	f thr	S_{solo}	S_{room}	H_{solo}	H_{room}	C_{solo}	C_{room}	OR	95% CI	p
< 150	≤ 1	6/33	0/17	12/35	1/17	9/17	0/21	24.2	[3.5, 169]	0.00001
< 150	≤ 2	6/33	0/17	12/35	1/17	9/17	2/21	11.9	[3.0, 47]	0.00005
< 150	≤ 3	6/33	0/17	12/35	1/17	10/17	6/21	6.3	[2.1, 19]	0.00069
< 200	≤ 1	6/33	0/17	12/35	1/17	9/17	0/21	24.2	[3.5, 169]	0.00001
< 200	≤ 2 *(paper)*	**6/33**	**0/17**	**12/35**	**1/17**	**9/17**	**2/21**	**11.9**	**[3.0, 47]**	**0.00005**
< 200	≤ 3	6/33	0/17	12/35	1/17	10/17	6/21	6.3	[2.1, 19]	0.00069
< 250	≤ 1	6/33	0/17	12/35	1/17	9/17	0/21	24.2	[3.5, 169]	0.00001
< 250	≤ 2	6/33	0/17	12/35	1/17	9/17	2/21	11.9	[3.0, 47]	0.00005
< 250	≤ 3	6/33	0/17	12/35	1/17	10/17	6/21	6.3	[2.1, 19]	0.00069
< 300	≤ 1	6/33	0/17	12/35	1/17	9/17	0/21	24.2	[3.5, 169]	0.00001
< 300	≤ 2	6/33	0/17	12/35	1/17	9/17	2/21	11.9	[3.0, 47]	0.00005
< 300	≤ 3	6/33	0/17	12/35	1/17	10/17	6/21	6.3	[2.1, 19]	0.00069

B.1. T4 ×3 Sonnet 4.6 : Complete AgentRoom Transcript

t=47s [Agent-A]

- CLAIMING: `src/types.ts`, `src/store.ts`, `src/accounting.ts`, `src/currency.ts`, `tests/accounting.test.ts`.
- STATUS: working.

t=75s [Agent-B]

- 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.

t=248s [Agent-A]

- 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()`.

t=372s [Agent-B]

- STATUS: DONE; all 70 tests passing (35 new from B).
- Notes: `src/audit.ts` (`AuditLog` class with SHA-256 hash chain), `src/transactions.ts` (`TransactionStore`, optimistic locking).
- Dev C: import from `src/accounting.ts` for routes.

t=392s [Agent-B]

- Developer B now claiming: `src/reconciliation.ts`, `src/fraud.ts`, `src/routes.ts`, and `src/server.ts` update.

t=577s [Agent-C]

Table 11. 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 the headline.

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

Table 12. 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 ×2 (budget-fair)
Sonnet 4.6	0.544 (n=32)	0.448 (range 0.20–0.62)	**0.669** (n=14)
Haiku 4.5	0.530 (n=35)	**0.604** (range 0.55–0.66)	0.662 (n=17)

- Bug fix in `src/routes.ts`: `/accounts/path/:path(*)` invalid in Express 5; fix changes 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.

B.2. T4 ×4 Sonnet 4.6 : Conflict Resolution

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

t=32s [Developer-3]
- CLAIMING: `src/types.ts`, `src/currency.ts`, `src/transactions.ts`.
- Module: core types, multi-currency, transaction processing.

t=45s [DEV-A]
- CONFLICT RESOLUTION: dropping `types.ts`, `transactions.ts`.
- Revised claim: `src/store.ts`, `src/accounts.ts`, `src/ledger.ts`.

t=48s [Developer-3]
- CONFLICT DETECTED with DEV-A on `types.ts`, `transactions.ts`.
- Revised: `src/currency.ts`, `src/audit.ts`, `src/fraud.ts`.

t=50s [DEV-B]
- CLAIMING: `src/audit.ts`, `src/fraud.ts`, `src/reconciliation.ts`, `src/routes.ts`.
- Note: DEV-A and Developer-3 conflict; please coordinate.

t=60s [Developer-3]
- Re-claiming dropped files. Final claims: `src/types.ts`, `src/currency.ts`, `src/transactions.ts`, `src/audit.ts`, `src/fraud.ts`.
- DEV-A keeps: `store`, `accounts`, `ledger`.

t=120s [DEV-C]
- Final claim (4th developer, no conflicts): `src/reconciliation.ts`, `src/server.ts`, `tests/integration.test.ts`.

t=180s [DEV-C]
- Revised: DEV-B has `reconciliation`; taking `src/errors.ts`, `src/validators.ts`, `src/server.ts`.

t=350s [DEV-C]

Table 13. DevBench cross-domain validation (Haiku 4.5, 600 s budget). *Pass* = hidden pytest suite passes; durations in seconds.

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

- Gap fill: Developer-3 has not created `transactions.ts` yet; to avoid blocking, DEV-C is writing the uncreated files.

t=500s [DEV-A]
- DONE: all 66 tests passing; modules ready for integration.

t=520s [DEV-B]
- DONE: `audit.ts`, `fraud.ts`, `reconciliation.ts`.
- Alert to DEV-C: do not create `audit.ts`, `fraud.ts`; already created by me.

Final result. 65/66 tests passing, 13 source files, 1 CRDT conflict resolved, 14 room messages.

C. 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`. The exact pinned versions are listed in Appendix E. Agents are told the test command, the expected file structure (where prescribed), and the requirements; they are not shown reference solutions.

C.1. T1: JWT Authentication System (Structured, 6 Files)

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.

C.2. T2: Marketplace API (Open-Ended, 10+ Files)

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

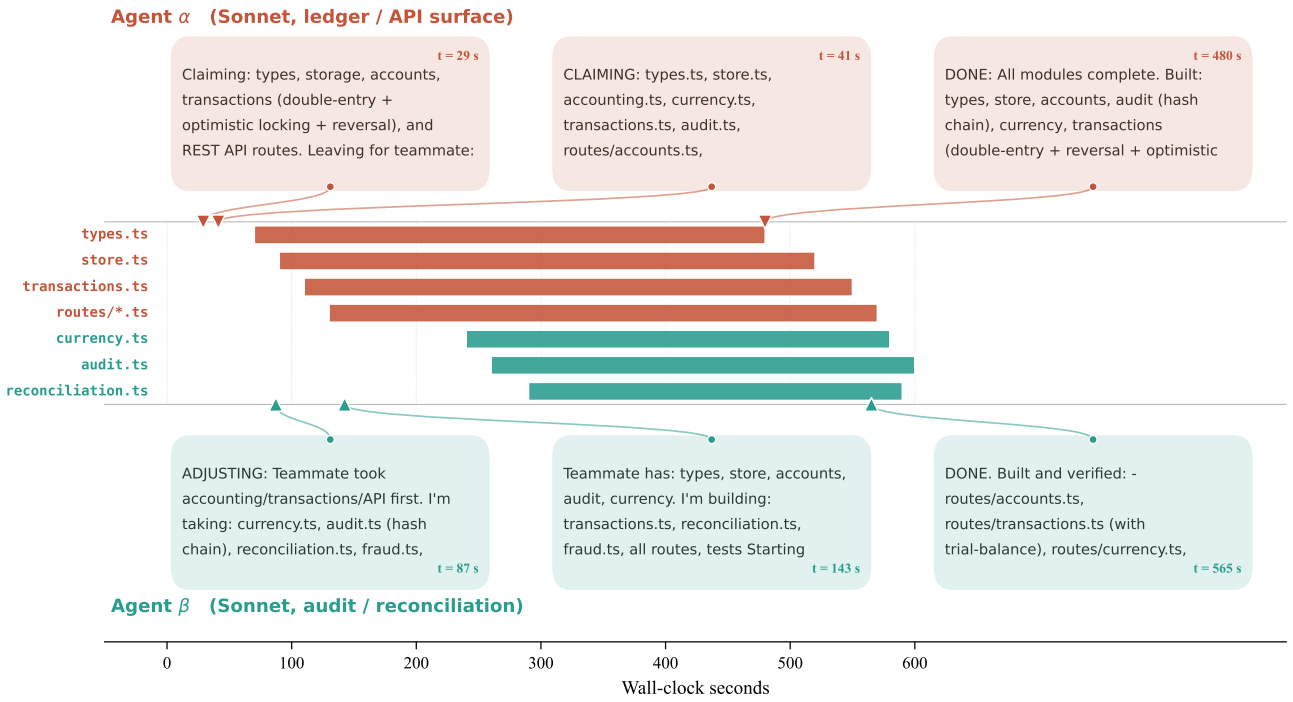

Figure 9. Single representative T4 Sonnet AgentRoom ×2 run. Top swim-lane: agent α broadcasts (timestamps in-card); middle: per-file ownership Gantt (rust = α, teal = β); bottom: agent β. Bezier S-curve leaders link each chat broadcast to its time on the Gantt. The whole protocol (claim, conflict-detect,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

to coordination in the multi-agent setting. 300-second wall-clock budget.

C.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.

C.4. T4: Fintech Ledger (Hard)

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.

D. Quality Scorer Details

LLM-judge (primary scorer for T4 budget-fair). 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. The full $\geq$ 30 s T4 scoring pool (n=282; the 30–700 s analysis pool used for all condition comparisons is a 235-run subset) was scored idempotently, with 14 transient `claude -p` CLI errors retried until convergence (final 282/282). The judge prompt and the four-dimension rubric are reproduced in this appendix.

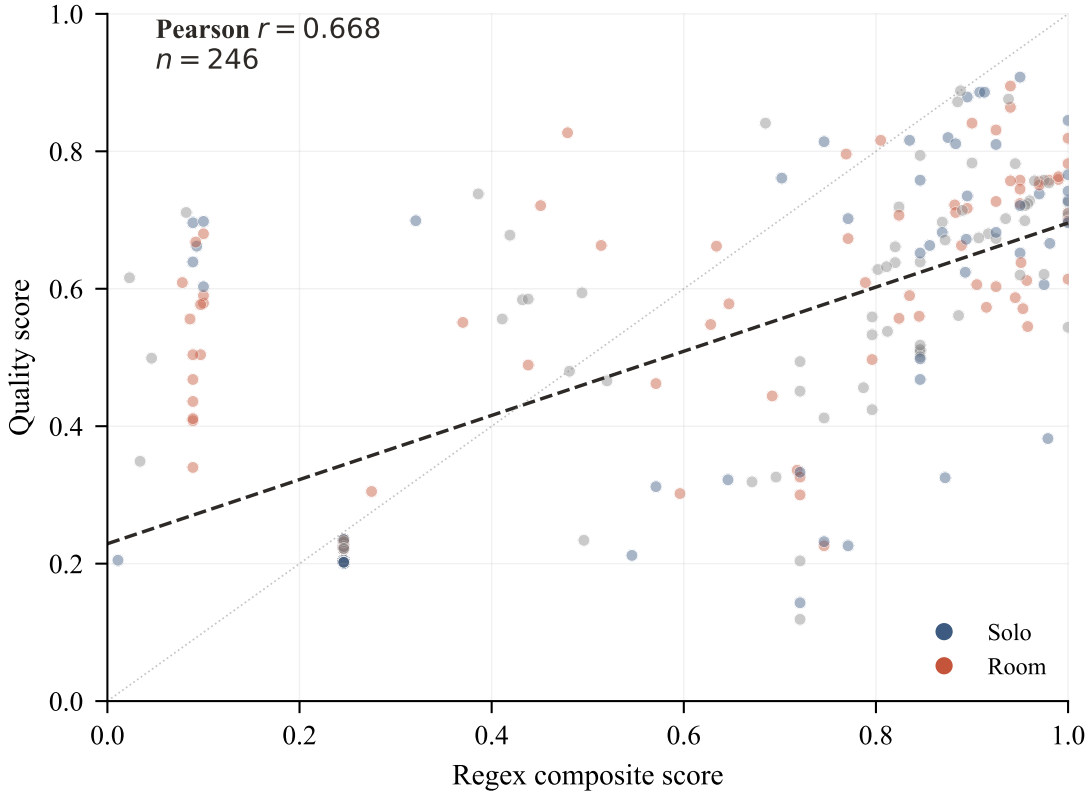

Figure 10. LLM-judge composite vs regex-scorer composite on the full T4 budget-fair pool (n=282); Pearson r=0.67, Spearman ρ=0.62. The two scorers agree on direction across all headline contrasts (CMH OR=13.7, AgentRoom > parallel-merge with p<0.01, AgentRoom > ChatDev with p<0.01).

Cross-validation against regex scorer. We initially used a regex-based intrinsic scorer (described below) and rescored every run with the LLM-judge as a robustness check. Pearson correlation across the full $\geq$ 30 s T4 scoring pool is r=0.67 (n=282; the 30–700 s analysis pool is a 235-run subset); the LLM-judge spreads substrate-only conditions slightly more than the regex scorer and tightens the AgentRoom condition. All headline 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 headline scorer for cells outside the T4 budget-fair pool (cross-model effect-size, T5, difficulty gradient).

Same-family judge concern. 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 headline 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 composition. Five dimensions with fixed weights pre-registered before any multi-agent runs were evaluated: Type Safety (0.25), Defensive Coding (0.25), Test Quality (0.20), Design (0.15), Test Pass Rate (0.15).

AST cross-check. As a robustness check on our regex-based scorer, we implemented an independent AST-based scorer using the TypeScript compiler API. 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 ρ=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 headline 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 round-1 reviewer concern that regex-only scoring could mistakenly reward lexical defensive-coding patterns disconnected from runtime behavior.

Leave-one-out sensitivity (alternative solo subset). Table 14 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 A. 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 14. Scorer sensitivity: leave-one-out on T4 (Sonnet 4.6, solo vs ×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	×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 Section 2.4; 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.

E. Reproducibility and Runtime Environment

Agent CLIs. Each agent is launched as an independent vendor coding-CLI subprocess (Claude Code, Codex CLI, Gemini CLI) in non-interactive mode with vendor-default sampling, attached to a shared AgentRoom MCP server. Pinned CLI versions and model identifier strings are recorded in the per-run logs.

Collaboration prompt (verbatim). The advisory prefix prepended to each agent's task prompt for the AgentRoom and shared-only conditions; the variable {`n_total`} is replaced with the agent count.

> You are one of {`n_total`} developers working simultaneously in a shared workspace.
>
> **Room State Management Tools**
> - `room_claim(path)`: claim a file BEFORE creating or modifying it. Fails if already claimed by another agent.
> - `room_release(path)`: release a file you no longer need to edit.
> - `room_read()`: see new messages and the current state (all claims, all agent statuses).
> - `room_broadcast(message)`: send a coordination message to teammates.
> - `room_state()`: see the full current state snapshot.
>
> **Workflow**

1. Call `room_read()` to see existing claims and teammate status.
2. Call `room_claim(path)` for each file you plan to create.
3. If a claim fails (CONFLICT), pick a different file.
4. Code your claimed files.
5. Periodically call `room_read()` for updates.
6. When done, `room_broadcast` your completion status.

The room enforces file ownership: you cannot claim a file another agent already claimed. Read existing files before writing. Import from teammates' code.

Task *[task spec inserted here]*

CRDT guard. A background process polls the workspace every 200 ms and reconciles any file whose on-disk content changed since the previous poll (content-comparison based, not file-mtime). On a detected collision, both versions are loaded into a `pycrdt` text document (Yrs bindings), merged character-level, and the merged buffer is written back. The sanity check on the merged buffer is the same one referenced in Section 2.5: for files ending in `.ts`, `.js`, `.tsx`, or `.jsx`, we count the difference between opening and closing braces and parentheses; if either difference exceeds ± 2, we discard the merge and keep the longer of the two original versions. We do *not* run `tsc --noEmit` inside the guard because the cost (1–3 s per call on the sandbox) would dominate the 500 ms poll interval; the method-section claim that all observed merges happened to compile under `tsc --noEmit` was verified *post-hoc* on the saved snapshots, not by the live guard.