WHEN SHOULD CONTROL MOVE? PHASE-SCOPED LEADERSHIP FOR EMBODIED LLM TEAMS

Anonymous authors
Paper under double-blind review

ABSTRACT

Embodied LLM teams must coordinate task dependencies while their members retain different observations and procedural memories. A task graph specifies what should happen, but not which agent should construct the next plan. We introduce phase-scoped dynamic leadership: agents exchange compact evidence, jointly select a coordinator and phase objective, execute a phase DAG, and replicate progress across coordination rounds. A locked-leader control preserves the same normal phase-planning loop. We formalize the distinction between sharing sufficient advice and accessing local context by transferring planning authority. We evaluate three-agent teams in Minecraft through an eleven-task, 330-run comparison and a 40-run leadership-trace evaluation. Free-form coordination is economical on simple collection, while locked leadership is the stronger structured default. On three tasks with asymmetric memories, Dynamic achieves 9/30 terminal successes versus Static's 1/30, at 1.55 times the mean LLM-call count. The trace evaluation records 4/20 versus 0/20 successes and directly observes leader changes in 8/20 Dynamic runs. Re-election often retains the same agent, and rotation alone does not ensure success. These findings support phase-scoped authority as a targeted coordination mechanism, while separating observed protocol behavior from causal claims about expertise alignment.

1 INTRODUCTION

Consider a Minecraft team asked to place a torch in a base containing wood and stone but no coal. One agent remembers tool prerequisites; another knows the charcoal production chain. The team must identify a feasible plan, allocate its steps, and verify that a torch was placed. A directed acyclic graph (DAG) can represent the dependencies, but its usefulness depends on the context available to its author. The organizational question therefore extends beyond assigning work: *who should decide what work comes next?*

LLM agents combine language reasoning with tools and embodied actions (Yao et al., 2023; Wang et al., 2023). For teams, graph-based coordination helps manage causal, spatial, and temporal dependencies (Dong et al., 2024), while Minecraft benchmarks expose persistent collaboration difficulties (Long et al., 2024; White et al., 2025). A distributed embodied runtime adds a constraint: each agent has its own observations, memory, and action history. No agent necessarily has a complete, current view of its teammates' procedural context. Centralizing graph updates is useful for coherent execution, but permanently centralizing their authorship can restrict which local knowledge directly informs planning.

We introduce *phase-scoped dynamic leadership*. Agents exchange shared facts and compact statements of procedural suitability, then jointly elect a leader and a phase objective. The elected agent constructs and updates a phase DAG using shared evidence and its local context. Workers act through their own embodied loops, and completed phase results are replicated before the next election. A typed handoff supports recovery when coordination changes during unfinished work. The mechanism is implemented as an independent Collaboration plugin atop the public `minecraft-ai` runtime (Minecraft AI Contributors, 2026) (Figure 1).

The key distinction is between *sharing advice* and *moving planning authority to its holder*. Procedural knowledge is not inherently unshareable. Our candidacy interface deliberately compresses it into short suitability statements; a fixed leader receiving a complete specialist-authored plan could

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

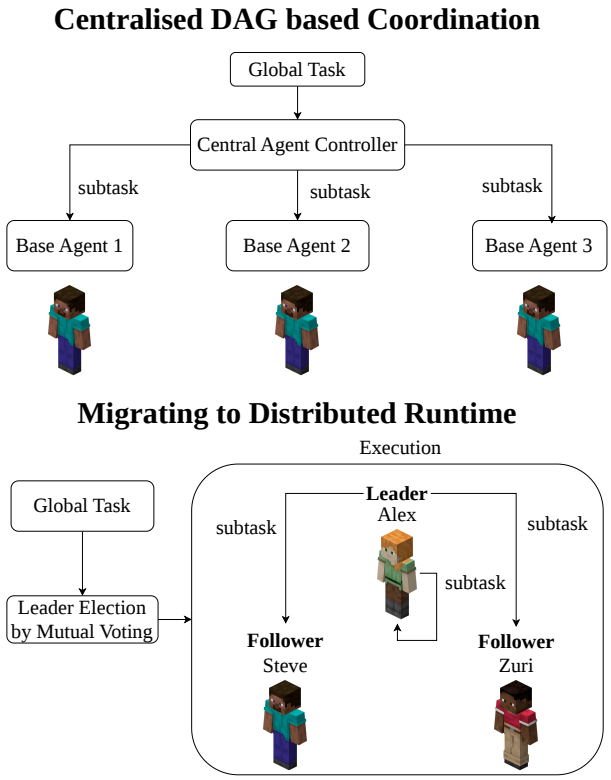

Figure 1: DAG coordination in a distributed embodied runtime. An external controller owns planning in the upper architecture; an elected team member owns the current phase graph in the lower architecture while retaining its embodied execution loop. Names and avatars illustrate roles.

avoid the same bottleneck. A decision-theoretic formulation makes this boundary explicit: authority transfer has no purely informational advantage when the incumbent already has sufficient decision-relevant context. Its benefit must also compensate for coordination cost.

We investigate three questions: when does explicit task structure help; when does allowing leadership to change improve full-protocol outcomes; and does the implemented election actually change the agent holding authority? The evaluation comprises 370 independently reset episodes: 330 runs spanning eleven tasks and three regimes, plus 40 runs comparing the two structured regimes on Torch and Chestplate with leadership records. The results identify a selective benefit rather than universal superiority. Static performs better on most tasks, while Dynamic records more successes on tasks with asymmetric procedural memories. The recorded sequences verify that authority changes occur, but also expose repeated incumbents, failed runs with rotation, and a successful run without an observed change.

Our contributions are: (1) an implementable distributed-DAG protocol with joint phase-and-leader selection, bounded graph authority, phase-progress replication, and validated recovery handoff; (2) an explicit account of the information and cost conditions that motivate transferable authority; and (3) a reproducible evaluation separating task-level outcomes, inference use, and actual leader changes, including all run summaries and all recorded leadership sequences. DAGs and leader election themselves are established ideas; our focus is their integration around phase-dependent, agent-local planning context.

2 Related Work

Embodied teams and task dependencies. MineDojo and Voyager establish Minecraft as a testbed for embodied agency (Fan et al., 2022; Wang et al., 2023). VillagerAgent constructs DAGs for inter-

agent dependencies and evaluates them on VillagerBench (Dong et al., 2024). TeamCraft studies multimodal collaboration and generalization, while MINDcraft/MineCollab investigates action-level collaboration (Long et al., 2024; White et al., 2025). We retain graph-based task organization and study the identity of its temporary author. Our Static regime is an internal architectural control, not a reproduction of VillagerAgent; the evaluation does not establish a ranking on external benchmarks.

Communication and organization. MetaGPT, CAMEL, and AgentVerse organize collaboration through roles and structured dialogue (Hong et al., 2023; Li et al., 2023; Chen et al., 2023); TDAG studies dynamic task decomposition and agent generation (Wang et al., 2025). Organized embodied teams motivate studying organizational structure (Guo et al., 2024). Parallelized planning–acting and Gated Coordination offer complementary approaches through concurrent planning or selective communication (Li et al., 2025; Jian et al., 2026). These approaches can improve information flow without moving graph authority. We specifically distinguish refreshing shared evidence, which both structured regimes perform, from allowing a different embodied agent to construct the next phase graph.

Temporary leadership. Distributed leader election provides precedents for selecting coordinators (Borsche & Attia, 2010; Pianini et al., 2022). Elected leadership has also been studied in LLM social-resource simulations (Faulkner et al., 2026). Our elections bind a coordinator to a semantic phase objective and a concrete embodied execution graph. This is an application-level protocol over a shared chat channel, not partition-tolerant or Byzantine consensus. Time-sensitive complementary collaboration (Yi et al., 2026) further motivates heterogeneous teams, although the present evaluation uses fixed resource layouts rather than a dynamic-event benchmark.

3 Local Context and Planning Authority

Let a team contain N agents and let g be a task with terminal world predicate ϕ_g. Agent i retains context $x_i = (o_i, m_i, h_i)$: observations, procedural memory, and interaction history. At phase k, agents maintain replicated phase history H_k and exchange summaries $s_{i,k} = f_i(x_i, g, H_k)$. Write $z_k = (g, H_k, s_{1,k}, \ldots, s_{N,k})$ for the shared decision context; communication delay can make local copies disagree.

A phase comprises scope p_k, a proposed exit condition, and a DAG $G_k = (V_k, E_k)$ whose nodes describe executor-compatible work and whose edges encode prerequisites. The coordinator ℓ_k constructs the graph through the common LLM planning interface:

$$G_k \sim \Pi_\theta(\cdot \mid z_k, x_{\ell_k}, p_k). \tag{1}$$

All agents use the same model; changing ℓ_k changes the context directly available to the planner, not its trained parameters. Static imposes $\ell_k = \ell_1$ for the entire task. Dynamic can change ℓ_k, while both regimes can revise the phase and construct a fresh graph.

3.1 Information Sufficiency

Fix a phase, shared context z, latent task-relevant state W, and a finite common set of feasible plans $\mathcal{D}$. For bounded loss $L(d, W) \in [0, 1]$ and agent-local information X_i, define ideal decision risks

$$\begin{aligned} R_i(z) &= \mathbb{E}_{X_i|z}\left[\min_{d\in\mathcal{D}} \mathbb{E}[L(d, W) \mid z, X_i]\right], \\ R_{\text{shared}}(z) &= \min_{d\in\mathcal{D}} \mathbb{E}[L(d, W) \mid z]. \end{aligned} \tag{2}$$

These quantities isolate information access under ideal decisions; the implemented LLM does not estimate them.

Sufficiency condition. Additional information cannot worsen ideal risk: $R_i(z) \le R_{\text{shared}}(z)$, since the planner can ignore it. If $W \perp X_i \mid z$, then $R_i(z) = R_{\text{shared}}(z)$. The same equality holds when local context leaves all relevant conditional losses unchanged. Thus, equally capable planners with sufficient shared advice gain no informational advantage simply by changing identity. Appendix A states the assumptions and proofs.

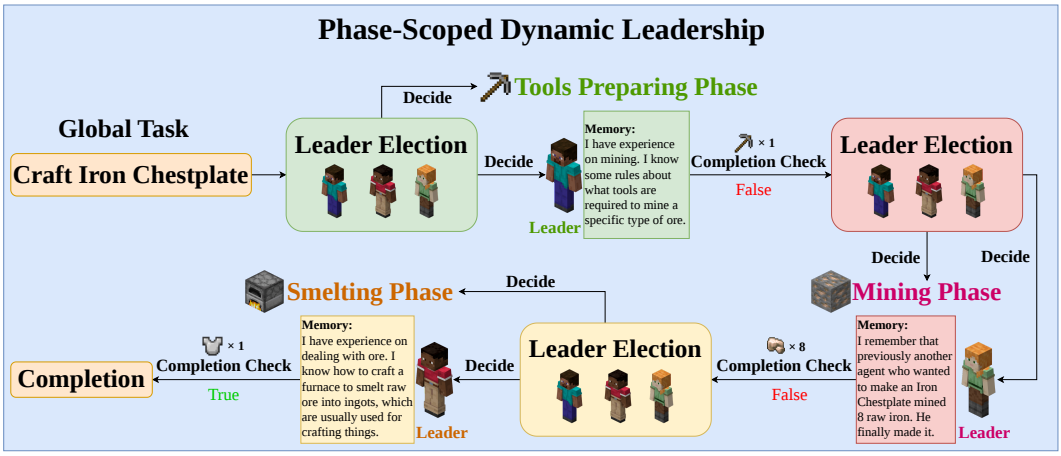

Figure 2: Phase-scoped dynamic leadership illustrated by the Iron Chestplate chain: tool preparation, mining, smelting, and final crafting. Elections bind a coordinator to a phase; False denotes an incomplete global goal after a phase. This is a mechanism schematic; Figure 3 reports measured leadership sequences.

An insufficient-summary example. Suppose two equiprobable states require different first steps, but an incumbent receives the same summary in both. Its error is at least $1/2$. A specialist whose local context identifies the state can choose correctly; appointing that specialist accesses the missing information. However, sending the identifying bit to the incumbent removes the advantage. This example motivates testing *the communication interface and leader choice together*, rather than assuming procedures cannot be transmitted. It does not establish that the particular character limit used here is insufficient for a Minecraft task.

3.2 When Is Transfer Worth Its Cost?

For incumbent ℓ, candidate i, additional cost $\Delta C_{\ell \to i}(z) \geq 0$, and conversion $\lambda \geq 0$ from cost to loss units, transferring authority improves the fixed-phase objective exactly when

$$R_\ell(z) - R_i(z) > \lambda \Delta C_{\ell \to i}(z). \tag{3}$$

This accounting relation motivates three testable expectations: sufficient shared advice should reduce the advantage of rotation; selecting phase-relevant context should matter more than arbitrary rotation; and benefits must be weighed against additional coordination and inference. It is not a long-horizon optimality theorem or the rule optimized by our election. The experiment tests protocol outcomes and realized authority changes; it does not measure the ideal risks or independently annotate expert alignment.

4 Phase-Scoped Coordination

4.1 Evidence Refresh and Joint Election

Each agent retains an independent embodied loop. Coordination uses typed JSON messages over Minecraft chat, fragmented above the configured message limit. At a phase boundary, agents solicit candidacies with local state, a shared-information statement, and a leadership-advantage statement. Each statement is capped at 240 characters. Shared information conveys facts such as inventory and resource locations; advantage describes procedural suitability for the immediate phase. Its prompt discourages fully spelling out recipes, so the interface supplies compressed advice rather than a lossless memory copy.

Each voter receives the global task, phase history, and candidate information, and proposes a leader together with a phase name, goal, rationale, and exit condition. The winner synthesizes the associated phase proposals into an executable scope. Terms distinguish election rounds. Vote ties are

Mechanism	Baseline	Static	Dynamic
Agent-local memory and executor	Yes	Yes	Yes
Phase evidence refresh and synthesis	No	Yes	Yes
Phase DAG and typed assignments	No	Yes	Yes
Later phase can elect another leader	No leader	No	Yes
Heartbeat failover and DAG handoff	No	No	Yes

Table 1: The three evaluated regimes. Static and Dynamic share the normal phase loop but differ in both authority transfer and monitoring/failover. Neither provides a dedicated interface for a non-leader to submit a complete phase DAG to a fixed leader.

Algorithm 1 Phase-scoped DAG coordination

Require: task g, mode $M \in \{\textsc{Static}, \textsc{Dynamic}\}$, deadline T
Initialize phase history $H \leftarrow \emptyset$; locked leader $\ell_0 \leftarrow \emptyset$
while task has not terminated and deadline has not elapsed **do**
  Collect candidate state, shared facts, and procedural-suitability claims
  Select phase p and leader ℓ; constrain $\ell = \ell_0$ if Static and initialized
  Set $\ell_0 \leftarrow \ell$ if uninitialized; ℓ constructs phase DAG G
  while G remains unfinished and task has not terminated **do**
    Assign ready nodes; workers execute and report completion/failure judgments
    Update the current graph; recover failed nodes when possible
    if $M = \textsc{Dynamic}$ and leader-failure detection triggers **then**
      Re-elect; restore a valid handoff if available, otherwise reconstruct from H
    end if
  end while
  Check global completion; if incomplete, broadcast phase result and update H
end while

resolved by $b(i) = 5\,\mathrm{health}_i + 2\min(\mathrm{food}_i, 20) + 2|\mathrm{inventoryEntries}_i|$, then lexical agent ID. This deterministic fallback is not a learned expertise score. Candidate collection lasts ten seconds, and tallying waits three seconds after the first vote.

Static elects its initial leader and constrains subsequent ballots to the same winner; evidence refresh and phase synthesis still occur. Dynamic leaves later leader choices unconstrained. Re-election therefore *permits*, but does not require, a new leader. Figure 2 illustrates how the graph author can change while other agents continue executing prerequisite work.

4.2 Execution and Recovery

The phase leader owns the active graph and assigns ready nodes whose predecessors are marked complete. Assignment is least-loaded by default, with optional preferred-agent hints. Structured subtasks pass through one `handleMessage` iteration, so descriptions should specify directly executable single-command work. Baseline instead gives every agent the full global task and permits ordinary chat and up to five responses per invocation. Its comparison with the structured regimes measures whole coordination systems, not graph structure alone.

A worker's LLM checker judges node completion from the task, actions, outputs, and state. Positive reports update the graph and release successors; negative reports can trigger retry, reassignment, instruction revision, node splitting, or rolling replanning. Recovery limits are sufficiently large that the global deadline is the practical bound. Node success is a control signal, not independent proof of the terminal world predicate. A separate global check determines whether another phase is needed.

Dynamic also sends heartbeats and monitors its leader. A timeout can trigger re-election within unfinished work. A responsive outgoing leader can transfer typed DAG state; a successor can restore an accepted preceding-term snapshot or reconstruct from replicated phase history. Abrupt failure cannot provide a fresh handoff and may lose unreplicated progress. Recovery-stall re-election is a separate option and is disabled in the experiments; heartbeat failover remains enabled.

4.3 Consistency Boundaries

Within an agreed term, the leader is the intended single writer of the active graph. Election messages carry terms. Task completion/failure handlers check active-node membership and reject a *supplied* attempt ID that differs from the active attempt; a compatibility path accepts a missing ID. Consequently, the implementation does not guarantee rejection of every stale report or consensus under arbitrary delay.

Handoff validation binds task identity, envelope/payload term, DAG version, node/attempt references, assignments, and completion records. Restoration requires the immediately preceding term and preserves the logical graph version. These checks establish structural consistency, not correctness of the LLM's world-state judgments. Completed phases use compact progress replication rather than restoration of an active DAG. Appendix F supplies schemas and decision prompts; Appendix G gives runtime parameters.

5 Experimental Design

Tasks and two complementary evaluations. The task-suite evaluation has eleven tasks, three regimes, and ten runs per task–regime cell (330 episodes). Class A covers single-resource collection, B independent resources, and C dependency-sensitive collection/crafting. Class D has three multistage tasks with asymmetric procedural memories: Craft Stone requires smelting cobblestone into the inventory item `stone`; Place Torch requires a torch placed within a 10×10 wood-and-stone base without coal; Iron Chestplate requires the tool–mining–smelting–crafting chain for eight iron ingots.

The leadership-trace evaluation has ten runs per task and structured regime on Torch and Chestplate (40 episodes). It uses the same tasks, memories, model/API configuration, manual reset procedure, and success criteria, with election winners retained in the summaries. These two tasks are selected diagnostic cases, not a random or held-out task sample. Baseline has no elected leader and is evaluated in the task suite. We report the two evaluations separately, preserving their collection strata rather than treating all repetitions as one randomized trial.

Agent-local memories. All agents share a model backend, so asymmetry is an intervention on accessible context, not disjoint pretrained knowledge. Craft Stone gives Lucy tool/mining instructions, Max furnace/smelting instructions, and Emily a null memory. Torch gives Lucy wood/tool knowledge and Max cobblestone/charcoal prerequisites. Its seed stores the intended third specialist under an unmatched `Agent3` key; Emily therefore receives the default memory. Chestplate distributes tool constraints, the eight-raw-iron requirement, and smelting instructions across Lucy, Max, and Emily. The actual mapping is identical across regimes and evaluations within each task. Appendix E includes verbatim seeds and loader semantics.

Environment and controls. We use three agents in Minecraft 1.21.1, a superflat seed-42 world, and a 600-second soft deadline. Before each episode, an experimenter resets the world and task materials manually. The reported common backend is DeepSeek-V4-Flash in non-thinking mode through the official API, using the configured `deepseek-chat` request identifier (DeepSeek, 2026). Sampling parameters use provider defaults; per-response model fingerprints are not retained, so configuration identity does not establish an immutable service revision. Experiments are neither paired nor randomized by a recorded block design. Initial leaders and total LLM calls are not matched. Dynamic–Static contrasts therefore concern complete protocols, including their monitoring differences.

Outcomes and analysis. An experimenter continuously observes each episode and retains success only when the exact global goal is visible in the client. Monitoring is unblinded; screenshots and terminal inventory/world dumps are not retained. For binary outcome y_r, task completion rate is $\mathrm{TCR} = n^{-1} \sum_r y_r$. We report successes out of runs and mean total LLM calls. The `ttc_seconds` field measures *termination duration*, including failures; it is not time-to-success. Subtask success rate (SSR) measures internal node judgments. Neither shorter failed runs nor higher SSR establish better global completion.

	Baseline		Static		Dynamic	
Class	Successes	Calls	Successes	Calls	Successes	Calls
A	29/30	8.2	28/30	15.7	26/30	107.1
B	8/30	16.3	26/30	19.6	23/30	62.9
C	2/20	13.9	17/20	52.5	15/20	170.6
D	0/30	53.8	1/30	135.9	9/30	210.0

Table 2: Task-suite evaluation: all 330 episodes. Calls are per-run means. A: single-resource collection; B: independent resources; C: dependency-sensitive tasks; D: asymmetric memories. Each task contributes ten runs per regime.

	Baseline		Static		Dynamic	
Task	Successes	Calls	Successes	Calls	Successes	Calls
Craft Stone	0/10	27.7	1/10	282.7	2/10	361.6
Place Torch	0/10	85.9	0/10	93.1	4/10	130.3
Iron Chestplate	0/10	47.7	0/10	31.8	3/10	138.2

Table 3: Class-D task-suite results. Every cell contains ten episodes. Calls count all inference requests, without matching a common call or token allowance.

Analysis uses 95% Wilson intervals and two-sided exact conditional Dynamic–Static comparisons stratified by task within each evaluation. These are exploratory, nominal calculations under independently reset episodes; they do not control for service changes, execution order, or manual-reset drift. No pairing is inferred from identifiers. All 370 experiment IDs are unique; one task ID is reused in two distinct Static records, so grouping uses experiment IDs. The release includes every measured summary, input hashes, and self-contained analysis code.

6 Results

6.1 Task Structure Helps Before Movable Authority Does

Table 2 shows that free-form coordination is effective and economical in Class A: 29/30 successes at 8.2 calls per run. Static achieves 28/30 at 15.7 calls; Dynamic achieves 26/30 at 107.1 calls. Extra coordination is expensive when a task has little need for it. Across B–C, Static succeeds in 43/50 episodes, versus Baseline's 10/50 and Dynamic's 38/50. This supports structured coordination as a useful default for dependencies, without identifying which of decomposition, assignment, judgment, or executor invocation policy produces the gain.

Dynamic is not the stronger general regime. On wooden tools, its SSR is higher than Static's (0.872 versus 0.743), but terminal successes are lower (6/10 versus 7/10), with 227.4 rather than 64.9 calls. Locally judged graph progress can diverge from the requested world outcome. Complete task-level results, durations, and SSR are in Appendix B.

6.2 Asymmetric Memories Change the Observed Trade-off

On Class D, Dynamic records 9/30 successes, Static 1/30, and Baseline 0/30 (Table 3). The Dynamic–Static difference is 26.7 percentage points, with nominal Wilson intervals $[0.167, 0.479]$ and $[0.006, 0.167]$. The task-stratified exact two-sided value is $p = 0.01365$. This is evidence of a full-protocol association on the selected tasks, not an isolated effect of rotation. Each task has only ten runs per regime: even zero successes has a Wilson upper bound of 0.278. Leaving out Torch or Chestplate increases the nominal pooled p above 0.05 (Appendix B.1), illustrating limited support across tasks.

Dynamic uses 210.0 calls per Class-D episode versus Static's 135.9, a ratio of 1.55. Task-level ratios range from 1.28 for Stone to 4.35 for Chestplate. Thus the observed completion gain comes with

Task	Static Successes	Static Calls	Dynamic Successes	Dynamic Calls
Place Torch	0/10	207.4	2/10	293.5
Iron Chestplate	0/10	113.1	2/10	140.9

Table 4: The 40-episode leadership-trace evaluation. Task, memory, environment, timeout, and outcome protocol are the same across regimes. Logging retains Dynamic's election winner, term, and reason.

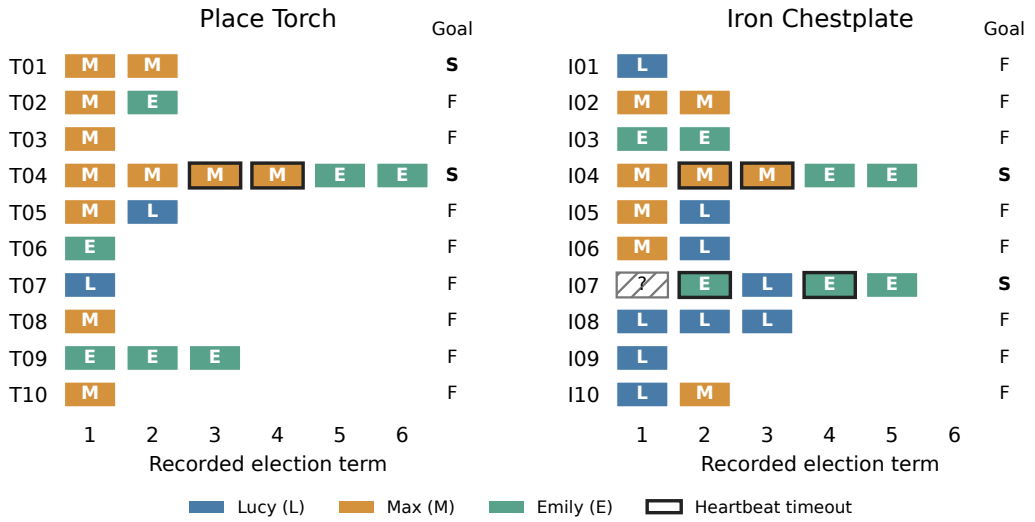

Figure 3: All 20 Dynamic leadership sequences in the trace evaluation. Each colored cell is a recorded election winner; letters denote Lucy, Max, or Emily. Black outlines indicate heartbeat-timeout elections. S/F denotes global success/failure. Horizontal position is election term, not duration or a semantic task phase. The hatched cell marks the unrecorded first term of I07. Row labels map to full experiment IDs in Appendix I.

additional inference, and the comparison cannot establish compute efficiency. Calls are not split reliably by election, planning, execution, judgment, and recovery.

6.3 Leadership-Trace Outcomes

Dynamic achieves 2/10 successes on each task; Static achieves 0/10 on each (Table 4). The combined difference is 20 percentage points, with Wilson intervals $[0.081, 0.416]$ for Dynamic and $[0, 0.161]$ for Static. The task-stratified exact value is $p = 0.11219$; these 40 episodes alone do not establish a conventionally significant performance difference. Mean calls are 217.2 versus 160.25, a 1.36 ratio. The direction agrees with the task-suite result, while both absolute success and call use vary across evaluations. The contribution of this evaluation is direct measurement of realized leadership behavior, alongside its outcomes.

6.4 Re-election Is Not the Same as Rotation

The exporter deduplicates election-end observations by term and winner. Its `phase_k_leader` keys enumerate these records; they do not independently label task phases. Figure 3 contains all 44 records. For recorded winners $\ell_{r,1}, \ldots, \ell_{r,K_r}$, we count actual observed changes as

$$S_r = \sum_{j=2}^{K_r} \mathbf{1}[\ell_{r,j} \neq \ell_{r,j-1}]. \tag{4}$$

There are 24 adjacent recorded transitions: nine change the leader and fifteen retain the incumbent. Eight of twenty runs contain at least one observed change (Torch 3/10; Chestplate 5/10). The sequences directly verify that different embodied agents can hold planning authority within the same episode.

Transition reasons further separate normal phase transfer from failover. Nineteen adjacent transitions are marked `phase_boundary`; eight change identity. Five are marked `heartbeat_timeout`; one changes identity. Thus eight of nine observed changes occur at phase boundaries, while timeout elections often return the incumbent. One Chestplate sequence starts at term 2 following a timeout, so its preceding leader and initial transition are unknown. Counts concern recorded adjacent transitions and can undercount full-episode changes.

Three of four Dynamic successes contain an observed leader change; the successful Torch run T01 retains Max across two rounds. Five of eight runs with a change still fail. These descriptions demonstrate that observed rotation alone is neither necessary nor sufficient for success in this sample; they are not a causal comparison between runs selected by post-treatment behavior. The summaries lack semantic phase goals, elapsed-time traces, and designated-expert annotations, so they cannot establish that the elected agent possessed the most relevant memory. They also cannot turn handoff-send counts into successful restoration rates.

7 Discussion and Limitations

What the evidence establishes. A phase DAG and its temporary author are distinct design choices. The task suite favors inexpensive free-form coordination for simple collection and locked DAG authority for many dependency tasks. On asymmetric-memory tasks, Dynamic has higher observed success at greater inference cost. The leadership records verify phase-boundary identity changes and show why election counts are an inadequate proxy. Together, these results motivate selective authority transfer without claiming that it should replace fixed leadership universally.

What remains unidentified. The Dynamic–Static contrast includes monitoring and failover, unequal calls, and uncontrolled initial leaders. A common initial leader, matched inference allowance, and identical failover policy would isolate normal-boundary authority changes more directly. A fixed leader receiving complete specialist-authored plans would test the competing advice-sharing explanation in Section 3.1. Those controls are not among the evaluated regimes. The intentionally compact suitability channel and prompt-induced memory asymmetry are part of the experimental setting, not unavoidable constraints of LLM teams.

Scope and reproducibility. The study uses three agents, one model configuration, fixed task layouts, and manual unblinded outcome checks. It does not establish generalization across worlds, team sizes, or model families. Reset snapshots, terminal world evidence, per-response model revisions, and full execution traces are unavailable; recorded leadership is restricted to the twenty Dynamic trace episodes. Protocol checks constrain state structure but do not certify physical outcomes or consensus under partitions. The supplied data and scripts reproduce the quantitative analysis, while an exact environment replay would require additional runtime evidence.

8 Conclusion

Phase-scoped dynamic leadership lets an embodied team reconsider who constructs its next plan while preserving structured DAG execution. Its value depends on the relevance of agent-local context and the cost of transferring authority. Across 370 episodes, the evidence supports locked coordination as a strong default and identifies asymmetric-memory tasks as a setting where the complete Dynamic protocol records more successes. Direct leadership measurements verify that phase-boundary transfers occur, often interspersed with re-election of the same agent. The resulting design lesson is to evaluate task structure, authority allocation, and information sharing separately, and to measure actual leader changes rather than infer them from election or handoff activity.

AI USE STATEMENT

Generative AI assisted literature search, development of the conceptual information-access framework, formulation and checking of mathematical arguments and proofs, feedback on experimental methodology, code debugging, cleaning and reformatting metric files, statistical-analysis code, figure preparation, interpretation of results, and manuscript drafting and editing. The experimental records are measurements from manually conducted Minecraft episodes; AI assistance in manuscript preparation did not synthesize or replace their outcomes or leadership sequences. LLM calls made by the evaluated agents are part of the research method. The human authors retain responsibility for the experimental record and the final text, claims, references, mathematical arguments, code, and disclosures.

REPRODUCIBILITY STATEMENT

Appendices D–G specify task objectives, memory contents and loading, decision prompts, message schemas, and protocol parameters. Appendix A gives the information-access proofs; Appendices B.1 and I define the statistical calculations and leadership parser. The anonymous supplementary artifact includes all 370 metric CSVs, all 44 leader records, input hashes, analysis scripts, and the coordination implementation. The manuscript source independently compiles and regenerates tables and figures from its included data. These materials reproduce the reported analysis; missing terminal world snapshots and API response revisions limit exact execution replay.

ETHICS STATEMENT

The study concerns artificial agents in a Minecraft environment. The researchers conduct outcome monitoring, and the tasks do not involve collecting personal data from human participants. The principal integrity considerations are distinguishing internal LLM judgments from observed global outcomes and reporting the limits of protocol attribution. The anonymous artifact excludes credentials, identifying repository metadata, and private conversation logs. Inference use is reported as request counts because token-level energy and monetary costs were not measured.

REFERENCES

Theodor Borsche and Sid Ahmed Attia. On leader election in multi-agent control systems. In *2010 Chinese Control and Decision Conference*, pp. 102–107, 2010. doi: 10.1109/CCDC.2010.5499115.

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.

DeepSeek. DeepSeek-V4 preview release. https://api-docs.deepseek.com/news/news260424/, 2026. Official release and compatibility mapping, April 24, 2026. Accessed: 2026-09-23.

Yubo Dong, Xukun Zhu, Zhengzhe Pan, Linchao Zhu, and Yi Yang. Villageragent: A graph-based multi-agent framework for coordinating complex task dependencies in minecraft. In *Findings of the Association for Computational Linguistics ACL 2024*, pp. 16290–16314, 2024.

Linxi Fan, Guanzhi Wang, Yunfan Jiang, Ajay Mandlekar, Yuncong Yang, Haoyi Zhu, Andrew Tang, De-An Huang, Yuke Zhu, and Anima Anandkumar. Minedojo: Building open-ended embodied agents with internet-scale knowledge. In *Thirty-sixth Conference on Neural Information Processing Systems (NeurIPS) Datasets and Benchmarks Track*, 2022.

Ryan Faulkner, Anushka Deshpande, David Guzman Piedrahita, Joel Z. Leibo, and Zhijing Jin. Evaluating cooperation in llm social groups through elected leadership, 2026. URL https://arxiv.org/abs/2604.11721.

Xudong Guo, Kaixuan Huang, Jiale Liu, Wenhui Fan, Natalia Vélez, Qingyun Wu, Huazheng Wang, Thomas L. Griffiths, and Mengdi Wang. Embodied llm agents learn to cooperate in organized teams, 2024. URL https://arxiv.org/abs/2403.12482.

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. *arXiv preprint arXiv:2308.00352*, 2023.

HuaDong Jian, Chenghao Li, Haoyu Wang, Jiajia Shuai, Jinyu Guo, Yang Yang, and Chaoning Zhang. Gated coordination for efficient multi-agent collaboration in minecraft game, 2026. URL https://arxiv.org/abs/2604.18975.

Guohao Li, Hasan Abed Al Kader Hammoud, Hani Itani, Dmitrii Khizbullin, and Bernard Ghanem. Camel: Communicative agents for "mind" exploration of large language model society, 2023. URL https://arxiv.org/abs/2303.17760.

Yaoru Li, Shunyu Liu, Tongya Zheng, Li Sun, and Mingli Song. Parallelized planning-acting for efficient llm-based multi-agent systems in minecraft, 2025. URL https://arxiv.org/abs/2503.03505.

Qian Long, Zhi Li, Ran Gong, Ying Nian Wu, Demetri Terzopoulos, and Xiaofeng Gao. Teamcraft: A benchmark for multi-modal multi-agent systems in minecraft, 2024. URL https://arxiv.org/abs/2412.05255.

Minecraft AI Contributors. Minecraft AI: Toward embodied turing test through ai characters. https://github.com/aeromechanic000/minecraft-ai, 2026. Software framework. Accessed: 2026-06-19.

Danilo Pianini, Roberto Casadei, and Mirko Viroli. Self-stabilising priority-based multi-leader election and network partitioning. In *2022 IEEE International Conference on Autonomic Computing and Self-Organizing Systems (ACSOS)*, pp. 81–90, 2022. doi: 10.1109/ACSOS55765.2022.00026.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models. *arXiv preprint arXiv:2305.16291*, 2023. URL https://arxiv.org/abs/2305.16291.

Yaoxiang Wang, Zhiyong Wu, Junfeng Yao, and Jinsong Su. TDAG: A multi-agent framework based on dynamic Task Decomposition and Agent Generation. *Neural Networks*, 185:107200, May 2025. ISSN 08936080. doi: 10.1016/j.neunet.2025.107200. URL https://linkinghub.elsevier.com/retrieve/pii/S0893608025000796.

Isadora White, Kolby Nottingham, Ayush Maniar, Max Robinson, Hansen Lillemark, Mehul Maheshwari, Lianhui Qin, and Prithviraj Ammanabrolu. Collaborating action by action: A multi-agent llm framework for embodied reasoning, 2025. URL https://arxiv.org/abs/2504.17950.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. ReAct: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=WE_vluYUL-X.

Juheon Yi, Jinglu Wang, Xiaoyi Zhang, and Yan Lu. Multi-agent framework for time-sensitive complementary collaboration in minecraft, 2026. URL https://arxiv.org/abs/2606.15684.

A Information Access: Assumptions and Proofs

This section formalizes only the fixed-phase information argument in Section 3.1. The facts below are elementary consequences of conditional decision risk, included to make the mechanism hypothesis precise. They are not claims of theoretical novelty, learned-policy optimality, or correctness of the distributed protocol.

Assumptions. Fix a shared context z having positive probability. All candidate leaders choose from the same finite, nonempty set of plans $\mathcal{D}$; the loss $L(d, W)$ is bounded and measured on the same phase state W. Decisions minimize the conditional expected loss exactly. Leadership changes do not alter the phase, world dynamics, decision set, or execution capability within this comparison. Local information is available before choosing the plan. For general non-discrete contexts the same statements hold almost surely using regular conditional distributions.

Proposition 1 (sufficient shared context). If $W \perp X_i \mid z$, then $R_i(z) = R_{\text{shared}}(z)$.

Proof. Conditional independence gives, for every $d \in \mathcal{D}$ and almost every x_i,

$$\mathbb{E}[L(d, W) \mid z, X_i = x_i] = \mathbb{E}[L(d, W) \mid z].$$

The minimand is therefore independent of x_i. Taking the minimum over the same finite decision set and then the expectation over $X_i \mid z$ gives

$$R_i(z) = \mathbb{E}_{X_i|z}[\min_d \mathbb{E}[L(d, W) \mid z]] = \min_d \mathbb{E}[L(d, W) \mid z] = R_{\text{shared}}(z).$$

Conditional independence is sufficient, not necessary: equality also holds when a shared-context optimal plan remains conditionally optimal for almost every local observation, as characterized below. □

Proposition 2 (additional information cannot worsen ideal risk). Under the assumptions above, $R_i(z) \le R_{\text{shared}}(z)$.

Proof. Choose a shared-context optimal plan $d_z \in \arg\min_d \mathbb{E}[L(d, W) \mid z]$. An agent with additional information can always ignore it and use d_z. Hence

$$\begin{aligned} R_i(z) &= \mathbb{E}_{X_i|z}[\min_d \mathbb{E}[L(d, W) \mid z, X_i]] \\ &\le \mathbb{E}_{X_i|z}[\mathbb{E}[L(d_z, W) \mid z, X_i]] \\ &= \mathbb{E}[L(d_z, W) \mid z] = R_{\text{shared}}(z), \end{aligned}$$

where the last equality uses conditional iterated expectation. □

More precisely, writing $q(d, x) = \mathbb{E}[L(d, W) \mid z, X_i = x]$, the value of the additional context is

$$R_{\text{shared}}(z) - R_i(z) = \mathbb{E}_{X_i|z}\left[q(d_z, X_i) - \min_d q(d, X_i)\right].$$

The integrand is nonnegative. Equality holds if and only if the shared-optimal d_z is also optimal for almost every local observation. Thus extra context can be statistically informative yet have no decision value when it never changes the appropriate plan.

This inequality does not compare two agents with incomparable information sets, nor does it ensure that an actual LLM will ignore distracting information. It cannot justify appointing a particular candidate without evidence about that candidate's available information and decision quality.

Counterexample under an insufficient summary. Let W be uniform on $\{0, 1\}$, let $\mathcal{D} = \{d_0, d_1\}$, and set $L(d_j, W) = \mathbf{1}[j \neq W]$. The shared context z is constant. An incumbent with no further information has risk $1/2$, even if it randomizes. A specialist with $X_i = W$ has risk zero. If the suitability channel reliably identifies which candidate is the specialist, appointing that agent can access the missing information without transmitting W through the summary. The availability of that reliable identification is an explicit assumption; the implemented LLM election does not guarantee it.

Sending W itself to the incumbent makes its risk zero as well. Thus this example supports a conditional bottleneck argument, not a general advantage of transferring authority over transmitting advice. It does not establish that 240 characters are insufficient for any task in the experimental suite.

Cost accounting and scope. For $J_i(z) = R_i(z) + \lambda C_i(z)$ and $C_i - C_\ell = \Delta C_{\ell \to i} \geq 0$, subtraction gives $J_i < J_\ell$ if and only if equation 3 holds. This is an accounting identity under the stated objective. It supplies a threshold for a design hypothesis; it is not a fitted cost model. In the implemented system, an election can change the proposed phase, later execution can alter future information, and recovery affects the rest of the episode. Those effects violate the fixed-phase abstraction unless included in a richer loss. No bound on total reward, regret, or terminal success follows here.

B Task-Level Results and Statistical Analysis

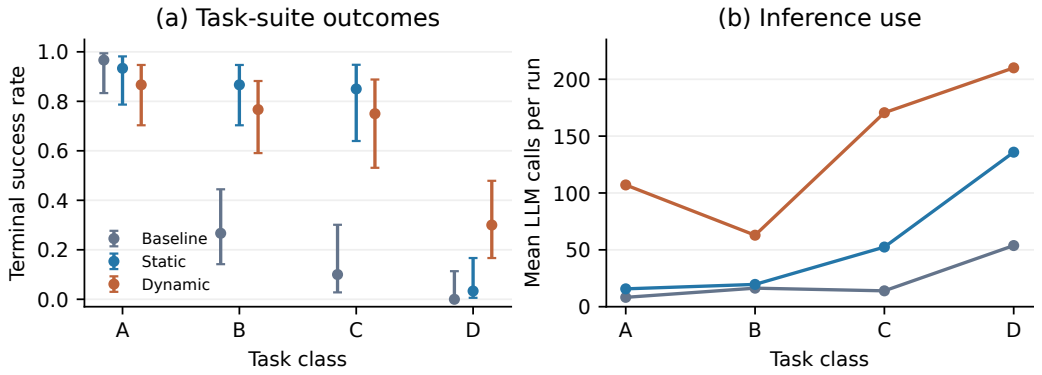

Figure 4: Task-suite outcomes and inference use. Error bars are nominal 95% Wilson intervals; A/B/D have 30 episodes per regime and C has 20. Each task contributes equally within its class.

Table 5 reports all evaluation–task–regime cells. TTC is termination duration, including failures. SSR is the retained per-run LLM-judged subtask rate; it is undefined for Baseline. Continuous entries are arithmetic means over ten episodes, not pooled-node rates or success-only averages. Suite and Trace denote the task-suite and leadership-trace evaluations.

Table 5: All 37 evaluation–task–regime cells.

Eval.	Class	Task	Mode	Successes	TTC	SSR	Calls
Suite	A	Gather dirt	Baseline	10/10	12.9	–	5.5
Suite	A	Gather dirt	Static	10/10	71.2	1.000	11.0
Suite	A	Gather dirt	Dynamic	9/10	199.3	0.930	95.3
Suite	A	Gather oak	Baseline	9/10	36.3	–	12.0
Suite	A	Gather oak	Static	10/10	89.8	1.000	15.0
Suite	A	Gather oak	Dynamic	8/10	267.7	0.841	85.5
Suite	A	Gather sand	Baseline	10/10	25.5	–	7.2
Suite	A	Gather sand	Static	8/10	149.4	0.800	21.1
Suite	A	Gather sand	Dynamic	9/10	259.0	0.851	140.4
Suite	B	Wood + dirt	Baseline	4/10	312.0	–	14.1
Suite	B	Wood + dirt	Static	9/10	211.8	0.900	18.5
Suite	B	Wood + dirt	Dynamic	7/10	416.8	0.857	112.1
Suite	B	Oak + leaves	Baseline	3/10	430.4	–	14.2
Suite	B	Oak + leaves	Static	9/10	166.9	0.700	16.4
Suite	B	Oak + leaves	Dynamic	9/10	388.5	0.967	32.3
Suite	B	Oak + sand	Baseline	1/10	543.3	–	20.6
Suite	B	Oak + sand	Static	8/10	226.3	0.800	23.9
Suite	B	Oak + sand	Dynamic	7/10	377.0	0.840	44.2
Suite	C	Build materials	Baseline	2/10	415.6	–	15.2

Eval.	Class	Task	Mode	Successes	TTC	SSR	Calls
Suite	C	Build materials	Static	10/10	244.1	1.000	40.0
Suite	C	Build materials	Dynamic	9/10	303.5	0.887	113.8
Suite	C	Wooden tools	Baseline	0/10	439.7	–	12.7
Suite	C	Wooden tools	Static	7/10	524.2	0.743	64.9
Suite	C	Wooden tools	Dynamic	6/10	556.3	0.872	227.4
Suite	D	Craft Stone	Baseline	0/10	438.6	–	27.7
Suite	D	Craft Stone	Static	1/10	492.9	0.745	282.7
Suite	D	Craft Stone	Dynamic	2/10	577.6	0.891	361.6
Suite	D	Place Torch	Baseline	0/10	429.0	–	85.9
Suite	D	Place Torch	Static	0/10	459.5	0.579	93.1
Suite	D	Place Torch	Dynamic	4/10	547.7	0.648	130.3
Suite	D	Iron Chestplate	Baseline	0/10	383.9	–	47.7
Suite	D	Iron Chestplate	Static	0/10	402.3	0.217	31.8
Suite	D	Iron Chestplate	Dynamic	3/10	398.9	0.420	138.2
Trace	D	Place Torch	Static	0/10	600.0	0.420	207.4
Trace	D	Place Torch	Dynamic	2/10	539.0	0.786	293.5
Trace	D	Iron Chestplate	Static	0/10	600.0	0.322	113.1
Trace	D	Iron Chestplate	Dynamic	2/10	563.9	0.838	140.9

B.1 Intervals and exact comparisons

For x successes in n runs, Wilson intervals use $z = 1.959964$ and $\hat{p} = x/n$:

$$\frac{\hat{p} + z^2/(2n) \;\pm\; z\sqrt{\hat{p}(1-\hat{p})/n + z^2/(4n^2)}}{1 + z^2/n}.$$

These are marginal intervals for a proportion, not intervals for the between-regime difference. They are nominal: runs are manually reset independent episodes under the collection procedure, but no randomized allocation or pairing manifest is available.

For stratum i, let n_{Di}, n_{Si} be group sizes and $s_i = x_{Di} + x_{Si}$ the total successes. Under equal odds, conditional on the margins,

$$X_i \sim \mathrm{Hypergeom}(n_{Di} + n_{Si}, s_i, n_{Di}), \qquad T = \sum_i X_i.$$

The script convolves the finite mass functions and sums probabilities no greater than that of the observed T, including ties. With balanced groups the distribution is symmetric. Strata are tasks within each evaluation; no pairing is inferred from IDs.

Comparison	Dynamic	Static	Two-sided p
Task-suite Class D (three task strata)	9/30	1/30	0.013653
Leadership-trace evaluation (two task strata)	4/20	0/20	0.112188
Combined, stratified by task and evaluation	13/50	1/50	0.001005

The combined calculation is a secondary description of the same episodes, not independent corroboration. It retains five task–evaluation strata rather than assuming one exchangeable batch, but does not remove task selection, service-revision, or collection-order confounding. Torch and Chestplate are deliberately selected diagnostic tasks. Comparisons are exploratory and unadjusted for multiplicity; no confirmatory family-wise error guarantee is claimed.

B.2 Task sensitivity

Leaving each task out of the task-suite Class-D comparison gives:

Excluded task	Dynamic	Static	Two-sided p
Craft Stone	7/20	0/20	0.009125
Place Torch	5/20	1/20	0.188366
Iron Chestplate	6/20	1/20	0.095486

The direction favors Dynamic in each task, but a small task inventory does not support a reliable task-general effect. These checks are not additional independent hypothesis confirmations. Calls, durations, and post-outcome leadership groupings are descriptive, not matched-compute or causal efficiency estimates.

C Data and Implementation Scope

The evaluation has two parts: a task-suite comparison (330 episodes; eleven tasks, three regimes, ten runs per cell) and a leadership-trace comparison (40 episodes; two tasks, two structured regimes, ten runs per cell). All 370 files retain the same 22 outcome, identifier, and diagnostic fields. The twenty Dynamic trace files additionally retain 44 election-winner records. No run is removed based on its outcome, leader, or inference count.

Task specifications, manual resets, common model configuration, and live terminal validation describe the experimental procedure. The CSVs support the reported counts and leader sequences; source code specifies protocol behavior. Neither summaries nor leader records preserve full world trajectories. The terms OBSERVED, LIVE-OBSERVED, CURRENT, AUTHOR-SPECIFIED, and MISSING below respectively denote recorded fields, live observations without snapshots, supplied implementation behavior, protocol settings, and unavailable run-level evidence.

D Task Specifications and Experimental Controls

D.1 Task catalog

Class	Task label	Semantic objective	Specification source
A	gather dirt	Collect 10 `dirt` blocks.	Count recovered from the task-creation record; terminal inventory count is the intended predicate.
A	gather oak	Collect 6 `oak_log` blocks.	Count recovered from the task-creation record.
A	gather sand	Collect 12 `sand` blocks.	Count recovered from the task-creation record.
B	wood+dirt	Collect 20 `oak_log` and 20 `dirt`, allowing parallel ownership.	Counts recovered from the task-creation record.
B	oak+leaves	Collect 8 `oak_log` and 16 `oak_leaves`, allowing parallel ownership.	Original task command confirmed by the authors.
B	oak+sand	Collect 8 `oak_log` and 16 `sand`, allowing parallel ownership.	Original task command confirmed by the authors; same quantities and collection policy as oak+leaves.
C	build materials	Collect 10 `oak_log`, 8 `cobblestone`, and 4 `stick`.	Counts recovered from the task-creation record.
C	wooden tools	Craft one `wooden_pickaxe` through the log–plank–stick dependency chain.	Target recovered from the task-creation record.
D	Craft Stone	Obtain one `stone`: mine stone to obtain `cobblestone`, craft/place a furnace, and smelt the cobblestone.	AUTHOR-SPECIFIED task and terminal item predicate.

Class	Task label	Semantic objective	Specification source
D	Place Torch	Place at least one torch inside a 10×10 base containing accessible oak logs and stone but no coal. The feasible chain is wood/tools $\rightarrow$ cobblestone/furnace $\rightarrow$ log+wood fuel/charcoal $\rightarrow$ stick+charcoal/torch $\rightarrow$ placement.	AUTHOR-SPECIFIED task; exact absolute base coordinates are not retained.
D	Iron Chestplate	Mine prerequisite resources, smelt eight raw iron into ingots, and craft one `iron_chestplate`.	AUTHOR-SPECIFIED task and terminal item predicate.

At task creation, the runtime accepts free-form `task_name` and `requirement` strings. The curated 22-field summaries omit the requirement, so the catalog combines retained task names with author-confirmed original commands and Class-D design records. The two matched Class-B commands were:

```
!collaborationTask("collect_oak_leaves",
  "Collect 8 oak_log and 16 oak_leaves. Use !collectBlocks.")
!collaborationTask("collect_oak_sand",
  "Collect 8 oak_log and 16 sand. Use !collectBlocks.")
```

The table above consolidates the eleven requirements that otherwise span retained task names, task-creation records, and author-confirmed launch commands.

D.2 WHY THE TWO TRICK TASKS ARE DIAGNOSTIC

The recipe chains and the 10×10, no-coal base are AUTHOR-SPECIFIED design constraints. Because the archive lacks reset records and boundary coordinates, they are not independently OBSERVED properties of each retained run.

Craft Stone. Minecraft distinguishes the block mined in the world from the dropped item. Mining ordinary stone without the Silk Touch enchantment yields cobblestone, so "mine stone" is not a valid terminal plan for obtaining the `stone` inventory item. Cobblestone must be smelted in a furnace. The task intentionally places a near-identical lexical cue on both sides of a non-trivial transformation, testing whether an LLM agent grounds the game rule or hallucinates that direct mining already achieved the requested item.

Place Torch. The base offers only oak logs and stone and contains no coal. Searching for coal is therefore an invalid recovery strategy even though coal is a conventional torch ingredient. A grounded plan must create charcoal by smelting an oak log, which itself requires cobblestone, a furnace, and wood fuel; the charcoal is then combined with a stick and the torch must be placed within the base. The task tests closed-world resource reasoning as well as cross-phase authority.

D.3 OPERATOR VALIDATION AND MACHINE-READABLE PREDICATES

Every reported run was continuously monitored by an experimenter. A terminal success was retained only after the exact goal was visible in the Minecraft client: the requested inventory/crafted item for acquisition and crafting tasks, or a newly placed torch inside the base for Place Torch. All success labels are therefore LIVE-OBSERVED, not merely accepted from an LLM completion message. The observation was not archived, so an independent auditor cannot replay it from the curated summaries. The following predicates are equivalent instrumentation requirements for an auditable replication; they were not logged as server-side terminal evidence in the reported runs:

Task	Required terminal predicate
Craft Stone	Team inventory contains at least the prompt-specified number of item ID `minecraft:stone`; `minecraft:cobblestone` must not alias to this ID. Record the furnace smelt event as mechanism evidence, but use the item predicate for success.
Place Torch	At least one `minecraft:torch` block changes from absent to present at coordinates inside the pre-registered base bounds. The reset manifest must verify zero accessible coal/coal ore and record the block palette before launch.
Iron Chestplate	Team inventory contains at least one `minecraft:iron_chestplate`.
Gather/craft tasks	Parse target item IDs and integer quantities from a versioned task catalog, then evaluate them with the same server-side inventory/world-state checker in every condition.

The runtime contains a numeric inventory fast path for simple requirements and LLM judgments for non-numeric node/global completion. Those judgments controlled execution, but TCR followed the stricter operator-confirmation protocol above. This distinction is material for Craft Stone: an LLM could confuse `cobblestone` with `stone`, whereas the operator retained success only after the requested result was observed. No terminal state was serialized, so the hard predicates above remain necessary for independently auditable replication.

D.4 Initial state, world control, and timeout

The experiment protocol fixed one Minecraft 1.21.1 superflat world at seed 42. Before each run, the authors manually restored the world and reinitialized the resource blocks required by that task, then continuously monitored execution and the terminal goal. Place Torch used a 10×10 base whose accessible resource palette contained oak logs and stone but no coal or coal ore. The same manual procedure, task layout, start policy, and success-validation rule were used for all three modes. This fixed-world design removes geographic variation and measures coordination/API stochasticity under one controlled scenario; it does not estimate across-world generalization.

Control	Recorded status
World seed / type	Seed 42 and superflat are AUTHOR-SPECIFIED protocol settings; the summaries do not contain the server generator manifest.
Reset	Manual before every run; no pre/post-reset checksum or block snapshot was retained.
Task resources	Manually restored to the task-specific layout; exact coordinates and counts beyond the task catalog and 10×10 Place Torch constraint are not encoded in the summaries.
Spawn / inventory	The repository configuration declares spawn $(0, 64, 0)$ and a nominal inventory, but the Collaboration plugin does not consume those fields; actual per-run spawn and inventory snapshots are MISSING.
Time / weather	Declared locked in framework configuration, but not enforced by the Collaboration plugin; run-level status is MISSING.
Timeout	600 seconds in all 370 summaries. It is a soft wall-clock deadline: 61 records exceed the nominal limit, by at most 1.541 seconds.

Declared spawn and inventory fields do not substitute for the manually prepared task layout. Run-level state is not retained.

E Agent Profiles and Seeded Memories

E.1 Profiles and loader semantics

The three experiment profiles are Lucy, Max, and Emily; all uniformly specify `deepseek-chat`, the configured request identifier for DeepSeek-V4-Flash non-thinking. The memory loader checks,

in order, an exact profile name, its lowercase form, or the numeric agent count ID (0, 1, 2). An agent-specific `memory_summary` replaces the default summary. For each Class-D task, the author-controlled protocol applies the same JSON and agent mapping in Baseline, Static, and Dynamic. The aggregate logs do not contain the selected seed path or hash.

File	Status	JSON key(s)	Size (whitespace words)
`craft_stone.json`	task seed	Lucy / Max / Emily	76 / 53 / 3
`place_torches.json`	task seed	Lucy / Max / Agent3	71 / 75 / 128
`current.json`	byte-identical alias	Lucy / Max / Agent3	71 / 75 / 128
`craft_iron_chestplate.json`	reported task seed / Section 5	Lucy / Max / Emily	21 / 21 / 27

```
craft_stone.json
  5d6bfa8abea776c55fd5180c0d0531f2d05481486507e852c54e0fec5bfcc7d2
place_torches.json = current.json
  5a5b15210682e39b73e5a10b136e26833daf62705fef77fb3f1c879aa9b8432b
craft_iron_chestplate.json
  f6a1fd94bdd4c4b090df79d17a13889cf15d04ec0cfd6be0cf3894c07b29f469
```

Three mapping details matter. First, `craft_stone.json` incorrectly labels its `task_name` as Iron Chestplate; that field is not interpolated into these memory strings, so the content is unaffected. Second, Place Torch stores the final specialist under `Agent3`; no runtime profile has that name and numeric ID 2 does not match the string. Emily therefore receives the default memory, as stated in the main paper. Third, the author-confirmed Iron configuration uses `craft_iron_chestplate.json`, the three-way split depicted in Section 5. The curated summaries do not store a per-run memory hash, so these mappings are protocol metadata and cannot by themselves establish memory–leader alignment.

E.2 Verbatim Craft Stone memory summaries

Lucy

```
I know how to prepare tool and get cobblestones:
1. I know 4 oak planks can be made by a oak log, 4 sticks can be made by
   2 oak planks.
2. I know 4 oak planks can make a crafting table.
3. I know a wooden pickaxe can be made by 3 oak planks and 2 sticks using
   a crafting table.
4. I know we can get cobblestones by mining stones using a wooden
↪  pickaxe.
```

Max

```
I know how to smelt cobblestones into stones:
1. I know a furnace can be made by 8 cobblestones using a crafting table.
2. I know we can use oak planks as a fuel to smelt cobblestone in a
↪  furnace.
3. I know we can smelt cobblestones using a furnace to get stones.
```

Emily

```
I know nothing.
```

E.3 Verbatim Place Torch memory summaries

Lucy

```
I am the wooden tool chain specialist:
1. I know punch trees to get oak logs.
2. I know 1 oak log crafts into 4 oak planks.
3. I know 2 oak planks craft into 4 sticks.
4. I know 4 oak planks craft into 1 crafting table.
5. I know a wooden pickaxe is made from 3 oak planks and 2 sticks.
6. I know wooden pickaxe can mine cobblestones.
```

Max

```
I am the resource gathering specialist:
1. I know cobblestones come from mining stone with wooden pickaxe.
2. I know we need at least 8 cobblestones for a furnace.
3. I know oak logs can be used as fuel in a furnace to create charcoal.
4. I know charcoal is created by smelting oak logs in a furnace.
5. I know we need to gather both cobblestones (for furnace) and oak logs
   (for charcoal fuel).
```

`Agent3` (unmatched)

```
I am the furnace and torch specialist:
1. I know a furnace is made from 8 cobblestones in a U-shape at a
   crafting table.
2. I know a furnace must be placed in the world before it can be used.
3. I know furnace has 3 slots: top for input, bottom for fuel, right for
   output.
4. I know smelting 1 oak log in a furnace produces 1 charcoal - this
   charcoal is our torch fuel.
5. I know torch recipe: 1 charcoal placed above 1 stick in crafting grid
   makes 4 torches.
6. I know torches provide light level 14 and can be placed on top or
   sides of blocks.
7. I know for base lighting, place torches at entrances, corners, and
   every 10 blocks in corridors.
```

E.4 Verbatim Iron Chestplate memory summaries

Lucy

```
I have experience on mining. I know some rules about what tools are
required to mine a specific type of ore.
```

Max

```
I remember that previously another agent who wanted to make an Iron
Chestplate mined 8 raw iron. He finally made it.
```

Emily

```
I have experience on dealing with ore. I know how to craft a furnace
to smelt raw ore into ingots, which are usually used for crafting things.
```

E.5 What is shared and what remains local

The candidacy LLM receives the full private memory, profile, local state, and local observation. The broadcast candidate contains a state summary plus two LLM-generated strings:

- `sharedInfo`: at most 240 characters, prompted for coordinates, nearby blocks, paths, inventory/equipment, position, or supported observations;
- `leadershipAdvantage`: at most 240 characters, prompted for recipe, prerequisite, ordering, decomposition, verification, or stage-transition suitability without spelling out a long procedure.

This is a prompt-level compression policy, not an information-flow guarantee: the model can still disclose procedural facts. An unused `state_request` handler can also return `privateMemory`, although no sender call was found in the supplied experiment snapshot. The proposed mechanism should therefore be described as lossy local-context ownership, not cryptographically private memory.

F PROMPTS AND MESSAGE SCHEMAS

F.1 PROMPT SOURCE-OF-RECORD

The following table indexes the principal protocol-facing prompt families. The authors attest that these fixed templates in the exact experiment-code artifact were used throughout. The summaries retain neither prompt hashes nor the per-call interpolated prompt and runtime state, so effective prompts remain author-attested rather than independently hash-bound.

Stage	Source	Prompt/response contract
shared information	`modules/election/election_manager.js`, `_generateSharedInfoStatement`	Plain text, one short sentence, truncated to 240 characters
leadership advantage	same file, `_generateLeadershipAdvantageStatement`	Plain text, one short sentence, truncated to 240 characters
Dynamic vote	same file, `_prepareVotePrompt`	JSON: agent ID plus phase name, goal, exit condition, and reason
Static phase vote	same function with `forcedWinnerId`	Same JSON, with fixed leader ID repeated exactly
phase synthesis	same file, `_synthesizeElectedPhaseContext`	JSON phase object
DAG generation	`modules/task_decomposer.js`, `decomposeTask` prompt	JSON array requiring `id`, `name`, `description`, and `dependencies`; optional `preferredAgentId` and `assignmentHint`
executor instruction	`modules/base_agent.js`, `_buildTaskPrompt`	Natural-language assignment with optional hint, predecessor, current-state, and recent-history context
node completion judgment	same file, `checkTaskCompletion`	JSON `completed` plus `feedback` containing `result`, `keyLocations`, `blocker`, and `suggestedFix`
global judge	`dynamic_coordinator.js` / `static_coordinator.js`	JSON completion, reason, evidence, and next-phase hint
Baseline executor	`baseline_coordinator.js`, `_buildBaselinePrompt`	Free-form execution
Baseline diagnostic judge	same file, `_checkCompletionWithLlm`	JSON completion decision; not the active finalization path
recovery / takeover	`modules/coordination/agent_coordinator.js`	Natural-language recovery policy and reconstructed DAG decision

This table is not exhaustive: the supplied experiment snapshot also contains prompts for state updating, environment filtering, preferred-agent contention, recovery and takeover decisions, advancement inference, and guided decomposition. Their source code, rather than this summary table, is authoritative.

F.2 VERBATIM DECISION CLAUSES

The Dynamic vote's operational clauses are:

```
- first infer the IMMEDIATE next incomplete phase from current progress,
  inventories, completed phases, and the global requirement
- do not skip prerequisites; choose the closest unmet stage, not an
↪  attractive
  later stage
- choose the candidate whose leadershipAdvantage best fits that immediate
↪  next phase
- phase scope must be small enough for one leader to decompose into a DAG
↪  now
- phase scope must stop at a concrete exit condition
- treat resource coordinates, known locations, inventory, tools, health,
↪  or position
```

```
  as sharedInfo/execution evidence only
- do NOT elect a candidate mainly because they know coordinates or hold
↪  tools
- prefer leadershipAdvantage that explains recipes, prerequisites,
↪  dependency
  ordering, or stage transition knowledge
- do NOT use capability score as the main criterion
```

The Static branch uses the same evidence but begins:

```
The leader is locked to agent ${this.forcedWinnerId} and MUST NOT change
in this round.
Use the candidacies from every agent to refresh shared task information,
then propose the immediate next incomplete phase for the fixed leader
to coordinate.
```

The phase synthesizer is explicitly forbidden to reconsider the leader and must consider every supporting proposal, preserve prerequisites, emit one bounded phase, and provide a concrete observable exit condition. The candidacy prompts explicitly prohibit unsupported facts, generic capability claims, and long recipes. The Baseline executor receives only the global task and organic-collaboration instruction:

```
Task: ${task.name || task.id}
Requirement: ${task.requirement || task.description || ''}

You are one of multiple Minecraft agents in the same world.
Work with other in-game agents if helpful. You may communicate with them
using normal Minecraft chat and minecraft-ai commands.
Collaborate organically with the other agents to complete the global
↪  task.
```

F.3 TYPED MESSAGE PAYLOADS

Every unfragmented message uses the outer wire wrapper $\{\texttt{t}, \texttt{v}, \texttt{ts}, \texttt{to?}, \texttt{f?}, \texttt{fn?}, \texttt{d}\}$. The table below lists the fields inside `d`; names are shown exactly as sent by the supplied experiment snapshot.

Type	Meaning	Inner `d` fields (wire names)
`es`	election start	`v, ts, tid, term, taskContext, experiment`
`c`	candidacy	`ts, s, n, si, la, adv, ss, term`
`vote_cast`	ballot	`from, term, data:{voteId, voteTo, phase}`
`er`	result	`task, winnerId, leaderName, term, authorityLocked, lockedLeaderId, phaseVotes, electedPhaseContext`
`lh`	heartbeat	`ts, tc, f, fn, term`
`ta`	assignment	`taskId, attemptId, task, to, from, leaderName, experiment, timestamp`
`tp`	progress	`t, f, to, d:{taskId, attemptId, status, timestamp}`
`tc`	completion	`t, f, to, d: {taskId, attemptId, agentId, result, timestamp}`
`tf`	failure	`t, f, to, d:{taskId, attemptId, agentId, status, message, blocker, suggestedFix, timestamp}`
`pp`	phase progress	`globalTaskId, globalTask, phaseResult, completion, phaseProgress, from, term, timestamp`; Static also includes `lockedLeaderId`
`tsh`	task-state handoff	`handoff, from, term`; schema-v1 handoff includes `globalTaskId, term, dagVersion`, node/attempt state, assignments, results, phase context/history, timestamp, and per-completed-node `completionEvidence.records`
`td`	terminal decision	`globalTaskId, status, completedCount, failedCount, totalCount, startTime, endTime, duration, term`
`leader_step_down`	recovery trigger	`from, timestamp, reason`; this payload itself has no term

The outer wrapper accepts a generic data object rather than validating an external JSON Schema. Handoff nevertheless has an explicit in-code validator: envelope and payload terms must match; task identity, term, DAG version, timestamp, node IDs, state lists, assignments, and completion records are checked before storage. Restoration accepts only the immediately preceding term, preserves the logical DAG version, and derives completed state from validated records carrying node, attempt, agent, completion time, term, DAG version, and result. This is structural provenance for an LLM judgment, not world-state verification. The configured 128-character "compression threshold" is unused. Messages above the 256-character limit are fragmented as base64 JSON. Ordinary completion/failure handling checks active-DAG membership and rejects a mismatched supplied `attemptId`. A missing attempt ID is accepted by a compatibility path, so stale-message rejection is conditional on well-formed reports.

G PROTOCOL PARAMETERS

Component	Experiment value	Semantics and provenance
Minecraft	1.21.1	Common experiment/runtime setting.
Agents	Lucy, Max, Emily	Three independent embodied loops.
Embodied-agent/protocol model	DeepSeek-V4-Flash, non-thinking	Reported common backend for all 370 runs; official API, configured request name `deepseek-chat`; no retained per-response fingerprint.
Sampling	provider defaults	No explicit temperature, top-p, max-token, or API random-seed parameter.
Global timeout	600 s	Same retained value in all 370 runs; soft cleanup may overshoot.
Concurrent nodes	3	CURRENT maximum active tasks.
Executor responses	1 structured; 5 Baseline	Maximum `handleMessage` responses in the supplied experiment configuration.
Heartbeat	every 5 s	Dynamic leader heartbeat.
Leader timeout	30 s	Follower timeout after initial 45 s grace period.
Active-node timeout	180 s	Inactivity threshold before task recovery.
Recovery-stall threshold	5	Consecutive recoveries without progress; re-election was disabled in the experiment configuration.
Retry / rewrite / local replan / full replan	1000 each	The experiment implementation is effectively global-timeout-bounded, not a small fixed recovery budget.
Election jitter	uniform 0–1 s	Reduces simultaneous initiators.
Candidate collection	10 s	Experiment election window.
Vote tally	3 s after first vote	Experiment tally delay.
Election safety timeout	60 s	Twice the configured 30 s election timeout.
Assignment loop	1 s	Ready-node polling interval.
Message limit	256 characters	Experiment protocol configuration.
Fragmentation	140-character base64 chunks	Triggered above the 256-character full-message limit for any category; reassembly timeout is 30 s. `maxFragments`=20 is configured but not enforced.
Handoff acceptance	schema v1; previous term on restore	Validates task, envelope/payload term, DAG version, node/attempt references, assignments, and one completion record per completed node.

There is no undefined learned "readiness score." The deterministic tally tie-break is

$$b(a) = 5\,\mathrm{health}(a) + 2\min(\mathrm{food}(a), 20) + 2|\mathrm{inventoryEntries}(a)|.$$

A remaining tie is resolved by lexical agent ID. Concurrent election initiators in the same term similarly retain the smaller ID. Election terms increase monotonically; stale starts, candidacies, votes, and results are ignored. Handoff uses the same term boundary: receipt admits only the current or immediately preceding term, and restoration requires exactly the preceding term plus a valid logical DAG version and complete attempt-linked records. These source-level checks do not establish how often handoff occurred or whether the recorded physical outcomes were correct.

H METRIC DEFINITIONS

H.1 OUTCOME METRICS

TCR is the retained binary terminal-success indicator after live operator confirmation. With $y_r = 1$ only when the experimenter observed the exact task goal, $\mathrm{TCR} = \frac{1}{n}\sum_{r=1}^{n} y_r$. TTC is the explicit `ttc_seconds` run-termination duration for success or failure; manuscript means include all ten runs in a cell rather than only successful runs. The event logger computes wall duration as $T_{\mathrm{wall}} = \frac{\texttt{endedAt} - \texttt{startedAt}}{1000}$, then derives TTC, CER, and COI. The curated summaries retain only the derived fields, not `startedAt`/`endedAt`; consequently TTC cannot be independently recomputed from this archive. All runs use a 600-second soft deadline. Sixty-one records exceed it by at most 1.541 seconds; they are left un-clipped and do not change TCR.

H.2 SUBTASK SUCCESS RATE

For node-instance key $k = (\mathrm{taskId}, \mathrm{dagVersion}, \mathrm{subtaskId})$, let K_A and K_E be the keys represented by assignment and `subtask_end` events:

$$\mathrm{SSR} = \frac{|\{k \in K_A \cup K_E : \text{at least one logged success outcome}\}|}{|K_A \cup K_E|}.$$

Retries do not create additional denominator entries. Any successful `subtask_end` outcome makes the instance successful; a represented instance with no success remains in the denominator. This event reflects the implementation's LLM completion judgment, not an independently checked world predicate. Reported table values are arithmetic means of retained per-run SSR, not a pooled-node rate.

H.3 CONCURRENCY EFFICIENCY RATIO

For each attempt, the logger uses start-to-end duration; for a retried node it retains the shortest observed duration. Missing durations use the estimated duration, defaulting to one second. It computes a longest dependency path within each DAG version, then takes the maximum over phase/DAG versions:

$$\mathrm{CER} = \min\left(1, \frac{\max_v \mathrm{CriticalPath}(G_v)}{T_{\mathrm{wall}}}\right).$$

Sequential phase graphs are not linked and retries beyond the shortest attempt are omitted, so CER can understate multi-phase work. It is an internal graph diagnostic.

H.4 ESTIMATED COORDINATION-ACTIVITY RATIO (LOGGER FIELD COI)

The logger uses constantized estimates:

$$\mathrm{COI} = \frac{T_{\mathrm{election}} + 50\mathrm{ms}\, N_{\mathrm{heartbeat}} + 5\mathrm{ms}\, N_{\mathrm{sent}} + 10\mathrm{ms}\, N_{\mathrm{fragmented}} + T_{\mathrm{handoff}}}{T_{\mathrm{wall}}}.$$

Intervals from different agents are summed rather than unioned, so overlapping work can be double counted. We therefore call this an estimated coordination-activity ratio, retain "COI" only as the immutable CSV field name, and exclude it from the main empirical tables. It is neither CPU utilization nor observed network time. Baseline counts ordinary free-chat message cost only. LLM counts are totals: the hook stores a channel label and latency but no reliable backend response identifier, call-purpose label, or token counts.

I LEADERSHIP RECORDS AND COMPLETE SEQUENCES

I.1 EXPORTER SEMANTICS AND COUNTING RULE

The exporter `formatLeaderTrace` collects `election_end` events for the active task from the agents' event files. It groups them by numeric election term, deduplicates observations of the same winner, sorts terms, and emits one `phase_k_leader` row per term. Conflicting winners would be

explicitly labeled `CONFLICT`; no such row appears in these summaries. This absence is not proof of consensus or complete event delivery.

Values retain winner name, agent ID, term, and trigger reason. The integer in `phase_k_leader` is the *record index*, not a semantic phase. A timeout can create a term without completing a phase. Changes require adjacent recorded winners to have different IDs (equation 4); re-election of the same ID is not a change.

Task	Records	Runs changing	Boundary changes	Timeout changes
Place Torch	20	3/10	3	0
Iron Chestplate	24	5/10	5	1

Torch has ten adjacent transitions (eight phase-boundary, two heartbeat-timeout); Chestplate has fourteen (eleven phase-boundary, three heartbeat-timeout). Eight of nineteen boundary transitions and one of five timeout transitions change identity. Fifteen transitions retain the incumbent. There are nine observed changes across eight episodes.

Nineteen sequences begin at term 1 with reason `initial`. I07 begins at term 2 with `heartbeat_timeout`; its preceding winner is unavailable. Thus there are six timeout-triggered *records*, but five timeout-triggered *adjacent transitions* with both winners known. Counts may be lower bounds for full episodes. Static's files have no leadership rows; its invariant leader is specified by the implementation, not reconstructed from absent records.

I.2 All twenty Dynamic traces

Table 9 maps Figure 3 to record identifiers. L/M/E denote Lucy/Max/Emily. Subscripts are election terms; superscripts I/P/H denote initial, phase-boundary, and heartbeat-timeout elections. Success is the global outcome. Files are sorted by ID only for display, not to imply chronology or pairing.

Table 9: Complete recorded leader sequences. The missing first term of I07 is not imputed.

Label	Experiment suffix	Success	Changes	Recorded sequence
T01	1789738064113	1	0	$\mathrm{M}_1^I \to \mathrm{M}_2^P$
T02	1789738673910	0	1	$\mathrm{M}_1^I \to \mathrm{E}_2^P$
T03	1789740133573	0	0	M_1^I
T04	1789741006963	1	1	$\mathrm{M}_1^I \to \mathrm{M}_2^P \to \mathrm{M}_3^H \to \mathrm{M}_4^H \to \mathrm{E}_5^P \to \mathrm{E}_6^P$
T05	1789817931701	0	1	$\mathrm{M}_1^I \to \mathrm{L}_2^P$
T06	1789818589535	0	0	E_1^I
T07	1789819238033	0	0	L_1^I
T08	1789820271571	0	0	M_1^I
T09	1789820926834	0	0	$\mathrm{E}_1^I \to \mathrm{E}_2^P \to \mathrm{E}_3^P$
T10	1789821589124	0	0	M_1^I
I01	1789822465607	0	0	L_1^I
I02	1789823229466	0	0	$\mathrm{M}_1^I \to \mathrm{M}_2^P$
I03	1789823872908	0	0	$\mathrm{E}_1^I \to \mathrm{E}_2^P$
I04	1789824494226	1	1	$\mathrm{M}_1^I \to \mathrm{M}_2^H \to \mathrm{M}_3^H \to \mathrm{E}_4^P \to \mathrm{E}_5^P$
I05	1789825066167	0	1	$\mathrm{M}_1^I \to \mathrm{L}_2^P$
I06	1789825694507	0	1	$\mathrm{M}_1^I \to \mathrm{L}_2^P$
I07	1789868102598	1	2	$\mathrm{E}_2^H \to \mathrm{L}_3^P \to \mathrm{E}_4^H \to \mathrm{E}_5^P$
I08	1789872715619	0	0	$\mathrm{L}_1^I \to \mathrm{L}_2^P \to \mathrm{L}_3^P$
I09	1789875047722	0	0	L_1^I
I10	1789875926425	0	1	$\mathrm{L}_1^I \to \mathrm{M}_2^P$

A full ID is `Place_Torch_dynamic_` followed by the suffix for T rows, or `Craft_Iron_ChestPlate_dynamic_` followed by the suffix for I rows. The machine-readable table retains full IDs and unabridged values.

Among the four successes, T01 retains Max, T04 and I04 change from Max to Emily, and I07 changes from Emily to Lucy and back to Emily over its recorded terms. Failed episodes also include transfers. This reports all outcomes rather than selecting favorable trajectories. The comparison of $3/8$ successes among episodes with a change and $1/12$ among those without one is post-treatment: run length, difficulty, and progress affect both election opportunities and success. No causal effect or mediation estimate is inferred.

I.3 Unmeasured quantities

These rows contain no semantic phase goals, designated experts, event timing, candidacy text, or successful-handoff-restoration evidence. They measure transferable authority, not memory–leader alignment. Alignment would require phase goals and an expert-mapping rule defined independently of the selected leader. Likewise, `handoff_send_count` measures transmissions; normal phase changes use progress messages, a send need not imply a new leader, and receipt need not imply restoration.

J Data Consistency and All Per-Run Results

The self-contained `analyze.py` reads vertical `metric,value` files, validates common fields and leadership grammar, checks binary outcomes and subtask counter consistency, and groups by unique `experimentId`. It generates all tables, intervals, exact tests, figures, and a SHA-256 input manifest. Each of 37 evaluation–task–regime cells has ten runs.

Check	Result
Distinct experiment IDs / task IDs	370 / 369
Task-suite / trace-evaluation records	330 / 40
Common fields per CSV	22
Dynamic trace files / leader records	20 / 44
Nonbinary outcomes	0
Subtask counter / SSR inconsistencies	0
Conflicting winner rows	0
Sequences lacking term 1	1
Runs with a 600-second configured deadline	370
Termination durations beyond the soft deadline	61
Maximum overshoot	1.541 s

Two Static Chestplate trace records have task ID `task_1789637400993`, but distinct experiment IDs ending in `1789637400993` and `1789637400994`; their LLM counts are 23 and 89. They are independently reset episodes under the collection protocol, not a cross-condition pair. Both are retained by experiment ID. Identifier proximity does not establish duplication, launch times, execution order, or randomization.

In the per-run table, evaluation S is the task suite and L the leadership-trace evaluation. Task codes are GD/GO/GS (A), WD/OL/OS (B), BM/WT (C), and CS/PT/IC (D). Modes B/S/D denote Baseline/Static/Dynamic. Run is the experiment-ID suffix; HS is the handoff-send count. Durations include failures and deadline cleanup. Baseline has no DAG-derived SSR.

Table 10: All 370 measured episodes; no outcome-based exclusions.

Eval.	Task	Mode	Run	Success	TTC (s)	SSR	Calls	HS
S	GD	B	1780123075715	1	12.673	–	9	0
S	GD	B	1780123255201	1	13.223	–	8	0
S	GD	B	1780123255202	1	13.219	–	1	0
S	GD	B	1780123324561	1	13.138	–	8	0
S	GD	B	1780123324562	1	13.133	–	5	0
S	GD	B	1780123397272	1	12.900	–	7	0
S	GD	B	1780123466499	1	12.339	–	6	0
S	GD	B	1780123520484	1	11.124	–	4	0
S	GD	B	1780123629587	1	13.702	–	2	0
S	GD	B	1780123629588	1	13.703	–	5	0

Eval.	Task	Mode	Run	Success	TTC (s)	SSR	Calls	HS
S	GD	D	1782010509378	1	187.348	1.0000	88	0
S	GD	D	1782020000001	1	176.282	1.0000	73	0
S	GD	D	1782020000002	0	600.000	0.5000	83	0
S	GD	D	1782020000003	1	130.051	1.0000	110	0
S	GD	D	1782020000004	1	121.309	1.0000	81	0
S	GD	D	1782020000005	1	123.533	1.0000	116	0
S	GD	D	1782020000006	1	145.643	1.0000	105	0
S	GD	D	1782020000007	1	182.010	1.0000	116	0
S	GD	D	1782020000008	1	149.865	1.0000	84	0
S	GD	D	1782020000009	1	177.333	0.8000	97	0
S	GD	S	1780239267858	1	111.918	1.0000	7	0
S	GD	S	1780239430797	1	56.742	1.0000	14	0
S	GD	S	1780239790841	1	75.723	1.0000	13	0
S	GD	S	1780239959885	1	55.059	1.0000	10	0
S	GD	S	1780240152938	1	86.330	1.0000	18	0
S	GD	S	1780240400976	1	62.571	1.0000	5	0
S	GD	S	1780240494847	1	81.364	1.0000	13	0
S	GD	S	1780240610220	1	66.744	1.0000	10	0
S	GD	S	1780240706553	1	56.728	1.0000	7	0
S	GD	S	1780240795267	1	58.839	1.0000	13	0
S	GO	B	1780123983821	1	32.124	–	11	0
S	GO	B	1780123983822	1	32.131	–	6	0
S	GO	B	1780124055626	1	31.206	–	13	0
S	GO	B	1780124055627	1	31.199	–	4	0
S	GO	B	1780124576242	1	29.488	–	20	0
S	GO	B	1780124657379	1	37.510	–	9	0
S	GO	B	1780124657380	1	46.047	–	3	0
S	GO	B	1780124884647	1	33.724	–	13	0
S	GO	B	1780124884648	1	34.653	–	9	0
S	GO	B	1780124975135	0	54.963	–	32	0
S	GO	D	1781579034823	1	231.648	1.0000	63	0
S	GO	D	1781581800998	0	600.010	0.9231	308	0
S	GO	D	1781583175672	0	600.014	0.8889	121	0
S	GO	D	1781666495085	1	375.952	0.0000	63	0
S	GO	D	1781856101868	1	74.283	1.0000	20	0
S	GO	D	1781856432923	1	22.815	1.0000	6	0
S	GO	D	1781857398526	1	259.015	1.0000	46	0
S	GO	D	1781869254689	1	60.961	1.0000	34	0
S	GO	D	1782020000010	1	199.004	0.8000	94	0
S	GO	D	1782020000011	1	253.585	0.8000	100	0
S	GO	S	1780240946502	1	83.113	1.0000	17	0
S	GO	S	1780241066554	1	57.370	1.0000	10	0
S	GO	S	1780241153291	1	78.153	1.0000	9	0
S	GO	S	1780241265421	1	70.732	1.0000	7	0
S	GO	S	1780241875773	1	172.546	1.0000	31	0
S	GO	S	1780242092130	1	59.328	1.0000	13	0
S	GO	S	1780242189316	1	109.705	1.0000	19	0
S	GO	S	1780242339150	1	80.909	1.0000	15	0
S	GO	S	1780242449478	1	107.762	1.0000	16	0
S	GO	S	1780242589446	1	78.665	1.0000	13	0
S	GS	B	1780141560089	1	20.774	–	7	0
S	GS	B	1780141639250	1	24.059	–	5	0
S	GS	B	1780141639341	1	23.968	–	5	0
S	GS	B	1780141639508	1	23.799	–	4	0
S	GS	B	1780141723429	1	24.687	–	3	0
S	GS	B	1780141723430	1	24.688	–	7	0
S	GS	B	1780141851916	1	25.673	–	11	0
S	GS	B	1780141910666	1	30.811	–	19	0
S	GS	B	1780142050134	1	23.507	–	5	0
S	GS	B	1780142106509	1	32.911	–	6	0
S	GS	D	1781674236326	1	108.797	1.0000	35	0
S	GS	D	1781929786056	1	442.025	1.0000	313	0
S	GS	D	1782020000012	0	600.000	0.6000	169	0
S	GS	D	1782020000013	1	267.328	0.8333	135	0
S	GS	D	1782020000014	1	183.335	0.8333	117	0
S	GS	D	1782020000015	1	258.492	0.8333	117	0
S	GS	D	1782020000016	1	166.013	0.8571	145	0
S	GS	D	1782020000017	1	165.510	0.8000	98	0
S	GS	D	1782020000018	1	221.939	1.0000	136	0
S	GS	D	1782020000019	1	176.915	0.7500	139	0
S	GS	S	1780242742198	1	135.867	1.0000	12	0
S	GS	S	1780242929187	1	114.629	1.0000	23	0
S	GS	S	1780243118320	0	86.314	0.0000	5	0
S	GS	S	1780243118321	1	86.310	1.0000	14	0
S	GS	S	1780243241213	1	217.280	1.0000	42	0
S	GS	S	1780243534595	1	197.414	1.0000	26	0
S	GS	S	1780243782748	1	225.438	1.0000	38	0
S	GS	S	1780244636664	0	158.887	0.0000	9	0
S	GS	S	1780244636767	1	158.774	1.0000	19	0

Eval.	Task	Mode	Run	Success	TTC (s)	SSR	Calls	HS
S	GS	S	1780244982865	1	113.121	1.0000	23	0
S	OL	B	1781574625533	0	599.999	–	45	0
S	OL	B	1781574625536	0	599.997	–	41	0
S	OL	B	1781574625542	0	599.994	–	16	0
S	OL	B	1781575426421	1	32.193	–	3	0
S	OL	B	1781575426422	1	32.191	–	4	0
S	OL	B	1781575541395	1	40.547	–	3	0
S	OL	B	1781575887182	0	600.003	–	12	0
S	OL	B	1781575887183	0	600.002	–	2	0
S	OL	B	1781577324744	0	599.772	–	10	0
S	OL	B	1781577324745	0	599.768	–	6	0
S	OL	D	1781662259341	1	450.019	1.0000	38	0
S	OL	D	1781662922567	1	350.000	1.0000	45	0
S	OL	D	1781664499297	1	324.016	1.0000	21	0
S	OL	D	1782110237392	1	423.008	1.0000	28	0
S	OL	D	1782111207942	1	298.090	1.0000	31	0
S	OL	D	1782112519024	1	249.991	1.0000	14	0
S	OL	D	1782113939202	1	400.009	1.0000	44	0
S	OL	D	1782114595307	1	600.007	1.0000	33	0
S	OL	D	1782115263313	0	389.878	0.6667	20	0
S	OL	D	1782116658357	1	399.999	1.0000	49	0
S	OL	S	1781617343951	1	112.445	1.0000	17	0
S	OL	S	1781617538301	1	119.137	1.0000	16	0
S	OL	S	1781617712616	1	337.117	1.0000	38	0
S	OL	S	1781618279893	1	112.776	1.0000	21	0
S	OL	S	1781622440145	1	181.388	0.0000	9	0
S	OL	S	1781622440146	1	181.379	1.0000	17	0
S	OL	S	1781622685281	1	168.050	0.0000	14	0
S	OL	S	1781622685282	1	168.042	1.0000	7	0
S	OL	S	1781622914439	1	104.045	1.0000	19	0
S	OL	S	1781624455170	0	184.432	0.0000	6	0
S	OS	B	1781579390557	0	600.103	–	55	0
S	OS	B	1781579390824	0	600.280	–	6	0
S	OS	B	1781581222735	0	599.989	–	20	0
S	OS	B	1781581222739	0	599.989	–	21	0
S	OS	B	1781581222740	0	599.988	–	21	0
S	OS	B	1781607471233	0	600.044	–	19	0
S	OS	B	1781607471247	0	600.044	–	8	0
S	OS	B	1781608327597	1	32.314	–	15	0
S	OS	B	1781609548266	0	600.034	–	4	0
S	OS	B	1781609548267	0	600.035	–	37	0
S	OS	D	1782117379951	1	399.998	1.0000	37	0
S	OS	D	1782118059215	1	269.971	1.0000	36	0
S	OS	D	1782118692899	0	599.996	0.4000	85	0
S	OS	D	1782119385398	1	200.067	1.0000	43	0
S	OS	D	1782128555334	1	259.982	1.0000	40	0
S	OS	D	1782129188397	0	599.983	0.5000	29	0
S	OS	D	1782129864347	1	259.948	1.0000	47	0
S	OS	D	1782131065217	1	340.038	1.0000	49	0
S	OS	D	1782133523360	0	599.987	0.5000	47	0
S	OS	D	1782134177100	1	240.175	1.0000	29	0
S	OS	S	1781611572468	1	276.019	1.0000	13	0
S	OS	S	1781611572469	0	276.025	0.0000	16	0
S	OS	S	1781613201777	1	285.903	1.0000	36	0
S	OS	S	1781613201779	0	285.909	0.0000	13	0
S	OS	S	1781613521399	1	86.198	1.0000	17	0
S	OS	S	1781613880572	1	134.375	1.0000	51	0
S	OS	S	1781614329475	1	136.556	1.0000	13	0
S	OS	S	1781614906091	1	438.414	1.0000	49	0
S	OS	S	1781616293287	1	155.887	1.0000	11	0
S	OS	S	1781616640980	1	187.580	1.0000	20	0
S	WD	B	1780213560506	1	32.295	–	11	0
S	WD	B	1780214302237	1	31.413	–	7	0
S	WD	B	1780214420738	0	499.990	–	14	0
S	WD	B	1780214965629	0	499.988	–	22	0
S	WD	B	1780215561010	0	499.990	–	24	0
S	WD	B	1780215561011	0	499.989	–	11	0
S	WD	B	1780217883264	1	28.011	–	10	0
S	WD	B	1780217883265	1	28.035	–	5	0
S	WD	B	1780220059604	0	500.003	–	24	0
S	WD	B	1780220059605	0	500.007	–	13	0
S	WD	D	1781674447235	0	600.015	0.8889	161	0
S	WD	D	1781926429232	1	234.724	1.0000	37	0
S	WD	D	1782020000020	0	552.513	0.6667	120	0
S	WD	D	1782020000021	0	592.074	0.7143	126	0
S	WD	D	1782020000022	1	342.048	0.8000	87	1
S	WD	D	1782020000023	1	365.392	0.8571	129	1
S	WD	D	1782020000024	1	343.224	0.8889	121	1
S	WD	D	1782020000025	1	366.169	0.8750	121	1

Eval.	Task	Mode	Run	Success	TTC (s)	SSR	Calls	HS
S	WD	D	1782020000026	1	383.270	1.0000	89	1
S	WD	D	1782020000027	1	388.123	0.8750	130	1
S	WD	S	1780280510739	1	319.414	1.0000	13	0
S	WD	S	1780281202100	0	284.939	0.0000	11	0
S	WD	S	1780281202101	1	284.931	1.0000	36	0
S	WD	S	1780282176664	1	172.117	1.0000	12	0
S	WD	S	1780282176665	1	172.124	1.0000	7	0
S	WD	S	1780282949208	1	229.315	1.0000	35	0
S	WD	S	1780283432365	1	163.389	1.0000	27	0
S	WD	S	1780283635015	1	146.648	1.0000	23	0
S	WD	S	1780283829210	1	172.344	1.0000	5	0
S	WD	S	1780283829213	1	172.336	1.0000	16	0
S	WT	B	1780228387674	0	600.005	–	10	0
S	WT	B	1780229075966	0	255.008	–	7	0
S	WT	B	1780229075967	0	200.008	–	18	0
S	WT	B	1780229613472	0	499.962	–	23	0
S	WT	B	1780229613720	0	459.962	–	16	0
S	WT	B	1780230151549	0	320.003	–	11	0
S	WT	B	1780230151551	0	500.003	–	6	0
S	WT	B	1780232458549	0	520.003	–	14	0
S	WT	B	1780234432423	0	500.009	–	5	0
S	WT	B	1780234432426	0	542.008	–	17	0
S	WT	D	1781676089697	1	510.569	1.0000	238	0
S	WT	D	1782020000036	1	536.595	1.0000	263	1
S	WT	D	1782020000037	1	483.178	1.0000	215	0
S	WT	D	1782020000038	1	546.354	1.0000	245	0
S	WT	D	1782020000039	1	530.245	1.0000	226	0
S	WT	D	1782020000040	0	586.825	0.5556	209	1
S	WT	D	1782020000041	0	596.021	0.5556	210	0
S	WT	D	1782020000042	0	588.180	0.7778	187	1
S	WT	D	1782020000043	0	591.815	0.8333	265	1
S	WT	D	1782020000044	1	593.169	1.0000	216	0
S	WT	S	1780379792721	1	450.159	1.0000	73	0
S	WT	S	1780380302819	0	600.013	0.8333	114	0
S	WT	S	1780381569808	1	394.043	1.0000	61	0
S	WT	S	1780382107539	1	504.042	0.0000	7	0
S	WT	S	1780382107543	1	504.029	1.0000	54	0
S	WT	S	1780382108027	1	503.554	0.0000	11	0
S	WT	S	1780383465375	1	496.805	1.0000	97	0
S	WT	S	1780387180966	0	599.991	0.8000	74	0
S	WT	S	1780388156784	0	599.128	0.8000	81	0
S	WT	S	1780388798120	1	590.586	1.0000	77	0
S	BM	B	1780221520719	0	500.008	–	12	0
S	BM	B	1780222113646	0	500.004	–	11	0
S	BM	B	1780222113652	0	500.004	–	16	0
S	BM	B	1780223955683	0	499.911	–	31	0
S	BM	B	1780223955684	0	499.911	–	17	0
S	BM	B	1780225277851	0	500.003	–	9	0
S	BM	B	1780225277852	0	500.003	–	15	0
S	BM	B	1780225277868	0	500.002	–	10	0
S	BM	B	1780227148610	1	77.904	–	26	0
S	BM	B	1780227148615	1	77.911	–	5	0
S	BM	D	1781926155502	1	120.951	1.0000	30	0
S	BM	D	1782009568252	0	600.015	0.6667	180	0
S	BM	D	1782020000028	1	261.699	0.8182	121	0
S	BM	D	1782020000029	1	258.606	0.9167	104	1
S	BM	D	1782020000030	1	346.180	1.0000	112	1
S	BM	D	1782020000031	1	248.881	0.8889	117	0
S	BM	D	1782020000032	1	278.550	0.8182	103	0
S	BM	D	1782020000033	1	326.843	0.9091	99	1
S	BM	D	1782020000034	1	328.480	0.8571	146	1
S	BM	D	1782020000035	1	264.612	1.0000	126	1
S	BM	S	1780305428034	1	343.669	1.0000	43	0
S	BM	S	1780306080913	1	196.234	1.0000	22	0
S	BM	S	1780306314582	1	148.353	1.0000	20	0
S	BM	S	1780306493540	1	231.204	1.0000	30	0
S	BM	S	1780306895291	1	119.690	1.0000	21	0
S	BM	S	1780307073641	1	114.926	1.0000	21	0
S	BM	S	1780307267119	1	395.772	1.0000	65	0
S	BM	S	1780325380696	1	204.358	1.0000	48	0
S	BM	S	1780325638380	1	282.645	1.0000	41	0
S	BM	S	1780326748467	1	403.955	1.0000	89	0
S	IC	B	1784189042420	0	259.111	–	78	0
S	IC	B	1784191000001	0	430.006	–	42	0
S	IC	B	1784191000002	0	600.012	–	35	0
S	IC	B	1784191000003	0	600.004	–	53	0
S	IC	B	1784191000004	0	450.021	–	47	0
S	IC	B	1784191000005	0	340.009	–	39	0
S	IC	B	1784191000006	0	250.015	–	58	0

Eval.	Task	Mode	Run	Success	TTC (s)	SSR	Calls	HS
S	IC	B	1784191000007	0	290.007	–	44	0
S	IC	B	1784191000008	0	460.018	–	32	0
S	IC	B	1784191000009	0	160.011	–	49	0
S	IC	D	1778392629678	0	90.123	0.0000	37	2
S	IC	D	1778559490957	1	130.449	1.0000	48	2
S	IC	D	1778560717207	0	120.993	0.0000	23	0
S	IC	D	1778562611811	1	534.005	1.0000	67	4
S	IC	D	1778580660124	0	142.578	0.2500	40	1
S	IC	D	1778585932597	1	587.010	1.0000	187	7
S	IC	D	1784193000001	0	596.842	0.2500	241	3
S	IC	D	1784193000002	0	600.000	0.2857	268	4
S	IC	D	1784193000003	0	587.319	0.1818	219	2
S	IC	D	1784193000004	0	600.000	0.2308	252	3
S	IC	S	1784189875333	0	240.018	0.0000	27	0
S	IC	S	1784192000001	0	300.008	0.6667	31	0
S	IC	S	1784192000002	0	423.016	0.0000	29	0
S	IC	S	1784192000003	0	460.005	0.0000	36	0
S	IC	S	1784192000004	0	600.019	0.0000	25	0
S	IC	S	1784192000005	0	600.010	0.0000	41	0
S	IC	S	1784192000006	0	600.014	0.5000	30	0
S	IC	S	1784192000007	0	450.006	0.0000	34	0
S	IC	S	1784192000008	0	210.022	0.6667	27	0
S	IC	S	1784192000009	0	140.013	0.3333	38	0
S	CS	B	1782011876878	0	423.094	–	23	0
S	CS	B	1782020000045	0	331.000	–	31	0
S	CS	B	1782020000046	0	600.000	–	21	0
S	CS	B	1782020000047	0	400.003	–	20	0
S	CS	B	1782020000048	0	132.000	–	32	0
S	CS	B	1782020000049	0	600.000	–	28	0
S	CS	B	1782020000050	0	600.000	–	27	0
S	CS	B	1782020000051	0	600.022	–	30	0
S	CS	B	1782020000052	0	457.000	–	31	0
S	CS	B	1782020000053	0	243.000	–	34	0
S	CS	D	1778642440524	0	600.013	0.9231	352	0
S	CS	D	1778644151471	0	565.404	0.8235	325	0
S	CS	D	1782020000062	0	596.706	0.8421	345	0
S	CS	D	1782020000063	0	579.218	0.8947	374	1
S	CS	D	1782020000064	0	567.392	0.8500	325	1
S	CS	D	1782020000065	0	587.253	0.8571	373	2
S	CS	D	1782020000066	0	593.330	0.8421	342	0
S	CS	D	1782020000067	0	568.446	0.8824	412	0
S	CS	D	1782020000068	1	586.309	1.0000	350	1
S	CS	D	1782020000069	1	532.331	1.0000	418	1
S	CS	S	1778643309752	0	470.834	0.6667	323	0
S	CS	S	1778645591593	0	350.616	0.7857	200	0
S	CS	S	1782020000054	1	552.041	1.0000	292	0
S	CS	S	1782020000055	0	572.713	0.7500	322	0
S	CS	S	1782020000056	0	411.933	0.7333	319	0
S	CS	S	1782020000057	0	532.774	0.6000	322	0
S	CS	S	1782020000058	0	519.182	0.7143	220	0
S	CS	S	1782020000059	0	404.848	0.6667	292	0
S	CS	S	1782020000060	0	596.429	0.7500	260	0
S	CS	S	1782020000061	0	517.816	0.7857	277	0
S	PT	B	1784741955214	0	600.020	–	42	0
S	PT	B	1784783902241	0	258.015	–	76	0
S	PT	B	1784790646152	0	123.009	–	93	0
S	PT	B	1784793453948	0	599.966	–	61	0
S	PT	B	1784794162798	0	259.001	–	165	0
S	PT	B	1784794918612	0	410.004	–	100	0
S	PT	B	1784795678200	0	600.005	–	154	0
S	PT	B	1784797603469	0	240.007	–	94	0
S	PT	B	1784798914697	0	599.990	–	50	0
S	PT	B	1784798914701	0	599.986	–	24	0
S	PT	D	1784626583426	0	600.014	0.0000	114	0
S	PT	D	1784627277920	0	600.020	0.0000	77	0
S	PT	D	1784800000001	1	438.672	1.0000	126	2
S	PT	D	1784800000002	1	512.308	1.0000	143	3
S	PT	D	1784800000003	1	367.945	1.0000	101	1
S	PT	D	1784800000004	1	558.126	1.0000	167	4
S	PT	D	1784800000005	0	600.011	0.6000	132	1
S	PT	D	1784800000006	0	599.996	0.6667	158	2
S	PT	D	1784800000007	0	600.018	0.5000	109	1
S	PT	D	1784800000008	0	599.989	0.7143	176	3
S	PT	S	1784390761504	0	600.042	0.9167	181	0
S	PT	S	1784391951547	0	120.009	0.7500	93	0
S	PT	S	1784702217880	0	230.002	0.7500	141	0
S	PT	S	1784702217881	0	600.003	0.0000	9	0
S	PT	S	1784703205775	0	340.012	0.7500	126	0
S	PT	S	1784703205776	0	600.013	0.0000	53	0

Eval.	Task	Mode	Run	Success	TTC (s)	SSR	Calls	HS
S	PT	S	1784703878724	0	453.005	0.8571	143	0
S	PT	S	1784705631313	0	599.980	0.9091	116	0
S	PT	S	1784705631314	0	599.981	0.0000	20	0
S	PT	S	1784707604367	0	452.005	0.8571	49	0
L	IC	D	1789822465607	0	600.011	0.7778	271	0
L	IC	D	1789823229466	0	600.007	0.9375	338	0
L	IC	D	1789823872908	0	600.008	0.5000	56	0
L	IC	D	1789824494226	1	524.058	1.0000	104	0
L	IC	D	1789825066167	0	600.008	0.8571	67	0
L	IC	D	1789825694507	0	600.017	0.8000	86	0
L	IC	D	1789868102598	1	315.254	1.0000	66	0
L	IC	D	1789872715619	0	600.014	0.9286	163	0
L	IC	D	1789875047722	0	600.017	0.7500	25	0
L	IC	D	1789875926425	0	600.008	0.8333	233	0
L	IC	S	1789637400993	0	600.042	0.0000	23	0
L	IC	S	1789637400994	0	600.043	0.7500	89	0
L	IC	S	1789638747504	0	600.013	0.9000	185	0
L	IC	S	1789641533243	0	599.988	0.6667	221	0
L	IC	S	1789641533244	0	599.988	0.0000	114	0
L	IC	S	1789643437572	0	600.023	0.9000	257	0
L	IC	S	1789644263449	0	600.018	0.0000	9	0
L	IC	S	1789645518486	0	600.001	0.0000	67	0
L	IC	S	1789645518493	0	599.994	0.0000	81	0
L	IC	S	1789646783182	0	599.997	0.0000	85	0
L	PT	D	1789738064113	1	132.122	1.0000	311	0
L	PT	D	1789738673910	0	600.256	0.7500	171	0
L	PT	D	1789740133573	0	600.013	0.8889	367	0
L	PT	D	1789741006963	1	456.011	0.8333	258	0
L	PT	D	1789817931701	0	600.004	0.8333	150	0
L	PT	D	1789818589535	0	601.541	0.0000	292	0
L	PT	D	1789819238033	0	600.016	0.9167	315	0
L	PT	D	1789820271571	0	600.032	0.8750	387	0
L	PT	D	1789820926834	0	600.013	0.8333	403	0
L	PT	D	1789821589124	0	600.013	0.9333	281	0
L	PT	S	1789541403074	0	600.018	0.0000	228	0
L	PT	S	1789542172531	0	600.027	0.0000	219	0
L	PT	S	1789542897232	0	600.018	0.0000	123	0
L	PT	S	1789543961497	0	600.007	0.8889	367	0
L	PT	S	1789545156632	0	600.006	0.6667	87	0
L	PT	S	1789545156641	0	600.000	0.0000	91	0
L	PT	S	1789545887294	0	599.910	0.8571	245	0
L	PT	S	1789546799565	0	600.005	0.9286	366	0
L	PT	S	1789547718186	0	600.051	0.0000	142	0
L	PT	S	1789549569108	0	600.066	0.8571	206	0

K Artifact and Reproduction

K.1 Reproducing the analysis

The anonymous source bundle includes `data/task_suite/A` through `D` and `data/leadership`: the exact 370 files used here. Recomputing statistics requires no private repository, Minecraft installation, or model API:

```
python -m pip install -r requirements-analysis.txt
python analyze.py
python build.py
```

Outputs include `analysis/per_run_results.csv`, `task_summary.csv`, `class_summary.csv`, `leadership_sequences.csv`, `leadership_summary.csv`, `audit.json`, TeX tables, and vector figures. Building requires pdfLaTeX, BibTeX, `pdfinfo`, and `pdftotext`; checks cover the nine-page main-text limit, resolved references, and required statements.

K.2 Coordination implementation

The supplementary artifact includes the Collaboration plugin, its configuration and protocol files, and three task seeds. The plugin extends the public `minecraft-ai` framework (Minecraft AI Contributors, 2026); it is independent of that framework's authors. Included integration source and

dependency manifests document the executor and memory-loading interface. Credentials must be configured locally.

To run: install framework dependencies; install the plugin at `src/plugins/Collaboration`; select Lucy, Max, and Emily profiles and the task seed; configure a DeepSeek endpoint and credential; launch Minecraft Java 1.21.1 and a seed-42 superflat world; open to LAN and configure its port; restore task resources; and start agents with `node main.js`. Set `experiment_mode` to `BASELINE`, `STATIC`, or `DYNAMIC` before launch. The mode is not an argument to the task command. For example:

```
@Lucy !collaborationTask("collect_oak_leaves",
  "Collect 8 oak_log and 16 oak_leaves. Use !collectBlocks.")
```

The plugin exports metrics automatically. A new execution should retain raw events and terminal evidence. Model aliases can change or be retired: a request string does not pin a backend revision. Recorded results are not relabeled using currently available models.

K.3 Limits of exact replay

Both evaluations use the same coordination method and reported model/API configuration; the trace evaluation adds winner records to the export. Per-run code hashes, response model fingerprints, hardware/software manifests, reset hashes, exact resource coordinates, and terminal snapshots are unavailable. Provider-default sampling and a hosted API limit bit-identical replay. The artifact reproduces the analysis and exposes the supplied implementation; it cannot recreate missing episode-level evidence.