Unified Resource-Grounded Coordination Protocol for Orchestrator-Free Heterogeneous Multi-Agent Systems

Vishal Pramanik
Department of Computer and Information Science and Engineering
University of Florida
Gainesville, FL, USA
vishalpramanik@ufl.edu

Maisha Maliha
School of Computer Science
University of Oklahoma
Norman, OK, USA
maisha.maliha-1@ou.edu

Olivera Kotevska
Computer Science and Mathematics Division
Oak Ridge National Laboratory
Oak Ridge, TN, USA
kotevskao@ornl.gov

Arvind Ramanathan
Data Science and Learning Division
Argonne National Laboratory
Lemont, IL, USA
ramanathana@anl.gov

Nathaniel D. Bastian
Institute for Artificial Intelligence
Syracuse University
Syracuse, NY, USA
ndbastia@syr.edu

Susmit Jha
Computer Science Laboratory
SRI International
Menlo Park, CA, USA
susmit.jha@berkeley.edu

Sumit Kumar Jha
Department of Computer and Information Science and Engineering
University of Florida
Gainesville, FL, USA
sumit.jha@ufl.edu

Abstract

Multi-agent language-model systems typically rely on a human designer or external orchestrator to fix the agent population, roles, and workflow—a single point of failure that places coordination decisions outside the team executing them. Existing approaches address fragments of this problem: communication standards lack coordination logic, automated workflow-search systems concentrate authority in a single planner, and decentralized peers lack explicit governance, handoff validation, and bounded repair. To close this gap, we introduce the **Unified Resource-Grounded Coordination Protocol (URCP)**, an **orchestrator-free** protocol in which a fixed pool of black-box models and executable resources collectively decides task framing, role formation, workflow topology, assignment, checkpoint validation, and final completion through supermajority voting under a single decision rule Φ_q. A five-level repair hierarchy reopens only the affected part of the workflow on failure; under standard timing and communication assumptions, every run terminates with an **explicit verdict**. Empirically, with twelve heterogeneous models against eight baselines on MultiAgentBench, τ-bench, MemoryAgentBench, GAIA, and

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

REALM-Bench, URCP exceeds the strongest baseline by $+9.7$ / $+7.9$ / $+10.6$ points on the three primary benchmarks, terminates deadlock-free in 100% of runs, and remains stable up to 20% agent failure. Trace analyses show that the gains come from voted role concentration on frontier models (73.9% of critical-path roles), frequent critic/verifier insertion (63.5% of accepted schemas with at least three roles), and bounded repair rather than from model strength alone. The code is publicly available.[1]

1 Introduction

Multi-agent language-model systems are now used to tackle software engineering, retrieval-augmented analysis, and tool-rich automation, and two findings argue for giving such systems more control over how they coordinate themselves. The first is that no single model is best across all task families. Routing and ranking studies consistently show that selecting among complementary models outperforms any fixed choice [12, 34, 26]. The second is that language-model agents can already learn to use tools, revise their own outputs, and coordinate with limited human intervention [31, 33]. Taken together, these results suggest that a heterogeneous pool of models should not merely execute a human-designed workflow. With the right governance mechanism, it should be able to negotiate its own roles, assignments, and execution structure.

Existing approaches each fall short on a different axis. Hand-designed role frameworks such as CAMEL [18], ChatDev [30], and MetaGPT [11] fix the team and the pipeline in advance, which limits them to whatever use cases the designer anticipated. Agent-communication standards such as MCP [2], ACP [14], and A2A [10] standardize how agents exchange messages but provide no mechanism for collectively deciding who does what; they are transport layers rather than coordination layers [7]. Automated workflow-search systems such as AFlow [42] and MetaGen-style approaches [35] replace hand-written roles with optimization but concentrate authority in a single architect whose mistakes propagate unchecked. Decentralized peer systems such as AgentNet [39] and ReSo [44] remove the orchestrator entirely yet lose every explicit handshake guarantee, so a single corrupted handoff can silently poison a long-horizon run. Across all of these lines, no existing system lets the participants collectively negotiate agent count, role definitions, workflow topology, and task assignments under formal guarantees that every decision is ratified before execution and that the protocol always terminates with an explicit verdict.

We close this gap with the Unified Resource-Grounded Coordination Protocol (URCP), which replaces external orchestration with a single supermajority decision rule Φ_q applied uniformly at every coordination step. Given only a task description and a shared resource pool, the participating models collectively agree on the task frame, the role schema, the assignment of roles to voters and resources, the checkpointed execution along a validated DAG, and the final output. Each of these stages is proposed, voted on, and logged in an auditable record, so the workflow is never chosen by a hidden planner and every handoff is collectively validated before any downstream work begins. When failures arise during execution, URCP reopens only the smallest affected scope through a five-level repair hierarchy that preserves validated work across epoch boundaries, and runs that genuinely cannot recover terminate with an explicit NO-CONSENSUS verdict rather than hanging silently or fabricating an output the voters do not endorse. We evaluate URCP on MultiAgentBench, τ-bench, and MemoryAgentBench using a heterogeneous pool of twelve contemporary models against eight baselines covering the protocol-only, centrally-orchestrated, and decentralized families, and we further study its scaling with task complexity, its robustness under model dropout and communication stress, the contribution of each repair level, the role of team composition, and trace-level emergent coordination patterns drawn from the full audit log.

2 Motivation and Background

A growing body of evidence motivates URCP from two directions. First, no single language model dominates all task families: different models excel at coding and repository repair [16, 15], tool-augmented automation [32], multi-step agentic decision-making [19], policy-grounded interaction [40], multimodal reasoning [41, 20, 21], and web or desktop navigation [46, 17, 37, 23].

[1] https://github.com/VishalPramanik/Unified-Resource-Grounded-Coordination-Protocol-

Table 1: Motivating design-property gap. URCP is the only compared system that combines peer governance, supermajority voting, negotiated roles, multi-level repair, formal guarantees, model-agnostic execution, and no central orchestrator. ✓ supported; ∼ partial; ✗ absent.

Property	**A2A** [10]	**MCP** [2]	**ACP** [14]	**MetaG.** [35]	**AFLOW** [42]	**DynR.** [43]	**AgNet** [39]	**ReSo** [44]	**URCP** (ours)
PG: peer governance	✗	✗	✗	✗	✗	✗	✓	✗	✓
SV: Φ_q voting	✗	✗	✗	✗	✗	✗	✗	✗	✓
RN: negotiated roles	✗	✗	✗	✓	✓	∼	✗	✓	✓
MR: multi-level repair	✗	✗	✗	∼	✗	✗	✗	✗	✓
FG: formal guarantees	✗	✗	✗	✗	✗	✗	✗	✗	✓
MA: model-agnostic	✓	✓	✓	✓	✗	✓	✓	✗	✓
NCO: no orchestrator	✗	✗	✗	✗	✗	✗	✓	✗	✓
Σ	1	1	1	2	1	1	3	1	**7**

Model-routing studies further show that selecting among heterogeneous models can yield large cost–quality gains [26, 6]. Second, language models can already coordinate and improve with limited human intervention: agents can learn tool use [31], self-reflect [33], and collaborate in multi-agent software-development settings [18, 11, 30]. However, these systems still typically receive their role structure from a human template or a central controller.

The missing piece is not communication alone, but governed coordination. Protocol-only systems such as A2A, MCP, and ACP [10, 2, 14] move messages but do not decide structure. Centrally orchestrated systems such as MetaGen [35] and AFLOW [42] automate workflow design, but place authority in a single planner. Decentralized systems such as AgentNet [39] and ReSo [44] remove the controller, but do not provide explicit supermajority governance, checkpoint handoff validation, and bounded repair. As Table 1 shows, existing systems cover fragments of the desired design space; URCP is designed to combine them in one protocol. URCP closes this gap by making coordination itself a voted protocol object. From a task and a shared resource pool, the participating models negotiate the task frame, subtasks, role schema, execution DAG, role assignments, checkpoint outputs, and final answer under the same supermajority rule Φ_q. The runtime is not a semantic planner: it only broadcasts messages, enforces timeouts, filters mechanical eligibility, records votes, locks accepted outputs, and maintains the audit log. Thus no data flows downstream until the team has a verified coordination state, and no single orchestrator decides the structure of the run.

3 Our Method

In this section, we introduce URCP which is a capability-aware coordination protocol for heterogeneous multi-model systems. It takes three inputs: a user task T, optional user guidance G, and a resource pool $\Omega = (\mathcal{M}, \mathcal{R}, \mathcal{U}, \mathcal{B}, \mathcal{C})$ containing the available models $\mathcal{M}$, executable resources $\mathcal{R}$, tools $\mathcal{U}$, shared and role-scoped memory $\mathcal{B}$, and authenticated communication channels $\mathcal{C}$. From these inputs URCP constructs and executes a coordination object $\Pi = (\mathcal{F}, \mathcal{B}, \mathcal{S}, \mathcal{A}, \mathcal{O}, \mathcal{H}, \Lambda)$ comprising a task frame $\mathcal{F}$, a subtask set $\mathcal{B}$, a role schema $\mathcal{S}$, an assignment map $\mathcal{A}$, an execution DAG $\mathcal{O}$, handoff rules $\mathcal{H}$, and validation conditions Λ. No component is supplied by an external planner; every element is proposed by the participants and ratified by a supermajority vote. The participants fall into two classes. Models in $\mathcal{M}$ (LLMs, VLMs) are *vote-capable*: they read structured proposals and emit signed votes (YES, NO, REVISE, ABSTAIN) with written justifications. Executable resources in $\mathcal{R}$ (image, audio, and video generators, VLA controllers, code executors, retrieval indices, tool wrappers) contribute capabilities at execution time but do not vote. We call the verified vote-capable set $\mathcal{M}^*$ formed at discovery the *voters*, and an *agent* is a role-bearing instantiation of a model or resource configured with a directive, permitted tools, memory scope, I/O interface, deadline, and completion criterion. Non-voting resources are therefore first-class executors even though they never vote. URCP has no semantic planner, router, or controller. The runtime performs only mechanical services: signed-message broadcast, timeout monitoring, canonicalization of duplicate proposals, eligibility filtering by declared types, stale-message invalidation, append-only audit logging, and maintenance of the shared state structures defined in Appendix A1.2—the capability registry, the specialization registry, the vote ledger, the decision registry, the per-role ranking table, and the output

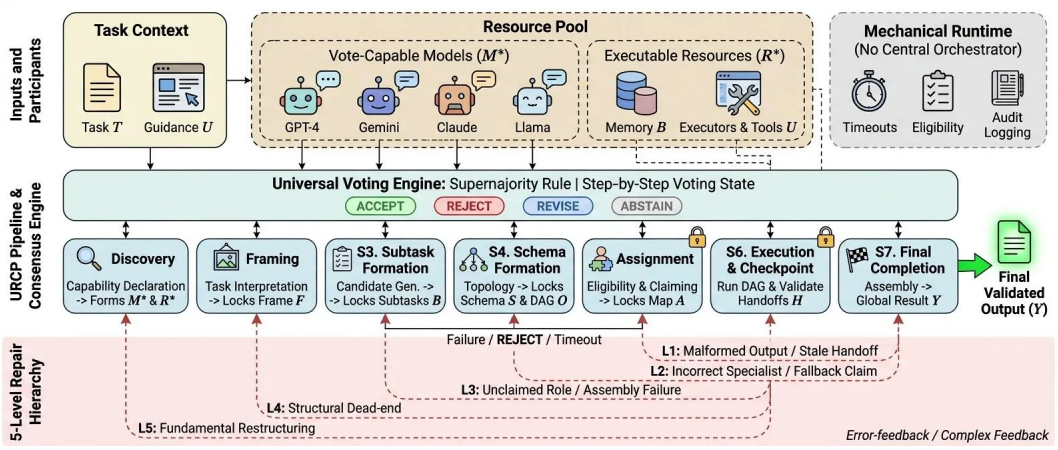


Figure 1: **URCP protocol overview.** Given task T, optional guidance G, and models and executable resources, URCP coordinates seven stages—discovery, framing, subtask formation, schema formation, assignment, execution, and final completion—under the shared supermajority rule Φ_q. Participants collectively ratify the workflow, assignments, checkpoint handoffs, and final output, while a mechanical runtime handles eligibility checks, timeouts, state maintenance, and audit logging. Five repair levels reopen outputs (L_1), assignments (L_2), schemas (L_3), subtasks (L_4), or framing (L_5), preserving unaffected validated work without a central semantic orchestrator.

lock table. It never selects subtasks, creates roles, picks among eligible claimants, chooses repair strategies, or produces outputs. Every semantic decision is made by the voters through proposals and the supermajority rule Φ_q (§3.2). Because every message is signed and timestamped, the full coordination history can be replayed from the log alone: each accepted decision records the proposal it ratified and the voters who approved it, each locked output records the upstream outputs it was built from, and each rejection records the justifications that caused it. Figure 1 gives a high-level overview of URCP, and the full procedural algorithm is provided in Appendix A10.

URCP runs in seven stages, each producing a typed decision object voted on under Φ_q. **Discovery** forms the verified voter set $\mathcal{M}^*$ and resource set $\mathcal{R}^*$ from participant identity broadcasts, and—once these sets are final—publishes a Specialization Registry (SR) listing the canonical executor families for each declared capability class (e.g., image generation $\rightarrow$ diffusion models, code execution $\rightarrow$ code-LLMs, embodied action $\rightarrow$ VLA controllers) so that all voters share the same expectations about which kind of resource is appropriate for which kind of subtask (Appendix A1.4). **Framing** agrees on a task frame $\mathcal{F}$ that fixes the interpretation of T, the expected output form, and the decomposition need. **Subtask formation** produces the accepted subtask set $\mathcal{B}$ through a two-tier vote (per-candidate, then coverage). **Schema formation** organizes the subtasks into a complete role schema $\mathcal{S}$ and execution DAG $\mathcal{O}$, specifying responsibilities, capability requirements, and interfaces. **Assignment** binds each role to an eligible claimant via the assignment map $\mathcal{A}$. **Execution** runs the DAG in topological order, locking each subtask output by checkpoint vote. **Final completion** assembles the locked outputs into the global result Y and submits it for one last ratification.

Stages 1–5 build Π; stages 6–7 execute and ratify it. Failures at any stage trigger structured repair (§3.3). Each stage description below specifies (i) the inputs it consumes, (ii) the message that carries proposals, (iii) the artifact it locks, and (iv) what happens on timeout or stalled vote. After every typed decision, the runtime broadcasts a signed `VOTE-OUTCOME` message to $\mathcal{M}^*$ recording the verdict, tally, and (on ACCEPT) the locked artifact hash, ensuring that every voter holds an identical view of decision history without requiring extra synchronisation queries (Appendix A1.6). Full message schemas, eligibility rules, and exhaustive deadlock-resolution tables are in Appendix A1.

3.1 Stages of URCP

S1. Discovery and capability declaration. URCP first constructs a verified capability table for the current epoch. Each vote-capable model broadcasts an `IDENTITY` message within τ_{disc}, declaring its identifier, public key, modalities, input/output types, available tools, declared capabilities, and

`vote_capable` flag. Executable resources that cannot participate as voters—for example diffusion models, audio/video models, VLA controllers, or tool wrappers—are registered by the protocol runtime from the deployment manifest in Ω as trusted proxy records with `vote_capable=false`. These resources cannot vote, but remain available for compatible execution roles. URCP then forms the active voting set $\mathcal{M}^*$ from verified vote-capable models and the executable resource set $\mathcal{R}^*$ from all verified callable resources. A `CONTEXT-VERIFY` round hashes T, G, Ω, $\mathcal{M}^*$, and $\mathcal{R}^*$ to ensure that all voters share the same task, guidance, resource pool, and participant sets before the epoch begins. Empty $\mathcal{M}^*$ terminates as NO-VOTERS, empty $\mathcal{R}^*$ as NO-RESOURCES, and inconsistent context hashes as CONTEXT-INCONSISTENT; full schemas and edge-case handling are given in Appendix A1.3 and Appendix A1.5.

S2. Framing. With the active sets fixed, voters establish a shared interpretation of the task before any roles or agents are created. Voters broadcast `FRAME-PROPOSE` messages specifying the task interpretation, expected output form, required modalities, and decomposition need (NONE, MINIMAL, or FULL). Suggestions in G are classified as MANDATORY or PREFERENCE; mandatory items act as hard constraints and auto-reject any violating frame, while preferences influence voting without blocking. A frame $\mathcal{F}$ is locked once it clears Φ_q and fixes the interpretation of T for the epoch. If no proposer responds within τ_{propose}, the lexicographically smallest voter ID acts as fallback; failure of the fallback, exhaustion of K_1 revisions, or two consecutive identical revisions terminate the stage as NO-CONSENSUS. Full schema, guidance handling, and tiebreaking are in Appendix A1.7.

S3. Subtask formation. Voters then decompose the locked frame into candidate subtasks. Each `SUBTASK-PROPOSE` message carries one or more candidates, each with description, I/O types, required capabilities, tool needs, and local completion criterion, plus a `coverage_argument` explaining how the proposed set covers T; proposals lacking the argument are auto-rejected. The runtime canonicalizes near-duplicates syntactically but does not judge necessity. Voting is two-tier: each candidate is voted individually under Φ_q, then the provisional set is voted as a whole for coverage. Coverage failure returns to candidate proposal rather than per-subtask voting, and previously accepted candidates are retained unless modified. After K_3 failed coverage rounds the stage terminates as NO-CONSENSUS. Full schemas and two-tier voting semantics are in Appendix A1.8.

S4. Schema formation. The accepted subtasks are then organized into a coherent role schema with a defined execution order. Voters broadcast `ROLE-SCHEMA-PROPOSE` messages containing complete role tuples $s_j = (\text{name}_j, \text{covers}_j, \text{caps}_j, \text{in}_j, \text{out}_j, \text{tools}_j, \text{mem}_j, \tau_{\text{exec},j}, \text{accept_crit}_j)$ and the connecting DAG edges. Schemas are voted atomically rather than role-by-role, since locally coherent roles can form globally incoherent workflows. Four mechanical pre-checks fire before the vote and auto-reject without consuming a revision round: COVERAGE-GAP ($\bigcup_j \text{covers}_j \neq \mathcal{B}$), CYCLIC-DAG, TYPE-MISMATCH (any edge with $\text{out}_i \not\subseteq \text{in}_j$), and UNREACHABLE-ROLE. K_4 exhausted revisions terminate as NO-CONSENSUS; three consecutive same-pre-check failures are flagged STRUCTURAL-DEAD-END and escalate to L_4 repair. Full pre-check algorithms and atomic-proposal rationale are in Appendix A1.9.

S5. Assignment. With the schema locked, each role is bound to a model or resource. Eligible participants broadcast `ROLE-CLAIM` messages justifying capability, modality, and tool fit. A claimant r is eligible for s_j iff $\text{caps}_j \subseteq r.\texttt{capabilities}$, $\text{in}_j \subseteq r.\texttt{input_types}$, $\text{out}_j \subseteq r.\texttt{output_types}$, and $\text{tools}_j \subseteq r.\texttt{tool_interfaces}$; ineligibles are filtered mechanically and never reach a vote. Eligible claimants are voted in timestamp order under the non-self threshold $\lceil q(|\mathcal{M}^*| - 1) \rceil$, since a model cannot vote on roles it claims; the first to clear is bound and remaining claims are dropped. The runtime additionally records a per-role ranking $\text{RRT}[s_j]$ ordering all eligible claimants—the bound model first, then unvoted claimants in claim-timestamp order, and finally any claimants whose own assignment vote ran but did not clear Φ_q, ranked by recorded YES count—so that during L_2 repair the runtime can present fallback candidates to the voters in a deterministic order. Each fallback binding still requires an explicit Φ_q-valid assignment decision; the runtime never substitutes its own choice for the vote (Appendix A1.10, Appendix A1.11). A role with no eligible claimant is flagged NO-SUITABLE-MODEL and reopened at L_3; K_5 rounds with no winner trigger L_3 repair to permit role splitting. Full eligibility computation and self-vote arithmetic are in Appendix A1.10.

S6. Execution and checkpoint validation. With every role bound, the runtime executes $\mathcal{O}$ in topological order. For each s_j, the assigned agent broadcasts `SUBTASK-COMPLETE` carrying the output, evidence, declared output type, and a local completion claim against accept_crit_j. Voters evaluate under Φ_q: ACCEPT locks the output and publishes its canonical hash, REVISE returns suggestions for

up to K_6 rounds, and REJECT triggers repair at the indicated level (typically L_1 for malformed output, L_2 for an incorrectly assigned specialist). Each downstream agent broadcasts `HANDOFF-RECEIVED` citing the locked hash before executing. Three pre-checks bypass the revision budget: output-type mismatch, missing evidence when $\mathrm{accept_crit}_j$ requires it, and stale handoffs. $\tau_{\mathrm{exec},j}$ expiry triggers a runtime `REOPEN` at L_1, τ_{handoff} expiry one at L_2, and concurrent `REOPEN`s at different levels are voted highest-first. Full schemas and cascading-failure handling are in Appendix A1.12.

S7. Final completion. Once every role has a locked output, the final-stage agent assembles and ratifies the global result. It broadcasts `FINAL-SUBMIT` carrying the assembled output Y, component hashes $\{\mathrm{SHA256}(x_j)\}$, an ordered provenance list of (`role_id`, `output_hash`, `vote_summary`) entries, and a global completion claim against $\mathcal{F}$ and Λ. Under Φ_q, ACCEPT terminates as COMPLETE and returns Y, REVISE returns to producing stages for up to K_7 rounds, and REJECT triggers repair at the indicated level. Components individually valid but collectively inconsistent trigger an ASSEMBLY-FAILURE `REOPEN` at L_3; K_7 exhaustion terminates as NO-CONSENSUS with the full audit log. Full schema and assembly-failure handling are in Appendix A1.13.

3.2 Supermajority Decision Rule

Every decision in URCP uses the same supermajority rule. Let $q \in (1/2, 1]$, with default $q = 3/4$ and $q = 1$ recovering unanimity. The threshold is fixed at epoch start and governs context verification, typed coordination decisions, and repair. For decision D, let $V_e(D)$ denote its authorized electorate: normally $\mathcal{M}^*$, but $\mathcal{M}^* \setminus \{r\}$ when assigning a role to claimant $r \in \mathcal{M}^*$. Assignment to a non-voting resource retains the full electorate.

Each eligible voter casts $v_i(D) \in \{\text{YES}, \text{NO}, \text{REVISE}, \text{ABSTAIN}\}$ with a structured justification. Missing ballots at the applicable decision deadline count as ABSTAIN. For $|V_e(D)| > 0$, let $Y(D)$ and $N(D)$ count eligible YES and NO ballots:

$$\Phi_q(D) = \begin{cases} \text{ACCEPT} & Y(D) \geq \lceil q|V_e(D)| \rceil, \\ \text{REJECT} & N(D) \geq \lceil q|V_e(D)| \rceil, \\ \text{REVISE} & \text{otherwise.} \end{cases} \tag{1}$$

An empty electorate produces a typed failure, never zero-vote acceptance. Every ACCEPT therefore requires an explicit affirmative supermajority. Splits meeting neither threshold resolve to REVISE; abstentions contribute to neither tally but can prevent a threshold from being reached. Claimant exclusion prevents a voter from authorizing its own role assignment.

3.3 Repair and Termination

When a stage fails, repair proposes reopening the smallest part of Π sufficient to address the failure: L_1 output, L_2 assignment, L_3 role schema, L_4 subtask set, or L_5 task frame. Any voter may broadcast a `REOPEN` message identifying the affected objects and a typed failure; the runtime may also propose repair through a predefined failure-to-level mapping. Every repair requires ratification under Φ_q by the current full electorate.

On ACCEPT, provided the epoch cap permits the transition, the epoch advances once, $e \to e + 1$, and execution returns to the indicated stage. Preservation is dependency-aware: affected outputs and their dependent descendants are invalidated, while independent validated artifacts may be retained under the compatibility rules of Appendix A1.14. During L_2 repair, stored fallback candidates are presented in deterministic order, and each replacement requires a fresh Φ_q-valid assignment vote. Exhausting fallbacks proposes L_3 repair; it does not authorize an automatic schema change.

Concurrent repair proposals are considered in descending scope. Old-epoch messages cannot modify current state; preserved artifacts remain accessible through explicit carry-forward records. Membership is not reduced to overcome a failed quorum. Participant or resource re-evaluation requires an accepted L_5 transition to discovery or a separate restart. If no repair is authorized within the applicable bounds, the run terminates as NO-CONSENSUS.

Every waiting operation has a finite timeout and a specified transition. Semantic attempt budgets bound voted revisions, while independent non-resetting stage deadlines or equivalent attempt caps also bound mechanical-rejection paths. The epoch cap $E_{\max}$ limits accepted repairs, and repair does

not reset the run deadline. Under Assumptions A1.1–A1.4, an available runtime, and the bounded-processing requirements of Appendix A1.15, execution terminates with an explicit verdict. This guarantees bounded execution, not successful task completion or semantic convergence. Timeout rules and the formal argument appear in Appendices A1.15 and A1.16.

4 Experiments

The formal analysis in Appendix A1.16 establishes termination with an explicit verdict under Assumptions A1.1–A1.4, an available runtime, and the stated processing and deadline bounds. We evaluate task effectiveness, protocol behavior, and scaling robustness. Complete protocol specifications, message schemas, and proofs appear in Appendix A1; emergent coordination patterns and adversarial/sensitivity analyses appear in §A3 and §A7.

4.1 Setup

Benchmarks. We use three primary benchmarks. *MultiAgentBench* (MAB) [47] reports milestone-level multi-agent coordination metrics; τ*-bench* [40] evaluates tool–agent–user interaction under domain policies and multi-turn constraints; and *MemoryAgentBench* (MemA) [13] evaluates memory-dependent capabilities including retrieval, test-time learning, and conflict resolution. Additional evaluations on GAIA [23], REALM-Bench [8], and the multimodal evaluation groups associated with τ-bench and MAB are reported in Appendix A4.6, together with their scope and interpretation.

Participant pool. Following the two-class participant model (§A1.1), we instantiate twelve vote-capable models—GPT-4.1, GPT-4.1 mini [29]; Claude Opus 4, Sonnet 4 [3]; Gemini 2.5 Pro, Flash [9]; Llama 4 Maverick [22]; Qwen3-235B-A22B [38]; DeepSeek-R1 [5]; Mistral Medium 3 [24]; Phi-4-reasoning [1]; and Command A [4]—and six non-voting executable resources: a code sandbox, retrieval index, OCR engine, image generator, TTS engine, and constrained-decoding form filler. The latter are registered through manifest-proxy `IDENTITY` records. Following discovery and verification (Appendix A1.3), the pool contains $|\mathcal{M}^*| = 12$ voters and $|\mathcal{R}^*| = 18$ executable participants, including the twelve models. Resource declarations are checked against the Specialization Registry (§A1.4). Per-model declarations and the full registry appear in Appendix A4.1. To examine the contribution of team composition, we also evaluate four homogeneous configurations and an open-weight subset of six models.

Scaling protocol. URCP does not fix the number of execution roles or workflow depth in advance. These are determined during subtask formation (Appendix A1.8), schema formation (Appendix A1.9), and assignment (Appendix A1.10). For the task-complexity scaling experiments, we decompose tasks into more detailed items, add tool and evidence requirements, and extend episode length while retaining the original objective. We then report the number of role-bearing executor instances n and DAG depth d that URCP instantiates. These quantities describe the execution workflow, not the number of distinct voters.

Implementation. Vote-capable models use temperature 0, top-$p = 1$, and an output limit of $4{,}096$ tokens. Coordination messages are requested as structured JSON and checked against the message schemas in Appendix A1. Open-weight models are served via vLLM with 4-GPU tensor parallelism on a single $8\times$A100 (80 GB) node; API request rates are matched to local throughput. The runtime records state transitions in a single-writer append-only log and maintains the active views described in §A1.2. It broadcasts a signed `VOTE-OUTCOME` after each typed decision (§A1.6); storage details appear in Appendix A4.2. Default hyperparameters (Table 2) were selected on a 10% held-out MAB split and retained except in the explicitly identified sensitivity experiments (§A7.3). The default decision threshold is $q = 3/4$; threshold sensitivity is reported in §A7.2. All baselines receive identical tools, memory, executable resources, and budgets (500 K tokens and 600 s wall clock); their coordination procedures differ.

4.2 Baselines

We compare against eight systems in three groups: *protocol baselines*—A2A [10], MCP [2], and ACP [14]; *centrally orchestrated dynamic systems*—MetaGen [35], AFLOW [42], and DynRole [43]; and *decentralized systems*—AgentNet [39] and ReSo [44]. Protocol baselines are instantiated with

Table 2: Default URCP hyperparameters; departures are identified in the sensitivity experiments (Table 27).

τ_{disc}	30 s	$\tau_{\text{exec},j}$	120 s	q (decision threshold)	$3/4$
τ_{verify}	15 s	τ_{handoff}	30 s	$E_{\max}$ (epoch cap)	11
τ_{vote}	45 s	τ_{repair}	30 s	K_ℓ (max revisions)	5
τ_{propose}	30 s	Token budget	500 K	Wall-clock budget	600 s

Table 3: Task performance on the three primary benchmarks: MAB (MultiAgentBench), τ-bench, and MemA (MemoryAgentBench). Mean $\pm$ 95% CI over 5 seeds; **bold** = best, underlined = second-best.

Bench.	**A2A**	**MCP**	**ACP**	**MetaG.**	**AFLOW**	**DynR.**	**AgNet**	**ReSo**	**URCP**
MAB	$54.7_{\pm 1.7}$	$52.1_{\pm 1.9}$	$55.3_{\pm 1.6}$	$62.4_{\pm 1.5}$	$60.8_{\pm 1.6}$	$58.9_{\pm 1.7}$	$59.3_{\pm 1.6}$	$61.2_{\pm 1.5}$	$\mathbf{72.1_{\pm 1.2}}$
τ-b	$48.3_{\pm 2.0}$	$46.9_{\pm 2.2}$	$49.1_{\pm 1.9}$	$55.6_{\pm 1.8}$	$54.2_{\pm 1.9}$	$52.8_{\pm 2.0}$	$53.7_{\pm 1.8}$	$54.9_{\pm 1.7}$	$\mathbf{63.5_{\pm 1.5}}$
MemA	$41.6_{\pm 2.3}$	$39.8_{\pm 2.5}$	$42.4_{\pm 2.2}$	$50.3_{\pm 2.1}$	$48.7_{\pm 2.2}$	$47.1_{\pm 2.3}$	$49.2_{\pm 2.1}$	$50.8_{\pm 2.0}$	$\mathbf{61.4_{\pm 1.7}}$

round-robin role assignment; the other systems retain their published coordination settings within the shared evaluation interface.

For the main controlled comparison, adapter-level retry and reconnect wrappers are disabled for all baselines, exposing failures to the coordination layer (Appendix A4.3). The resulting comparisons apply to these configured systems. They do not establish performance against every production configuration or isolate individual URCP components. Supplementary comparisons with retries enabled are reported in Appendix A11. The robustness perturbations and limits of cross-system comparison are specified in Appendices A4.4 and A4.5.

4.3 Main results

Table 3 reports task performance for the heterogeneous twelve-model team under the controlled baseline configuration. URCP outperforms all eight baselines on every benchmark, with gains of $+9.7$, $+7.9$, and $+10.6$ points over the strongest baseline on each: MetaGen on MAB and τ-bench, and ReSo on MemA.

The protocol baselines trail URCP by approximately 14–22 points, while systems with automated coordination narrow the gap. URCP's largest gain over the strongest comparator occurs on MemA ($+10.6$ points), consistent with the potential value of validated handoffs and repair in memory-dependent workflows. These comparisons establish an advantage for the complete URCP configuration; they do not by themselves attribute that advantage to voting, schema formation, or checkpoint validation individually. Component analyses are discussed in Appendix A11.

4.4 Protocol behavior, scaling, robustness, and team composition

Tables 4 and 5, together with Figure 2, characterize behavior beyond task performance. The robustness study appears in Table 9 in Appendix A2; condition definitions and comparison limits appear in Appendices A4.4 and A4.5. Per-level repair analysis appears in §A9.7.

Across the three benchmarks, URCP forms a coordination object in most runs and requires 1.9 decision rounds on average. Its average handoff failure rate is 9.8%, compared with 14.7% for MetaGen and 16.3% for AgentNet. Every evaluated run terminates without deadlock, consistent with the conditional termination result in Appendix A1.16. Figure 2(A) decomposes the 24.9% average repair rate across the hierarchy in Appendix A1.14. Most repair triggers occur at the output and assignment levels; schema, subtask, and framing repairs are less frequent.

Table 5 shows that URCP expands its execution structure with task complexity: the median agent-instance count increases from 5 to 73, and median DAG depth from 2 to 28. These are execution instances drawn from the participant pool, not an expansion of the voting electorate. Performance decreases by 5.2 points from T1 to T5 and by 9.5 points across the observed depth bins. All listed baselines lose more than 15 points between the shallowest and deepest bins. Because depth is observed rather than independently randomized, these results describe performance across workflow regimes; they do not isolate a causal effect of depth.

Table 4: Protocol-level metrics (mean over 5 seeds); — indicates the metric is undefined for that system. The avg. column averages URCP's three per-benchmark values. The per-level marginal repair-trigger breakdown appears in Figure 2(A), following Table 6. *Takeaway:* URCP uses fewer than two decision rounds on average, has approximately one-third fewer handoff failures than MetaGen (9.8% vs. 14.7%), and terminates without deadlock in every evaluated run. STD ≤ 0.4.

Metric	**MetaGen**	**AgentNet**	**URCP**			
			MAB	τ-b	MemA	avg.
Coord.-object form. rate (%)	—	—	91.3	88.7	85.4	88.5
Avg. decision rounds	1.1	—	1.4	1.8	2.6	1.9
NO-CONSENSUS rate (%)	—	—	2.1	3.5	4.8	3.5
Repair rate (%)	12.4	—	18.7	24.3	31.6	24.9
Handoff failure (%)	14.7	16.3	6.4	9.1	13.8	9.8
Deadlock-free termination (%)	—	—	100	100	100	100

Table 5: Scaling on MABench (heterogeneous team, mean over 5 seeds). Left: performance by task-complexity tier with URCP's median observed agent-instance count n and DAG depth d. Right: performance by observed sequential DAG depth; Δ is the drop from the shallowest to deepest depth bin. *Takeaway:* URCP degrades the least across both task-complexity tiers and observed depth bins, while the baselines drop more sharply in the deepest regime. STD ≤ 0.5.

	Complexity tier					Observed depth bin				
Method	T1	T2	T3	T4	T5	2–4	5–10	11–20	21+	Δ
URCP med. n/d	5/2	11/5	18/9	38/17	73/28	—	—	—	—	
A2A	60.1	56.8	54.1	50.2	46.4	58.9	55.2	50.6	43.8	-15.1
MetaGen	68.2	64.7	61.9	58.3	54.6	66.8	62.1	57.4	50.7	-16.1
AFLOW	66.7	63.4	60.2	56.8	52.5	65.1	60.7	55.8	48.6	-16.5
AgentNet	65.1	61.9	58.6	55.2	50.8	63.8	59.1	54.4	46.9	-16.9
ReSo	67.0	63.8	60.9	57.4	53.1	65.6	61.0	56.5	49.8	-15.8
URCP	**74.3**	**73.1**	**72.4**	**70.6**	**69.1**	**74.8**	**72.7**	**68.9**	**65.3**	$\mathbf{-9.5}$

Under the failure-timing conventions in Appendix A4.4, Table 9 reports recovery above 86% for the tested agent-failure conditions up to 20%, and deadlock-free termination in all eleven perturbation conditions. These results do not imply recovery under arbitrary voter dropout: mid-epoch failures leave the authorized electorate and its threshold unchanged. When the required decisions cannot be ratified within the applicable bounds, URCP terminates with an explicit NO-CONSENSUS verdict.

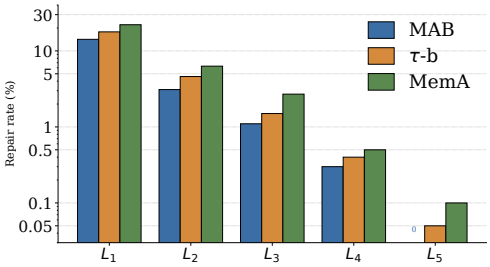
Figure 2(B) shows that the heterogeneous team exceeds the best homogeneous score on each benchmark by 4.7, 4.3, and 5.3 points, respectively, averaging approximately 4.8 points. The open-weight subset trails the full pool by 3.0 points on average, and even the weakest homogeneous configuration outperforms every protocol baseline in Table 3. The results show that URCP operates across the tested team compositions while also benefiting from the heterogeneous pool; they do not establish independence from model choice. Detailed complexity analysis appears in Appendix A8.

4.5 Critical Observations from URCP Traces

Across approximately $5{,}400$ runs, the traces reveal recurring patterns in how participants organize execution through voting. Frontier models—GPT-4.1, Claude Opus 4, Gemini 2.5 Pro, and DeepSeek-R1—constitute 33.3% of the voter pool but receive 73.9% of critical-path roles (Appendix A3.1). Smaller models also contribute through governance even when they receive few execution roles. For example, the traces include a compact model requesting a missing conflict-checking step that subsequently appears in a ratified schema (Appendix A3.2). Such a request can inform revision, but a single REVISE vote does not override the Φ_q decision rule.

In 5.9% of runs, a smaller specialist or non-voting resource receives an important role (Appendix A3.3). These assignments show that role selection does not always favor the largest models. Critic, verifier, or consistency-checker roles appear in 63.5% of accepted schemas with at least three roles and in 91.7% of schemas with eight or more roles (Appendix A3.4). This association concerns

(A) Repair breakdown



(B) Team composition

Team	MAB	τ-b	MemA
Homo (GPT-4.1)	$67.4_{\pm 1.4}$	$59.2_{\pm 1.6}$	$55.3_{\pm 1.9}$
Homo (Opus 4)	$66.8_{\pm 1.5}$	$58.7_{\pm 1.7}$	$56.1_{\pm 2.0}$
Homo (GM2.5P)	$65.9_{\pm 1.6}$	$57.8_{\pm 1.8}$	$53.8_{\pm 2.1}$
Homo (DSR1)	$63.2_{\pm 1.7}$	$55.4_{\pm 1.9}$	$51.6_{\pm 2.2}$
Open-weight (6)	$69.3_{\pm 1.3}$	$60.8_{\pm 1.5}$	$57.9_{\pm 1.8}$
Hetero (12)	$\mathbf{72.1_{\pm 1.2}}$	$\mathbf{63.5_{\pm 1.5}}$	$\mathbf{61.4_{\pm 1.7}}$

Figure 2: **(A)** Per-benchmark URCP marginal repair-trigger rate (%) across the five-level hierarchy of Table 6, log scale. The MAB L_5 bar is exactly 0 (annotated). Per-benchmark totals match the "Repair rate" row of Table 4; pooled marginal averages are $L_1 = 18.0$, $L_2 = 4.7$, $L_3 = 1.8$, $L_4 = 0.4$, and $L_5 = 0.05$. **(B)** Team composition under URCP, reported as mean $\pm$ 95% CI over 5 seeds. Abbreviations: GM2.5P = Gemini 2.5 Pro; DSR1 = DeepSeek-R1. *Takeaway:* repair triggers concentrate at the local output level and decrease at broader levels; the heterogeneous team wins on every benchmark, and even the weakest homogeneous setup beats every protocol baseline.

schema size, not DAG depth directly. The execution structure also varies with task requirements: role-bearing executor counts remain comparatively small on τ-bench and MemoryAgentBench and grow on high-tier MAB tasks with deeper decompositions (Appendix A3.5).

Encrypted handoffs occur in 5.7% of τ-bench runs and 4.1% of MemoryAgentBench runs (Appendix A3.6). Public hashes and redacted summaries permit checks of the properties they expose; their presence does not establish the correctness of hidden plaintext. The evidence requirements and limitations are detailed in Appendix A6. In multimodal traces, vision-capable agents provide visual evidence for downstream reasoning, while unresolved coordination decisions can terminate as auditable NO-CONSENSUS outcomes (Appendices A3.7 and A3.8). Neither checkpoint approval nor a termination verdict alone guarantees semantic correctness.

These observations document adaptive assignment, explicit checking roles, and bounded failure handling under URCP. They complement the aggregate results without independently establishing the causal contribution of each mechanism. Additional discussion of governance semantics, guarantee conditions, supplementary experiments, and deployment trade-offs appears in Appendix A11.

5 Related Work

URCP connects to agent benchmarks, model routing, and multi-agent coordination. Recent benchmarks evaluate long-horizon, tool-rich, multimodal, and collaborative agents [47, 40, 23, 46, 17, 37, 13], while routing work shows that different models excel on different tasks [12, 34, 26, 6]. Multi-agent systems such as CAMEL, ChatDev, MetaGPT, AFlow, MetaGen, AgentNet, and ReSo show that agents can collaborate or self-organize [18, 30, 11, 42, 35, 39, 44]. URCP differs by making roles, assignments, handoffs, repair, and final acceptance explicit objects of supermajority agreement rather than relying on fixed roles, a central planner, or transport-only protocols [10, 2, 14].

6 Conclusion

We introduced URCP, a coordination protocol in which heterogeneous models collectively decide the task frame, roles, assignments, checkpointed execution, repair, and final output. URCP adds governance beyond transport protocols, removes the need for a central semantic orchestrator, and makes failures auditable through bounded repair and explicit NO-CONSENSUS outcomes. The results suggest that multi-agent gains come not only from stronger models, but from making coordination itself voted, visible, and repairable. Deployment considerations are discussed in Appendix A9.

Acknowledgments

The authors were partially supported by the Defense Advanced Research Projects Agency (DARPA) under Agreement No. HR0011-26-9-E231 and under subaward No. 800018648; by the National Science Foundation under Award No. 2610233; and by University of Florida startup funds. The views and conclusions expressed are those of the authors and do not necessarily reflect the official position of the sponsors.

Competing interests. The authors declare no competing interests.

This material is based upon work co-supported by the U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research under Contract No. DE-AC05-00OR22725.

This manuscript has been co-authored by UT-Battelle, LLC under Contract No. DE-AC05-00OR22725 with the U.S. Department of Energy. The United States Government retains and the publisher, by accepting the article for publication, acknowledges that the United States Government retains a non-exclusive, paid-up, irrevocable, world-wide license to publish or reproduce the published form of this manuscript, or allow others to do so, for United States Government purposes. The Department of Energy will provide public access to these results of federally sponsored research in accordance with the DOE Public Access Plan (`http://energy.gov/downloads/doepublic-access-plan`).

References

[1] Marah Abdin, Sahaj Agarwal, Ahmed Awadallah, Vidhisha Balachandran, Gustavo de Rosa, Suriya Gunasekar, Harkirat Behl, Lingjiao Chen, Mojan Javaheripi, Piero Kauffmann, Caio César Teodoro Mendes, Neel Joshi, Yash Lara, Arindam Mitra, Besmira Nushi, Dimitris Papailiopoulos, Olli Saarikivi, Shital Shah, Vaishnavi Shrivastava, Vibhav Vineet, Yue Wu, Safoora Yousefi, and Guoqing Zheng. Phi-4-reasoning technical report. *arXiv preprint arXiv:2504.21318*, 2025.

[2] Anthropic. Introducing the Model Context Protocol. `https://www.anthropic.com/news/model-context-protocol`, 2024.

[3] Anthropic. Introducing Claude 4: Claude opus 4 and Claude sonnet 4. Model release announcement, 22 May 2025, 2025. URL `https://www.anthropic.com/news/claude-4`.

[4] Cohere. Command A: An enterprise-grade foundation model. Model release announcement, March 2025, 2025. URL `https://cohere.com/blog/command-a`. Hugging Face model identifier: `c4ai-command-a-03-2025`.

[5] DeepSeek-AI, Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, Xiaokang Zhang, Xingkai Yu, Yu Wu, et al. DeepSeek-R1: Incentivizing reasoning capability in LLMs via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.

[6] Dujian Ding, Ankur Mallick, Chi Wang, Robert Sim, Subhabrata Mukherjee, Victor Ruhle, Laks V. S. Lakshmanan, and Ahmed Hassan Awadallah. Hybrid llm: Cost-efficient and quality-aware query routing, 2024. URL `https://arxiv.org/abs/2404.14618`.

[7] Abul Ehtesham, Aditi Singh, Gaurav Kumar Gupta, and Saket Kumar. A survey of agent interoperability protocols: MCP, ACP, A2A, and ANP. *arXiv:2505.02279*, 2025.

[8] Longling Geng and Edward Y. Chang. REALM-Bench: A benchmark for evaluating multi-agent systems on real-world, dynamic planning and scheduling tasks. *arXiv preprint arXiv:2502.18836*, 2025.

[9] Google DeepMind. Gemini 2.5: Gemini 2.5 Pro and Gemini 2.5 Flash. Model release announcement, March 2025, 2025. URL `https://blog.google/technology/google-deepmind/gemini-model-thinking-updates-march-2025/`.

[10] Google Developers Blog. Announcing the Agent2Agent protocol (A2A). `https://developers.googleblog.com/en/a2a-a-new-era-of-agent-interoperability/`, 2025.

[11] Sirui Hong, Mingchen Zhuge, Jonathan 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. In *International Conference on Learning Representations (ICLR)*, 2024. URL https://arxiv.org/abs/2308.00352.

[12] Qitian Jason Hu, Jacob Bieker, Xiuyu Li, Nan Jiang, Benjamin Keigwin, Gaurav Ranganath, Kurt Keutzer, and Shriyash Kaustubh Upadhyay. RouterBench: A benchmark for multi-LLM routing system. *arXiv:2403.12031*, 2024.

[13] Yuanzhe Hu, Yu Wang, and Julian McAuley. Evaluating memory in LLM agents via incremental multi-turn interactions. *arXiv preprint arXiv:2507.05257*, 2025.

[14] IBM Research and Linux Foundation. Agent communication protocol (ACP). Open standard, launched March 2025; merged with A2A under the Linux Foundation, August 2025, 2025. URL https://agentcommunicationprotocol.dev/.

[15] Naman Jain, King Han, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. LiveCodeBench: Holistic and contamination free evaluation of large language models for code. *arXiv preprint arXiv:2403.07974*, 2024. URL https://arxiv.org/abs/2403.07974.

[16] Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. SWE-bench: Can language models resolve real-world GitHub issues? In *International Conference on Learning Representations (ICLR)*, 2024. URL https://arxiv.org/abs/2310.06770.

[17] Jing Yu Koh, Robert Lo, Lawrence Jang, Vikram Duvvur, Ming Chong Lim, Po-Yu Huang, Graham Neubig, Shuyan Zhou, Ruslan Salakhutdinov, and Daniel Fried. VisualWebArena: Evaluating multimodal agents on realistic visual web tasks. In *Annual Meeting of the Association for Computational Linguistics (ACL)*, 2024. URL https://arxiv.org/abs/2401.13649.

[18] 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. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023. URL https://arxiv.org/abs/2303.17760.

[19] Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, Shudan Zhang, Xiang Deng, Aohan Zeng, Zhengxiao Du, Chenhui Zhang, Sheng Shen, Tianjun Zhang, Yu Su, Huan Sun, Minlie Huang, Yuxiao Zhao, and Jie Tang. AgentBench: Evaluating LLMs as agents. In *International Conference on Learning Representations (ICLR)*, 2024. URL https://arxiv.org/abs/2308.03688.

[20] Yuan Liu, Haodong Duan, Yuanhan Zhang, Bo Li, Songyang Zhang, Wangbo Zhao, Yike Yuan, Jiaqi Wang, Conghui He, Ziwei Liu, Kai Chen, and Dahua Lin. MMBench: Is your multi-modal model an all-around player? *arXiv preprint arXiv:2307.06281*, 2024. URL https://arxiv.org/abs/2307.06281.

[21] Pan Lu, Hritik Bansal, Tony Xia, Jiacheng Liu, Chunyuan Li, Hannaneh Hajishirzi, Hao Cheng, Kai-Wei Chang, Michel Galley, and Jianfeng Gao. MathVista: Evaluating mathematical reasoning of foundation models in visual contexts. In *International Conference on Learning Representations (ICLR)*, 2024. URL https://arxiv.org/abs/2310.02255.

[22] Meta AI. The Llama 4 herd: Native multimodal AI innovation (Llama 4 Scout and Llama 4 Maverick). Model release announcement, 5 April 2025, 2025. URL https://ai.meta.com/blog/llama-4-multimodal-intelligence/.

[23] Grégoire Mialon, Clémentine Fourrier, Craig Swift, Thomas Wolf, Yann LeCun, and Thomas Scialom. Gaia: a benchmark for general ai assistants, 2023. URL https://arxiv.org/abs/2311.12983.

[24] Mistral AI. Mistral Medium 3: Frontier performance at lower cost. Model release announcement, 7 May 2025, 2025. URL https://mistral.ai/news/mistral-medium-3.

[25] Yoav Nir and Adam Langley. ChaCha20 and Poly1305 for IETF protocols. RFC 8439, RFC Editor, June 2018. URL https://www.rfc-editor.org/rfc/rfc8439.html.

[26] Isaac Ong, Amjad Almahairi, Vincent Wu, Wei-Lin Chiang, Tianhao Wu, Joseph E. Gonzalez, M Waleed Kadous, and Ion Stoica. Routellm: Learning to route llms with preference data, 2025. URL https://arxiv.org/abs/2406.18665.

[27] Isaac Ong, Amjad Almahairi, Vincent Wu, Wei-Lin Chiang, Tianhao Wu, Joseph E. Gonzalez, M Waleed Kadous, and Ion Stoica. RouteLLM: Learning to route LLMs from preference data. In *International Conference on Learning Representations*, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/5503a7c69d48a2f86fc00b3dc09de686-Abstract-Conference.html.

[28] Diego Ongaro and John Ousterhout. In search of an understandable consensus algorithm. In *2014 USENIX Annual Technical Conference (USENIX ATC 14)*, pages 305–319, Philadelphia, PA, 2014. USENIX Association. ISBN 978-1-931971-10-2. URL https://www.usenix.org/conference/atc14/technical-sessions/presentation/ongaro.

[29] OpenAI. GPT-4.1 and GPT-4.1 mini. Model release announcement, 14 April 2025, 2025. URL https://openai.com/index/gpt-4-1/.

[30] Chen Qian, Wei Liu, Hongzhang Liu, Nuo Chen, Yufan Dang, Jiahao Li, Cheng Yang, Weize Chen, Yusheng Su, Xin Cong, Juyuan Xu, Dahai Li, Zhiyuan Liu, and Maosong Sun. ChatDev: Communicative agents for software development. In *Annual Meeting of the Association for Computational Linguistics (ACL)*, 2024. URL https://arxiv.org/abs/2307.07924.

[31] Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023. URL https://arxiv.org/abs/2302.04761.

[32] Yongliang Shen, Kaitao Song, Xu Tan, Wenqi Zhang, Kan Ren, Siyu Yuan, Weiming Lu, Dongsheng Li, and Yueting Zhuang. Taskbench: Benchmarking large language models for task automation, 2024. URL https://arxiv.org/abs/2311.18760.

[33] Noah Shinn, Federico Cassano, Edward Berman, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023. URL https://arxiv.org/abs/2303.11366.

[34] Tal Shnitzer, Anthony Ou, Mírian Silva, Kate Soule, Yuekai Sun, Justin Solomon, Neil Thompson, and Mikhail Yurochkin. Large language model routing with benchmark datasets. *arXiv:2309.15789*, 2023.

[35] Yimeng Wang, Jiaxing Zhao, Hongbin Xie, Hexing Ma, Yuzhen Lei, Shuangxue Liu, Xuan Song, Zichen Zhang, and Haoran Zhang. Metagen: Self-evolving roles and topologies for multi-agent llm reasoning, 2026. URL https://arxiv.org/abs/2601.19290.

[36] Yimeng Wang, Jiaxing Zhao, Hongbin Xie, Hexing Ma, Yuzhen Lei, Shuangxue Liu, Xuan Song, Zichen Zhang, and Haoran Zhang. MetaGen: Self-evolving roles and topologies for multi-agent LLM reasoning, 2026. URL https://arxiv.org/abs/2601.19290.

[37] Tianbao Xie, Danyang Zhang, Jixuan Chen, Xiaochuan Li, Siheng Zhao, Ruisheng Cao, Toh Jing Hua, Zhoujun Cheng, Dongchan Shin, Fangyu Lei, Yitao Liu, Yiheng Xu, Shuyan Zhou, Silvio Savarese, Caiming Xiong, Victor Zhong, and Tao Yu. OSWorld: Benchmarking multimodal agents for open-ended tasks in real computer environments. In *Advances in Neural Information Processing Systems*, 2024. URL https://arxiv.org/abs/2404.07972.

[38] An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jing Zhou, Jingren Zhou, Junyang Lin, Kai Dang, Keqin Bao, Kexin Yang, Le Yu, Lianghao Deng, Mei Li, Mingfeng Xue, Mingze Li, Pei Zhang, Peng Wang, Qin

Zhu, Rui Men, Ruize Gao, Shixuan Liu, Shuang Luo, Tianhao Li, Tianyi Tang, Wenbiao Yin, Xingzhang Ren, Xinyu Wang, Xinyu Zhang, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yinger Zhang, Yu Wan, Yuqiong Liu, Zekun Wang, Zeyu Cui, Zhenru Zhang, Zhipeng Zhou, and Zihan Qiu. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

[39] Yingxuan Yang, Huacan Chai, Shuai Shao, Yuanyi Song, Siyuan Qi, Renting Rui, and Weinan Zhang. Agentnet: Decentralized evolutionary coordination for llm-based multi-agent systems, 2025. URL https://arxiv.org/abs/2504.00587.

[40] Shunyu Yao, Noah Shinn, Pedram Razavi, and Karthik Narasimhan. τ-bench: A benchmark for tool-agent-user interaction in real-world domains, 2024. URL https://arxiv.org/abs/2406.12045.

[41] Xiang Yue, Yuansheng Ni, Kai Zhang, Tianyu Zheng, Ruoqi Liu, Ge Zhang, Samuel Stevens, Dongfu Jiang, Weiming Ren, Yuxuan Sun, Cong Wei, Bo Yu, Ruibin Yuan, Renliang Sun, Ming Yin, Boyuan Zheng, Zhenzhu Yang, Yibo Liu, Wenhao Huang, Huan Sun, Yu Su, and Wenhu Chen. MMMU: A massive multi-discipline multimodal understanding and reasoning benchmark for expert AGI. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2024. URL https://arxiv.org/abs/2311.16502.

[42] Jiayi Zhang, Jinyu Xiang, Zhaoyang Yu, Fengwei Teng, Xionghui Chen, Jiaqi Chen, Mingchen Zhuge, Xin Cheng, Sirui Hong, Jinlin Wang, Bingnan Zheng, Bang Liu, Yuyu Luo, and Chenglin Wu. Aflow: Automating agentic workflow generation, 2025. URL https://arxiv.org/abs/2410.10762.

[43] Miao Zhang, Junsik Kim, Siyuan Xiang, Jian Gao, and Cheng Cao. Dynamic role assignment for multi-agent debate. *arXiv preprint arXiv:2601.17152*, 2026.

[44] Heng Zhou, Hejia Geng, Xiangyuan Xue, Li Kang, Yiran Qin, Zhiyong Wang, Zhenfei Yin, and Lei Bai. Reso: A reward-driven self-organizing llm-based multi-agent system for reasoning tasks, 2025. URL https://arxiv.org/abs/2503.02390.

[45] Heng Zhou, Hejia Geng, Xiangyuan Xue, Li Kang, Yiran Qin, Zhiyong Wang, Zhenfei Yin, and Lei Bai. ReSo: A reward-driven self-organizing LLM-based multi-agent system for reasoning tasks, 2025. URL https://arxiv.org/abs/2503.02390.

[46] Shuyan Zhou, Frank F Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, Uri Alon, and Graham Neubig. WebArena: A realistic web environment for building autonomous agents. In *International Conference on Learning Representations (ICLR)*, 2024. URL https://arxiv.org/abs/2307.13854.

[47] Kunlun Zhu, Hongyi Du, Zhaochen Hong, Xiaocheng Yang, Shuyi Guo, Zhe Wang, Zhenhailong Wang, Cheng Qian, Xiangru Tang, Heng Ji, and Jiaxuan You. MultiAgentBench: Evaluating the collaboration and competition of LLM agents. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 8580–8622, Vienna, Austria, 2025. Association for Computational Linguistics. URL https://aclanthology.org/2025.acl-long.421/.

Appendix
Contents

1 Introduction **2**

2 Motivation and Background **2**

3 Our Method **3**

3.1 Stages of URCP 4

3.2 Supermajority Decision Rule 6

3.3 Repair and Termination 6

4 Experiments **7**

4.1 Setup 7

4.2 Baselines 7

4.3 Main results 8

4.4 Protocol behavior, scaling, robustness, and team composition 8

4.5 Critical Observations from URCP Traces 9

5 Related Work **10**

6 Conclusion **10**

Appendix **15**

A1 Complete Protocol Specification **18**

A1.1 System Model and Assumptions 18

A1.2 Runtime Shared State and Memory Model 19

A1.3 Stage 1: Discovery and Capability Declaration 20

A1.4 Specialization Registry Publication 23

A1.5 Stage 1b: Context Verification 23

A1.6 Vote Message Format and Tally Broadcast 24

A1.7 Stage 2: Task Framing 26

A1.8 Stage 3: Subtask Formation 26

A1.9 Stage 4: Role Schema Formation 27

A1.10 Stage 5: Model-Agent Assignment 28

A1.11 Role Ranking Table Construction 29

A1.12 Stage 6: Execution and Checkpoint Validation 30

A1.13 Stage 7: Final Completion 31

A1.14 Repair: Mechanics and Per-Level Model Selection 31

A1.15 Timeouts and Deadlock Reference 32

A1.16 Formal Properties 33

A2 Robustness Results **36**

A3 Emergent Coordination Phenomena **36**

A3.1 Capability-Aware Role Concentration . . . 37
A3.2 Governance Without Execution . . . 39
A3.3 Specialization Over Size . . . 41
A3.4 Spontaneous Critic Roles . . . 42
A3.5 Self-Pruning of the Agent Pool . . . 43
A3.6 Secure Handoff Subchannels . . . 44
A3.7 Multimodal Coordination Patterns . . . 45
A3.8 NO-CONSENSUS as a Meaningful Safety Outcome . . . 46
A3.9 Integration With External Frameworks . . . 47
A3.10 Summary and Limitations of the Trace Analysis . . . 48

A4 Experimental Details **49**
A4.1 Participant Pool and Specialization Registry . . . 49
A4.2 Runtime Implementation and Shared-State Layout . . . 51
A4.3 Baseline Configuration Details . . . 52
A4.4 Robustness Failure-Timing Conventions . . . 52
A4.5 Scope of the Robustness Comparison . . . 53
A4.6 Additional Benchmarks and Multimodal Subsets . . . 54

A5 Emergent-Phenomena Trace Analysis: Details **55**

A6 Secure-Handoff Compatibility Condition and Possible Extensions **56**

A7 Adversarial Robustness and Sensitivity Analysis **57**
A7.1 Adversarial Voting Behavior . . . 58
A7.2 Voting-Threshold Sensitivity . . . 61
A7.3 Protocol Hyperparameter Sensitivity . . . 62

A8 Protocol Overhead and Complexity Analysis **63**
A8.1 Asymptotic Message Complexity . . . 64
A8.2 Computational Complexity . . . 64
A8.3 Empirical Coordination Overhead . . . 65
A8.4 Scaling with Voter Set Size . . . 65
A8.5 Scaling with DAG Depth . . . 66
A8.6 Storage and Auditability Overhead . . . 66
A8.7 Summary . . . 67

A9 Practical Deployment and Real-World Limitations **67**
A9.1 Integration with Production APIs and Legacy Systems . . . 67
A9.2 Real-World Pilot Experience . . . 68
A9.3 Limitations and Failure Modes Observed in Practice . . . 68
A9.4 Generalization Beyond LLMs . . . 69
A9.5 When URCP Is Not the Right Choice . . . 69
A9.6 Summary . . . 70

A9.7 Repair-Level Effectiveness 70

A10 Algorithm **72**

A11 Additional Experiments **78**

A11.1 Unified governance and the meaning of orchestrator-free coordination 78

A11.2 Guarantee conditions, quorum arithmetic, and bounded repair 79

A11.3 Mechanical-runtime recovery and infrastructure availability 80

A11.4 Native-retry and quality-aware routing comparisons 80

A11.5 Assignment, repair, and heterogeneous-team effects 81

A11.6 Execution concentration and voting behavior 82

A11.7 Committee voting and participation-efficient variants 82

A11.8 Selecting a voter pool and interpreting coordination cost 83

A11.9 Deployment scope, privacy, and reproducibility 84

A11.10 Scaling to One Million Logical Agents 85

A1 Complete Protocol Specification

This appendix specifies URCP's participant model, shared state, message formats, voting rules, execution stages, and repair transitions. Sections A1.1–A1.2 define the system model and runtime state. Sections A1.3–A1.5 describe discovery and context verification. Section A1.6 defines the common decision rule. Sections A1.7–A1.13 specify workflow formation, assignment, execution, and completion. Sections A1.14–A1.16 describe repair, bounded execution, and the scope of the protocol guarantees.

Throughout, $\mathcal{M}^*$ denotes the verified voter set, $\mathcal{R}^*$ the verified executable-resource set, $q \in (1/2, 1]$ the voting threshold, and e the current epoch. A participant may belong to both sets. An agent is a role-bearing instance of a participant with an assigned interface, tool scope, memory scope, deadline, and completion criterion.

Message envelope and identifiers. Every protocol message carries a common envelope containing its message kind, run identifier, epoch, sender identifier, timestamp, nonce, and signature. The message schemas below specify the additional payload fields; repeated envelope fields are shown only where useful. Fields are required unless marked optional. Non-voting resources communicate through authenticated adapters or runtime proxies whose authority is recorded in the deployment manifest.

Hashes use $\mathrm{SHA256}$ over a deterministic serialization. The serialization specifies key ordering, string encoding, numeric representation, and ordering for fields interpreted as sets. A signature covers the envelope and payload, excluding the signature field itself.

The proposal hash identifies the substantive decision object, including its referenced dependencies but excluding transport metadata such as timestamps, nonces, and signatures. A decision identifier additionally distinguishes the run, epoch, decision kind, and decision attempt. Consequently, retrying an unchanged proposal does not conceal a stale revision, while a fresh authorized vote on the same candidate receives a distinct decision identifier.

A1.1 System Model and Assumptions

URCP distinguishes two participant classes. *Vote-capable models* inspect structured proposals and evidence and emit signed votes with justifications. *Executable resources* perform assigned operations, including code execution, retrieval, image generation, audio synthesis, or embodied actions. Executable resources need not participate in governance, even when their interfaces can produce structured text.

Vote-capable models register with `vote_capable=true`. Non-voting resources register through manifest-authorized proxies with `vote_capable=false`. Both classes use capability, modality, and interface declarations for eligibility checks. Registration does not by itself establish that a participant will perform every declared capability correctly.

Assumption A1.1 (Authenticated Communication)**.** *Every accepted protocol message is authenticated to an admitted sender or its authorized proxy. Participant identities and public keys are checked against the deployment's admission policy. No participant can forge a message attributed to another admitted participant. Runtime-originated messages and manifest-proxy records are authenticated using the runtime's trusted identity.*

Assumption A1.2 (Bounded-Time Delivery)**.** *Messages exchanged among active participants and the runtime are delivered within a known finite bound τ_{send}. Failed or unavailable participants may produce no message; their absence is handled by the specified timeout rules.*

Assumption A1.3 (Synchronous Protocol Phases)**.** *Each protocol phase has an announced start, finite timeout, and deterministic timeout transition. The runtime provides the authoritative deadline clock. Active voters share the phase view up to bounded communication delay and clock skew.*

Assumption A1.4 (Fixed Voting Set Per Epoch)**.** *After discovery closes, $\mathcal{M}^*$ remains fixed for the epoch. A missing or divergent response does not silently remove a voter or reduce a decision threshold. For ordinary decisions, the electorate is $\mathcal{M}^*$. Assignment of a voter r uses $\mathcal{M}^* \setminus \{r\}$; assignment of a non-voting resource uses the full voter set. Membership changes require an authorized transition to discovery in a new epoch or a separate restart.*

Runtime and processing requirements. The mechanical runtime is trusted to enforce the published transition rules, maintain the ordered log, and tally votes correctly. It is not a semantic planner, but its availability is required for progress and timeout handling. The reference architecture therefore does not provide infrastructure-level decentralization or tolerance of an indefinitely unavailable runtime.

Bounded execution additionally requires finite message and proposal sizes, finite participant and role sets, bounded processing per operation, and an independently enforced deadline or attempt bound covering every stage. Mechanical rejection paths are included in these bounds even when they do not consume semantic revision rounds. These are requirements of the complete specification; they are not additional experimental findings.

A1.2 Runtime Shared State and Memory Model

The runtime maintains seven shared-state structures backed by an append-only event history. Registries and tables are materialized views of that history: updating or invalidating an entry appends an event rather than erasing its previous record. Thus an output lock can be invalidated and a role binding replaced without deleting the evidence that established the earlier state.

Every committed update is assigned a runtime sequence number, authenticated, and broadcast to the voters. Replicas apply updates idempotently in sequence order. Their views may temporarily differ because of bounded delivery delay; applying the same committed prefix produces the same state.

1. Capability Registry CR**.** The Capability Registry maps each admitted participant to its identity, authenticated key or proxy, protocol version, voting flag, registration source, executor family, modalities, capabilities, input and output types, tools, constraints, and admission evidence. Its entries are frozen after discovery for the current epoch.

The registry is the authoritative input to eligibility checks. A role claim may cite an admitted capability or executor family, but cannot introduce a new one. Changes to these declarations require discovery in a new epoch.

2. Specialization Registry SR**.** The Specialization Registry maps capability classes to compatible executor families and any associated modality requirements. Each row is marked HARD or SOFT. Hard rows constrain eligibility; soft rows provide advisory information and the explicitly defined ranking tiebreaker.

The SR is deployment policy, not a model-generated capability assessment. It does not bind roles to particular participants. Its contents are fixed within an epoch and included in context verification.

3. Vote Ledger VL**.** The Vote Ledger preserves authenticated ballots and their runtime receipt information:

$$\mathrm{VL}[(e, d)] = [m_1, m_2, \ldots].$$

Ballot revisions are appended rather than overwritten. For tallying, only the latest valid ballot from each eligible voter received before the decision deadline is effective. Late ballots remain available for audit but cannot change a closed decision.

4. Decision Registry DR**.** The Decision Registry records each decision's kind, substantive proposal hash, electorate, threshold, deadline, tally, verdict, accepted-artifact hash, and outcome certificate. Decision kinds include

$$\{\texttt{CONTEXT}, \texttt{FRAME}, \texttt{SUBTASK}, \texttt{COVERAGE}, \texttt{SCHEMA},\\ \texttt{ASSIGNMENT}, \texttt{OUTPUT}, \texttt{FINAL}, \texttt{REPAIR}\}.$$

An open decision and its terminal outcome are separate events. A terminal outcome is immutable.

For an electorate $V_e(d)$, the recorded denominator and threshold are

$$n_e(d) = |V_e(d)|, \qquad t_e(d) = \lceil q\, n_e(d) \rceil.$$

An empty electorate cannot authorize a decision.

5. Role Ranking Table RRT**.** For each role, the Role Ranking Table stores eligible claimants and their assignment history:

$$\mathrm{RRT}[s_j] = \big[(r_k, \mathrm{tier}_k, Y_k, N_k, R_k, A_k, t_k)\big]_k.$$

The three tiers are BOUND, UNVOTED, and VOTED-FAILED. Tallies are undefined for candidates who have not been voted on.

The ordering in §A1.11 determines which fallback candidate is presented next. Ordering never authorizes execution: each replacement binding requires a valid assignment decision. A repair attempt also records which candidates have already been tried, preventing failed candidates from being repeatedly reinserted into the same attempt.

6. Output Lock Table OLT**.** The Output Lock Table records accepted artifacts and their dependencies:

$$\mathrm{OLT}[s_j] = \big(\mathrm{outhash}_j, \mathrm{producer}_j, \mathrm{decision}_j, \mathrm{upstream}_j, \mathrm{contracthash}_j\big).$$

Here $\mathrm{upstream}_j$ identifies the exact upstream artifacts consumed, and $\mathrm{contracthash}_j$ identifies the role specification and acceptance criterion under which the output was ratified.

An active lock is usable only while its recorded dependencies and contract remain valid. Repair appends invalidation or carry-forward events as specified in §A1.14. A hash establishes artifact identity; it does not establish semantic correctness or reconstruct missing artifact bytes.

7. Audit Log AL**.** The Audit Log contains the runtime-ordered history of admitted messages, decision openings, ballots, outcomes, state updates, timeouts, epoch transitions, and terminal verdicts. Rejected or late messages relevant to protocol decisions are recorded with their disposition, subject to the deployment's bounded logging policy.

The log supports deterministic reconstruction of committed coordination state. Replaying it does not reproduce model inference or external tool effects unless the corresponding artifacts and execution records are also retained.

Read/write semantics. Voters read replicated state and submit typed messages. The runtime validates each message's identity, epoch, phase, authorization, schema, and dependencies before applying its permitted state transition. Only runtime-originated outcome certificates can close decisions, and only accepted decisions can establish semantic locks.

Historical records remain immutable. The current state is obtained by replaying updates, invalidations, and carry-forward events. Preserved artifacts retain references to their original authorization rather than being represented as newly voted outputs.

Storage and recovery. A single-writer append-only log with indexed materialized views is sufficient for normal operation. Crash recovery additionally requires durable, consistently ordered commits, retained artifact bytes, and persisted deadlines and budget state. Signed records alone do not guarantee durable recovery. External state-changing operations require application-specific idempotency or reconciliation before replay.

A1.3 Stage 1: Discovery and Capability Declaration

Discovery admits participants, checks their declarations, and constructs the Capability Registry. Capability checks combine schema validation, authenticated identity, controlled-vocabulary membership, and any probes required by the deployment. Vocabulary membership verifies the form of a claim, not its general semantic accuracy.

Controlled vocabularies. Each protocol version specifies controlled vocabularies for capabilities, modalities, interface types, tools, and executor families. Examples include `reasoning`, `coding`, `code_execution`, `ocr`, `retrieval`, `tool_use`, `image_generation`, `audio_synthesis`, `video_generation`, and `embodied_action`. Executor families include `llm`, `vlm`, `retriever`, `code_executor`, `code_llm`, and modality-specific generators.

Tool identifiers are drawn from the deployment resource pool. Task-specific interface schemas must be registered through the deployment's declared type system before being referenced. Unknown

Algorithm 1 Runtime admission and application of a protocol message.

```
1:  procedure APPLY(m)
2:      if run deadline or current stage bound has expired then
3:          apply the corresponding terminal transition
4:          return
5:      end if
6:      if m is oversized, malformed, or belongs to another run then
7:          reject m without changing coordination state
8:          return
9:      end if
10:     if m.epoch ≠ e then
11:         record disposition as stale or unexpected epoch
12:         return
13:     end if
14:     if m.kind = IDENTITY then
15:         process m using Algorithm 2
16:         return
17:     end if
18:     authenticate m using the admitted participant or proxy key
19:     if authentication, replay, phase, or sender authorization fails then
20:         record rejection and return
21:     end if
22:     append m and its receipt sequence to AL
23:     if m.kind = VOTE then
24:         append ballot and on-time/late status to VL
25:     else if m.kind = ROLE-CLAIM then
26:         record the claim for the named role and claim window
27:     else if m.kind = SUBTASK-COMPLETE then
28:         stage the output for checking; do not create a lock
29:     else if m.kind = HANDOFF-RECEIVED then
30:         check the receiver, edge, and active upstream lock
31:     else if m.kind = REOPEN then
32:         queue the repair proposal under the bounded repair procedure
33:     else if m is an admissible proposal then
34:         queue the proposal under the current stage's ordering rule
35:     else if m.kind = VOTE-OUTCOME then
36:         require runtime origin and a matching closed decision
37:     end if
38:     append and broadcast any resulting committed state updates
39: end procedure
```

identifiers are rejected as VOCAB-VIOLATION. Related capability names are not silently treated as synonyms; any mapping is part of the published vocabulary.

Message: IDENTITY	
`public_key`	Participant key, or the trusted key of its manifest-authorized proxy.
`protocol_version`	Protocol and vocabulary version.
`vote_capable`	Whether the participant is authorized to vote.
`registration_source`	`self` or `manifest`.
`executor_family`	Family admitted for subsequent SR checks.
`modalities`	Declared supported modalities.
`capabilities`	Declared capabilities from the versioned vocabulary.
`input_types`	Supported input interfaces.
`output_types`	Supported output interfaces.
`tool_interfaces`	Available deployment-authorized tools.
`constraints`	Optional cost, latency, privacy, safety, or resource constraints.
`capability_proofs`	Probe responses or evidence references; required for capabilities subject to mandatory probes.

Algorithm 2 IDENTITY verification and admission.

procedure VERIFYIDENTITY(m)
  if discovery is closed **then**
    reject as late and **return**
  end if
  validate envelope, identity schema, and protocol version
  check identity/key binding against the admission policy
  verify the participant or authorized proxy signature
  check vocabularies, timestamp window, and replay identifier
  if any required check fails **then**
    record the rejection reason and **return**
  end if
  $\text{caps} \leftarrow m.\text{capabilities}$
  for each required probe capability $c \in \text{caps}$ **do**
    issue or validate a challenge bound to this participant and discovery attempt
    if the response is absent, late, or fails the declared criterion **then**
      remove c from caps
      record `PROBE-FAIL` for c
    end if
  end for
  admit the record with the retained capability set
  append and broadcast the corresponding CR update
end procedure

Probe scope. The deployment declares which capabilities require probes and the associated acceptance criteria. All probe exchanges are bounded by the discovery deadline. A passing probe supports the tested capability under that criterion; it does not prove unrestricted competence. Capabilities outside the probe set remain declarations.

Set construction. At the close of discovery,

$$\mathcal{M}^* = \{p \in \text{CR} : \text{CR}[p].\text{vc} = \top\}, \qquad \mathcal{R}^* = \{p \in \text{CR} : \text{CR}[p].\text{caps} \neq \emptyset\}.$$

A vote-capable model with executable capabilities belongs to both sets. A non-voting resource belongs only to $\mathcal{R}^*$.

Edge cases. An exact replay is discarded without removing an already admitted participant. Replay checks use the run, epoch, sender, and nonce; a nonce collision across distinct senders does not by itself justify excluding both. Conflicting registrations for the same identity are rejected or resolved according to the fixed admission policy. An unregistered manifest resource is unavailable.

An empty voter set terminates as NO-VOTERS; an empty executable-resource set terminates as NO-RESOURCES. With one voter, ordinary decisions remain possible, but assigning that voter to a role would leave an empty assignment electorate. Such an assignment cannot be accepted.

Bounded discovery and complexity. Discovery closes at τ_{disc}; late identity and probe messages cannot extend it. If P participants each undergo at most C probes, discovery uses $O(P(1 + C))$ logical messages, excluding broadcast fan-out. Delivering each broadcast separately to all voters adds the corresponding delivery cost.

A1.4 Specialization Registry Publication

After discovery, the runtime publishes the deployment's Specialization Registry. The registry provides a common mapping from capability classes to executor families. It standardizes eligibility policy while leaving task decomposition and actual assignments to the voters.

`Message: SR-PUBLISH`

`sr_version`	Registry schema and policy version.
`rows[]`	Capability class, allowed executor families, required modalities, HARD/SOFT status, and optional notes for each row.
`registry_hash`	Hash of the canonical registry.

Registry content. Representative mappings are `image_generation` to `diffusion`, `audio_synthesis` to `audio_generator`, `video_generation` to `video_generator`, `embodied_action` to `vla_controller`, `code_execution` to `code_executor` or `code_llm`, `ocr` to `ocr_engine` or `vlm`, `retrieval` to `retriever`, and `reasoning` to `llm` or `vlm`. A `tool_use` row may admit several executor families. The deployment specifies the exact rows and their hardness.

Use sites. Subtask and schema proposals cite relevant SR rows. Hard rows constrain the admitted families and modalities of an eligible executor. Soft rows do not exclude otherwise eligible participants; they inform proposals, votes, and the ranking tiebreaker in §A1.11. A declared SR capability with no compatible deployed resource does not create a new executor.

Mutation policy. The SR is immutable within an epoch. Changing registry policy or admitting new participants requires an accepted L_5 repair that explicitly returns to discovery, or a separate restart. An ordinary repair that preserves discovery state also preserves the SR.

A1.5 Stage 1b: Context Verification

`Message: CONTEXT-VERIFY`

`task_hash`	Hash of the user task T.
`guidance_hash`	Hash of optional guidance G.
`resource_hash`	Hash of the canonical resource pool, including SR policy.
`voting_set_hash`	Hash of the sorted discovered voter identifiers.
`resource_set_hash`	Hash of the sorted executable-resource identifiers.

Canonical context. The runtime announces the canonical context tuple after discovery and SR publication. Canonicalization sorts participants, tools, and registry rows according to the declared serialization rules. All voters receive the same announced tuple and independently compare it with their local inputs.

Verification and quorum. A matching authenticated `CONTEXT-VERIFY` response is an affirmative ballot on the announced tuple. A divergent response is non-affirmative and is recorded as `CONTEXT-MISMATCH`; a missing response becomes ABSTAIN at τ_{verify}.

The electorate is the discovered $\mathcal{M}^*$, and acceptance requires $\lceil q|\mathcal{M}^*| \rceil$ matching responses. Divergent or non-responding voters are not silently removed from the denominator. Once context is accepted,

subsequent proposals must reference the accepted context and dependencies. A participant that cannot adopt that context cannot contribute valid execution messages, but remains in the fixed electorate.

Outcomes. Acceptance locks the epoch context. If the required affirmative threshold is not reached by the verification deadline, the run terminates as CONTEXT-INCONSISTENT. Late responses do not change the outcome. For a repair that preserves discovery state, the accepted epoch-transition record carries that context forward; a return to discovery repeats context verification.

A1.6 Vote Message Format and Tally Broadcast

All semantic decisions use the same supermajority rule. For decision d, with claimant r when applicable, define

$$V_e(d) = \begin{cases} \mathcal{M}^* \setminus \{r\}, & d \text{ assigns a role to } r \in \mathcal{M}^*, \\ \mathcal{M}^*, & \text{otherwise.} \end{cases}$$

Thus assignment to a non-voting resource retains the full electorate. Let

$$n_e(d) = |V_e(d)|, \qquad t_e(d) = \lceil q n_e(d) \rceil.$$

For $n_e(d) > 0$, the decision rule is

$$\Phi_q(d) = \begin{cases} \text{ACCEPT}, & Y(d) \geq t_e(d), \\ \text{REJECT}, & N(d) \geq t_e(d), \\ \text{REVISE}, & \text{otherwise.} \end{cases}$$

Because $q > 1/2$, acceptance and rejection cannot both meet their thresholds. REVISE and ABSTAIN are distinct ballot values, but neither contributes to $Y(d)$ or $N(d)$.

An empty electorate produces the typed failure `NO-ELIGIBLE-VOTERS`; it never produces zero-vote acceptance. The enclosing stage must follow its bounded failure or repair transition.

Message: VOTE

`decision_id`	Identifier of the open decision attempt.
`decision_kind`	One of the decision kinds in §A1.2.
`proposal_hash`	Hash of the substantive proposal being evaluated.
`vote_value`	YES, NO, REVISE, or ABSTAIN.
`justification`	A list of tagged explanations with optional evidence references. Tags include `capability`, `coverage`, `coherence`, `evidence`, `cost`, `safety`, and `guidance`.

Ballot validity and storage. A ballot is effective only if its sender belongs to $V_e(d)$, its proposal and decision identifiers match the open decision, and it arrives within the announced voting window. All valid ballot revisions are appended to VL. The last eligible ballot by runtime receipt sequence before the deadline is effective for that voter. An exact replay does not constitute a ballot revision.

Algorithm 3 Supermajority tally and outcome broadcast.

procedure TALLY(e, d)
  $V \leftarrow V_e(d)$ from the decision-opening record
  if $V = \emptyset$ **then**
    record and broadcast `NO-ELIGIBLE-VOTERS`
    return
  end if
  $t \leftarrow \lceil q|V| \rceil$
  for each $v \in V$ **do**
    select v's last valid on-time ballot
    if no such ballot exists **then**
      record an effective ABSTAIN
    end if
  end for
  count effective ballots as Y, N, R, A
  if $Y \geq t$ **then**
    verdict $\leftarrow$ ACCEPT
  else if $N \geq t$ **then**
    verdict $\leftarrow$ REJECT
  else
    verdict $\leftarrow$ REVISE
  end if
  append the immutable outcome certificate to DR
  if verdict $=$ ACCEPT **then**
    commit the decision-kind-specific accepted artifact
  end if
  broadcast the signed `VOTE-OUTCOME`
end procedure

`Message: VOTE-OUTCOME`

`decision_id`	Identifier of the closed decision attempt.
`decision_kind`	Kind of decision.
`proposal_hash`	Hash of the evaluated proposal.
`electorate_hash`	Hash of the authorized voter set for this decision.
`denominator`	$n_e(d)$; claimant exclusion applies only to voter claimants.
`threshold`	$t_e(d) = \lceil q n_e(d) \rceil$.
`yes_count`, `no_count`	Effective affirmative and negative counts.
`revise_count`, `abstain_count`	Effective revision and abstention counts.
`verdict`	ACCEPT, REJECT, or REVISE; a typed failure is recorded separately when tallying is impossible.
`locked_artifact_hash`	Hash of the accepted artifact, or `NIL`. For an output decision this identifies the output artifact, not merely the transport envelope.
`justification_index`	References to effective ballots and their justification hashes; missing ballots are explicitly marked.

Outcome authority and visibility. The runtime broadcasts the outcome to all voters. Subsequent transitions require the corresponding certificate and accepted artifact. Voters can audit the certificate against the recorded electorate and ballots. This establishes a reproducible decision record, not agreement that the accepted content is factually correct.

The replicated ledger may expose ballots before tallying. The core protocol therefore does not assume independent or secret voting.

Revisions and late ballots. A REVISE outcome returns the recorded dissent to the proposers. A revised proposal opens a new decision attempt. Late ballots are marked as late and cannot modify a terminal outcome, even if they arrive before the next stage begins.

A1.7 Stage 2: Task Framing

Message: FRAME-PROPOSE

`task_interpretation`	Proposed interpretation of T.
`guidance_policy`	Treatment of guidance as mandatory constraints or preferences.
`required_modalities`	Modalities required by the task.
`decomposition_need`	NONE, MINIMAL, or FULL.
`resource_policy`	Required tools, memory scopes, and specialist resources.
`justification`	Reasons supporting the proposed interpretation and policy.

Procedure. Any voter may propose a frame. Admissible proposals are considered in timestamp order, with lexicographic sender identifier and proposal hash as deterministic tiebreakers. Timestamps must satisfy the deployment's clock validation policy.

Acceptance locks F. Rejection discards that proposal. Revision returns the dissent to the original proposer or another volunteering voter. Pending alternatives and revisions are handled under the same bounded proposal procedure.

Guidance handling. Explicit user-mandated constraints cannot be downgraded by a vote. Where guidance is ambiguous, its interpretation is part of the framing decision. The runtime rejects mechanically checkable violations of explicit constraints. Semantic compliance is evaluated by voters; the runtime does not independently interpret natural-language intent.

Failure handling. If no proposal arrives within τ_{propose}, the lexicographically smallest voter identifier is designated as fallback proposer. Failure within τ_{fallback} terminates as NO-CONSENSUS.

An unchanged substantive proposal is a stale revision regardless of a new timestamp or nonce. Two consecutive stale revisions may terminate the stage early. At most K_1 semantic decision attempts are permitted for framing. Mechanical failures remain subject to the independent stage bound in §A1.15.

A1.8 Stage 3: Subtask Formation

Message: SUBTASK-PROPOSE

`frame_hash`	Hash of the accepted task frame.
`subtasks[]`	Candidate subtasks, each with name, description, input and output types, required capabilities, tool requirements, local completion criterion, and cited SR rows.
`coverage_argument`	Explanation of how the proposed set covers the accepted frame.

Canonicalization. The runtime consolidates candidates only when their canonical substantive fields match. Whitespace normalization or other normalization is applied only where the declared field semantics permit it. Different capability requirements, interfaces, or completion criteria remain distinct. The runtime does not merge candidates based on inferred semantic similarity.

Mechanical consistency checks. The proposal must reference the current frame and contain a coverage argument at the proposal level. For each hard capability requirement, the runtime checks the SR and admitted resource declarations for compatible execution support. An unsupported requirement is rejected as `SR-INCONSISTENT-PROPOSAL`. These checks establish compatibility with declarations, not that the proposed decomposition is semantically sufficient.

Two-tier voting. Each admissible candidate receives a Φ_q vote. Accepted candidates enter a provisional set. Rejected candidates are discarded; revised candidates are evaluated as new substantive objects.

The complete provisional set then receives a coverage vote. Acceptance locks $\mathcal{B}$. Otherwise, voters may propose additions, removals, merges, or interface changes within the remaining budget. An unchanged accepted candidate need not be voted on again, but every modified candidate and every changed complete set require the appropriate ratification.

Failure handling. Coverage is attempted at most K_3 times. Each candidate list and proposal window is finite, and candidate revision paths are covered by the stage deadline. Repeated rejected candidates cannot reset that deadline. Missing proposals invoke the bounded fallback procedure. Exhaustion without accepted coverage terminates as NO-CONSENSUS.

A1.9 Stage 4: Role Schema Formation

Each role is represented by

$$s_j = (\mathrm{name}_j, \mathrm{covers}_j, \mathrm{caps}_j, \mathrm{in}_j, \mathrm{out}_j, \mathrm{tools}_j, \mathrm{mem}_j, \tau_{\mathrm{exec},j}, \mathrm{accept_crit}_j),$$

where $\mathrm{covers}_j \subseteq \mathcal{B}$. The remaining fields specify capability requirements, interfaces, tools, memory scope, execution timeout, and completion criterion.

Message: ROLE-SCHEMA-PROPOSE

`subtask_set_hash`	Hash of the accepted subtask set.
`roles[]`	Complete role specifications.
`dag_edges[]`	Directed dependencies forming $\mathcal{O}$, with sufficient interface information to determine producer–consumer bindings.
`sr_alignment`	SR rows supporting the capability requirements of each role.
`justification`	Explanation of coverage and workflow coherence.

Atomic ratification. The role schema and execution DAG are voted on as a complete object. Acceptance locks $\mathcal{S}$ and $\mathcal{O}$. Roles are not independently accepted when their combination has not been checked for interface compatibility and coverage.

Mechanical pre-checks. Before voting, the runtime checks:

1. **Coverage:** every accepted subtask is covered by at least one declared role.
2. **Acyclicity:** the proposed dependency graph is a DAG.
3. **Interface compatibility:** each dependency supplies the required consumer input through a declared compatible output. For roles with multiple inputs, the combined incoming bindings and permitted external inputs must cover the required interface.
4. **Reachability:** every role is reachable from task inputs and contributes to a declared final output.

A generic set-inclusion comparison between all producer outputs and all consumer inputs is insufficient for multi-input roles; the check uses the declared bindings. Mechanically invalid proposals do not consume a semantic K_4 attempt, but do consume the independent stage bound.

Failure handling. After K_4 unsuccessful semantic attempts, the stage terminates as NO-CONSENSUS. Three consecutive failures of the same pre-check trigger a `STRUCTURAL-DEAD-END` proposal at L_4. Alternating pre-check failures remain bounded by the stage deadline. A repair is executed only after its own acceptance.

A1.10 Stage 5: Model-Agent Assignment

Message: ROLE-CLAIM

`role_id`	Role being claimed.
`capability_evidence`	References to admitted capabilities satisfying the role.
`modality_fit`	Compatibility with the role's input and output interfaces.
`tool_fit`	Availability of the required tools.
`sr_family_self_decl`	Executor-family declaration; must match the admitted CR record.
`cost_latency`	Optional execution-cost and latency estimate.
`claim_justification`	Reasons supporting the proposed assignment.

Eligibility. The runtime checks each claim against the admitted participant record and the accepted role contract. A non-voting resource may submit a claim through its authorized adapter or proxy. Such a claim is not a ballot.

Algorithm 4 Eligibility filter for role s_j and claimant r.

procedure ELIGIBLE(s_j, r)
  if $r \notin \mathcal{R}^*$ **then**
    return `ELIGIBILITY-FAIL/RESOURCE`
  end if
  $\rho \leftarrow \mathrm{CR}[r]$
  if $\mathrm{caps}_j \not\subseteq \rho.\mathrm{caps}$ **then**
    return `ELIGIBILITY-FAIL/CAPS`
  end if
  if $\mathrm{in}_j \not\subseteq \rho.\mathrm{in}$ **then**
    return `ELIGIBILITY-FAIL/IN`
  end if
  if $\mathrm{out}_j \not\subseteq \rho.\mathrm{out}$ **then**
    return `ELIGIBILITY-FAIL/OUT`
  end if
  if $\mathrm{tools}_j \not\subseteq \rho.\mathrm{tools}$ **then**
    return `ELIGIBILITY-FAIL/TOOLS`
  end if
  if required modalities or mechanically enforced scopes fail **then**
    return `ELIGIBILITY-FAIL/SCOPE`
  end if
  if the claim's executor family differs from $\rho.\mathrm{family}$ **then**
    return `ELIGIBILITY-FAIL/DECLARATION`
  end if
  for each hard SR row required by caps_j **do**
    if $\rho.\mathrm{family}$ is not admitted by that row **then**
      return `ELIGIBILITY-FAIL/SR`
    end if
  end for
  return `ELIGIBLE`
end procedure

The input/output inclusion tests refer to the registered interface identifiers supported by the executor. Compatibility among workflow edges is checked separately during schema formation.

Per-claimant voting. Claims are collected within τ_{claim}. Eligible claimants are processed in claim timestamp order with deterministic identifier tiebreaking. For claimant r, the assignment electorate is

$$V_e(d_{j,r}) = \mathcal{M}^* \setminus \{r\}.$$

This expression removes a voter only when $r \in \mathcal{M}^*$. Accordingly, the threshold is

$$t_e(d_{j,r}) = \begin{cases} \lceil q(|\mathcal{M}^*| - 1) \rceil, & r \in \mathcal{M}^*, \\ \lceil q|\mathcal{M}^*| \rceil, & r \notin \mathcal{M}^*. \end{cases}$$

An empty electorate produces `NO-ELIGIBLE-VOTERS`, not acceptance.

The first candidate receiving ACCEPT is bound to the role. Remaining candidates are not voted on, but their claims are retained as unvoted fallbacks. Candidates receiving REJECT or REVISE are recorded as voted-failed before the next candidate is considered.

Failure handling. A role permits at most K_5 candidate-vote attempts in an assignment invocation. If there are no eligible candidates, all candidates fail, or the assignment budget is exhausted, the runtime proposes L_3 repair. The schema changes only if that repair is accepted. If repair cannot be authorized within its bounds, the run terminates as NO-CONSENSUS.

A1.11 Role Ranking Table Construction

The Role Ranking Table preserves eligible alternatives from the initial claim window so that assignment repair can present fallbacks without reopening unrestricted claim solicitation.

Three-tier ordering. For a successfully assigned role s_j, its eligible claimants are partitioned into:

- **Tier A: bound.** The unique claimant with the active accepted binding.
- **Tier B: unvoted.** Claimants not evaluated before the first accepted binding. They are ordered by claim timestamp, then the soft-SR tiebreaker, then lexicographic identifier.
- **Tier C: voted-failed.** Claimants evaluated without acceptance. They are ordered by descending recorded YES count, ascending NO count, ascending claim timestamp, and lexicographic identifier.

The full ordering is

$$\mathrm{RRT}[s_j] = [\text{Tier A}; \text{Tier B}; \text{Tier C}].$$

Raw tallies are used only as presentation evidence; they are not a new authorization and need not be directly comparable across different assignment electorates.

Soft-SR tiebreaking. Among Tier-B candidates tied on timestamp, the runtime prefers a candidate whose admitted family matches a relevant soft SR row over one with no such match. Remaining ties are resolved by identifier. If all candidates match, the rule provides no additional preference. Hard SR requirements have already been enforced by eligibility. The tiebreaker does not bind a candidate or imply measured performance superiority.

Update policy. Initial assignment constructs the table. L_1 repair preserves the binding and ranking. Accepted L_2 repair invalidates the failed binding and presents fallback candidates in the recorded order. Each candidate is tried at most once within that repair invocation. Failed attempts remain in the audit history but are not requeued for immediate reconsideration. A changed schema invalidates the ranking for changed roles.

Worked example. Suppose four eligible voter claimants r_a, r_b, r_c, r_d arrive in that order for role s_3, with $|\mathcal{M}^*| = 8$ and $q = 3/4$. Each assignment electorate contains seven voters and requires six YES votes.

Candidate r_a receives $(Y, N, R, A) = (4, 2, 0, 1)$ and is not accepted. Candidate r_b receives $(6, 1, 0, 0)$ and is bound. Candidates r_c and r_d are not voted on. The resulting ranking is

$$\mathrm{RRT}[s_3] = [\, r_b \mid r_c, r_d \mid r_a \,].$$

After an accepted L_2 repair, the failed bound candidate r_b is removed from the fallback queue exactly once. The runtime presents r_c, then r_d, then r_a if needed, subject to the assignment and stage bounds. Every replacement requires a fresh assignment vote.

A1.12 Stage 6: Execution and Checkpoint Validation

Message: SUBTASK-COMPLETE	
`role_id`	Producing role with an active accepted assignment.
`subtask_id`	Completed subtask or the corresponding role-output identifier.
`output`	Produced artifact or its authenticated retained-artifact reference.
`evidence`	Required citations, tool records, execution traces, or other support.
`output_type`	Declared interface type.
`upstream_outhashes`	Exact hashes of consumed upstream artifacts.
`completion_claim`	Explanation of compliance with the accepted completion criterion.

Message: HANDOFF-RECEIVED	
`source_role`	Upstream producing role.
`target_role`	Receiving role.
`output_hash`	Hash of the received upstream artifact.

Execution and locking. Roles execute according to the accepted DAG. A role becomes eligible to execute only when its required upstream outputs are actively locked and their handoffs have been acknowledged. Independent ready roles may execute concurrently if the deployment supports parallel execution.

The assigned executor produces `SUBTASK-COMPLETE`. After mechanical checks, the full electorate $\mathcal{M}^*$ evaluates the output against accept_crit_j. The default checkpoint electorate includes the producer if it is a voter; self-vote exclusion applies to assignment decisions only.

Acceptance creates an output lock binding the artifact, producer, role contract, decision certificate, and upstream hashes. Revision returns suggestions to the same executor. Rejection initiates a repair proposal identifying the failure and affected scope.

Mechanical checks. Before opening a checkpoint vote, the runtime checks that:

- the sender is the assigned executor or its authorized proxy;
- the output satisfies the declared syntactic interface;
- mechanically required evidence fields are present;
- every cited upstream artifact matches an active lock; and
- the referenced role contract remains current.

These checks do not establish factual correctness. Invalid submissions are rejected without consuming a semantic checkpoint attempt, but remain subject to the independent execution-stage bound.

Handoff validation. For edge (s_i, s_j), the assigned consumer acknowledges the exact locked artifact of s_i. A mismatched hash produces `STALE-HANDOFF`. The consumer may correct the acknowledgment within the original handoff window; a correction does not reset the deadline. Failure to acknowledge the valid artifact within that window triggers an L_2 repair proposal for the consumer role.

Failure handling. Execution timeout proposes L_1 repair. Exhausting K_6 semantic checkpoint attempts proposes L_2 repair. A malformed-output stream cannot bypass the independent stage deadline. A repair proposal does not itself authorize a retry in a new epoch or a replacement executor.

Non-voting resources submit outputs through their authenticated adapters or proxies and never cast checkpoint ballots. Tool side effects are governed by the deployment's execution policy; output validation alone does not reverse an external state change.

A1.13 Stage 7: Final Completion

Message: FINAL-SUBMIT

`result`	Assembled task output Y.
`provenance`	Ordered references to contributing roles, active output hashes, and their acceptance certificates.
`component_hashes`	Hashes of the components required by the accepted final interface.
`global_completion_claim`	Explanation of compliance with the task frame and global validation conditions.

Procedure. The assigned final-stage executor assembles Y from valid locked components. The runtime checks provenance and interface consistency, then opens a final vote over $\mathcal{M}^*$.

Acceptance terminates as COMPLETE and returns Y. A revision confined to final assembly may be attempted within the remaining K_7 budget. If revision requires changing an upstream locked output, it must follow an accepted repair transition rather than modifying that output in place.

Assembly failure. Inconsistent components or an inadequate assembly interface trigger a repair proposal at the appropriate scope. A structural assembly failure proposes L_3. No schema or upstream artifact is changed until the corresponding repair is accepted. If final completion or repair exhausts its bounds, the run terminates as NO-CONSENSUS.

Terminal record. Every terminal outcome records the final epoch, termination reason, relevant decisions, and available audit history. A COMPLETE outcome identifies the accepted result. A NO-CONSENSUS outcome may include available partial artifacts, explicitly marked as not constituting a ratified final answer. No semantic ranking of partial artifacts is performed by the mechanical runtime.

A1.14 Repair: Mechanics and Per-Level Model Selection

Message: REOPEN

`repair_level`	L_1 output, L_2 assignment, L_3 schema, L_4 subtasks, or L_5 frame.
`failure_type`	Typed cause such as `TIMEOUT`, `VALIDATION-FAIL`, `NO-SUITABLE-MODEL`, `TYPE-MISMATCH`, `HANDOFF-FAIL`, `ASSEMBLY-FAILURE`, `STRUCTURAL-DEAD-END`, or `STATE-CONTRADICTION`.
`affected_objects`	Directly affected roles, assignments, outputs, subtasks, or frame.
`justification`	Evidence supporting the requested scope.
`rediscovery`	Whether an L_5 repair explicitly requires participant, resource, or registry re-evaluation; false at lower levels.

Table 6: Repair hierarchy and dependency-aware preservation. Every repair requires ratification before its state changes occur.

Level	Returns to	Reopened	Preserved
L_1	Stage 6	Affected outputs and dependent descendants	Frame, subtasks, schema, assignments, and independent valid outputs
L_2	Stage 5	Affected bindings, their outputs, and dependent descendants	Frame, subtasks, schema, unaffected bindings, and independent valid outputs
L_3	Stage 4	Role schema and DAG; incompatible bindings and outputs	Frame and subtasks; other artifacts only after explicit compatibility checks
L_4	Stage 3	Subtasks and all downstream coordination state	Frame and unchanged discovery state
L_5	Stage 2 or 1	Frame and downstream coordination state; discovery when requested	Audit history; discovery state only when rediscovery is not requested

Authorization and epoch transition. Any voter may propose a repair. The runtime may also propose a repair through a published mapping from mechanically detected failures to repair levels. The full current electorate $\mathcal{M}^*$ votes on the proposal. The runtime cannot execute an unaccepted repair.

On acceptance, the runtime checks the epoch limit and commits one transition from e to $e + 1$. That transition records the accepted repair, retained authorizations, invalidated state, resume stage, and whether discovery must be repeated. Repair helper procedures do not increment the epoch again.

Dependency-aware invalidation. For a set of directly affected roles U, let

$$\mathrm{cl}_{\mathcal{O}}(U) = U \cup \{s : s \text{ is a descendant of a role in } U \text{ in } \mathcal{O}\}.$$

For L_1 and L_2, outputs in this dependency closure are invalidated before execution resumes. Outputs outside the closure may be carried forward if their role contracts and dependencies are unchanged.

At L_3, an artifact is reusable only when the newly accepted schema preserves its producer binding, role contract, acceptance criterion, and required upstream artifacts, and permits that reuse. Otherwise it is invalidated. Reuse is checked in dependency order. At L_4 and L_5, downstream workflow locks are cleared as specified in Table 6.

A retained artifact is an explicitly carried-forward historical authorization. An old-epoch message cannot create a new lock or modify current state.

Assignment repair. For L_1, the accepted repair preserves the current binding; the same executor is retried. For L_2, the failed bound executor is removed once, and the remaining RRT candidates are presented in deterministic order. Each replacement receives a fresh assignment vote using the decision-specific electorate.

A failed candidate is removed from the current repair queue before the next is considered. Its failure remains in the audit history, but it is not reinserted into that queue. The repair invocation is bounded by the remaining candidates, K_5, and its enclosing stage deadline.

Escalation. Exhausting output revisions proposes L_2. Exhausting fallback assignments proposes L_3. Repeated structural schema failures may propose L_4. A framing defect or requested participant/resource re-evaluation may propose L_5. Every escalation requires a new repair vote; exhausting candidates does not authorize an automatic schema change.

Concurrent proposals and repair failure. Repair proposals are collected within τ_{repair} and considered in descending scope, with deterministic timestamp and identifier tiebreaking. If no proposal arrives, the runtime emits the predefined default proposal for the observed failure type. Each proposal receives one bounded decision attempt within the repair invocation. A non-accepted proposal does not change state. The process stops at the first accepted repair or terminates as NO-CONSENSUS when the bounded candidate set or repair deadline is exhausted.

Quorum loss and membership. Unavailable voters remain in the electorate and contribute abstentions. If repair cannot obtain the required affirmative votes, the runtime cannot shrink the electorate to make the repair pass. The run terminates under its bounded failure rule. Membership re-evaluation occurs only through an accepted rediscovery transition or a separate restart.

Stale messages and epoch cap. Messages from earlier epochs cannot mutate current coordination state. Their historical records remain available for audit. Preserved artifacts are accessed through explicit carry-forward records, not by accepting stale messages.

Every accepted repair consumes one epoch transition. If advancing would exceed $E_{\max}$, the run terminates as NO-CONSENSUS. The epoch cap bounds repeated repairs; it does not replace bounds on operations within an epoch.

A1.15 Timeouts and Deadlock Reference

Individual timeouts bound waiting operations. Semantic budgets bound proposal or output decisions. Neither alone bounds an arbitrary stream of mechanically invalid submissions. Therefore every stage

Algorithm 5 Candidate presentation after an accepted repair. The epoch transition and dependency invalidation occur before this procedure.

procedure PRESENTREPAIRCANDIDATE(ℓ, s_j)
  if $\ell = L_1$ **then**
    require a carried-forward valid binding for s_j
    return the bound executor
  else if $\ell = L_2$ **then**
    remove the failed bound entry exactly once
    $Q \leftarrow$ ordered remaining eligible RRT candidates
    $a \leftarrow 0$
    while $Q \neq \emptyset$ and $a < K_5$ and bounds remain **do**
      $r \leftarrow \text{pop_front}(Q)$
      $a \leftarrow a + 1$
      if ELIGIBLE(s_j, r) is not `ELIGIBLE` **then**
        record ineligibility
      else
        open a fresh assignment decision for (s_j, r)
        use electorate $\mathcal{M}^* \setminus \{r\}$
        tally under Algorithm 3
        if verdict is ACCEPT **then**
          commit r as the new bound executor
          record the remaining fallback state
          return r
        else
          record failure; do not reinsert r into Q
        end if
      end if
    end while
    propose L_3 repair for the unresolved role
    return `NIL`
  else if $\ell = L_3$ **then**
    resume schema formation
    rebuild affected bindings after schema acceptance
  else if $\ell = L_4$ **then**
    resume subtask formation
  else if $\ell = L_5$ **then**
    resume discovery if ratified; otherwise resume framing
  end if
end procedure

invocation, including repair, has an independent non-resetting deadline Δ_{stage} or an equivalent finite attempt cap covering all processed submissions.

A run also has a non-resetting wall-clock limit T_{run} and its declared resource budget. Stage deadlines do not reset after malformed messages, stale proposals, or repeated pre-check failures. An accepted repair creates a new bounded invocation in a new epoch, but does not reset the run limit.

These bounds are implementation requirements supporting the termination statement. Their inclusion does not assert an additional measured implementation or experimental result.

The existing notation uses K_1 for framing, K_3 for coverage, K_4 for schema decisions, K_5 for assignment attempts, K_6 for checkpoint decisions, and K_7 for final decisions. Each counter counts semantic attempts, including unsuccessful initial attempts, within its specified invocation. Mechanical rejections are bounded separately.

A1.16 Formal Properties

The properties below concern protocol authorization, declared compatibility, state provenance, and bounded execution. They are conditional on the assumptions in §A1.1 and the processing bounds in

Table 7: Timeouts and execution bounds. All waits are additionally limited by the enclosing stage and run deadlines.

Parameter	Scope	Expiry behavior
τ_{disc}	Discovery	Close admission and probes; construct the voter and resource sets.
τ_{verify}	Context	Tally matching responses with the fixed discovered electorate; terminate if context is not accepted.
τ_{propose}	Proposals	Invoke the designated fallback if no proposal arrives.
τ_{fallback}	Fallback	Terminate as NO-CONSENSUS if no admissible fallback arrives.
τ_{vote}	Decisions	Count missing ballots as abstentions and close the decision.
τ_{claim}	Assignment	Close the claim window and process received eligible claims.
$\tau_{\text{exec},j}$	Role execution	Propose L_1 repair for an execution timeout.
τ_{handoff}	Handoff	Propose L_2 repair for an unacknowledged or invalid handoff.
τ_{repair}	Repair proposals	Close collection; emit the predefined default if none arrived.
K_ℓ	Semantic attempts	Apply the stage-specific terminal or repair transition.
Δ_{stage}	Stage invocation	Stop all local retry paths, including mechanical rejections; terminate as NO-CONSENSUS.
T_{run}	Entire run	Terminate on the wall-clock limit; repair cannot reset it.
$E_{\max}$	Epochs	Terminate if an accepted repair would exceed the epoch limit.

Table 8: Representative failure conditions and bounded transitions. Any repair in the table requires a separate accepted repair vote.

Condition	Detection	Transition
No voters	Empty discovered electorate	Terminate NO-VOTERS.
No resources	Empty executable-resource set	Terminate NO-RESOURCES.
Invalid identity	Schema, identity, signature, or vocabulary failure	Reject admission within the discovery bound.
Probe failure	Missing or unsuccessful required probe	Remove the affected capability from the admitted record.
Context mismatch	Insufficient matching context responses	Terminate CONTEXT-INCONSISTENT; do not shrink the electorate.
Unsupported capability	No compatible admitted execution support	Reject the candidate; the stage deadline continues.
No proposer	Proposal window expires	Invoke fallback, then terminate if fallback fails.
Competing proposals	Multiple admissible proposals	Process in the published deterministic order.
Stale revision	Unchanged substantive proposal	Reject without resetting the stage deadline.
Missing ballot	Voting window expires	Count abstention and close the decision.
Polarized vote	Neither YES nor NO reaches $t_e(d)$	Return REVISE within the remaining stage budget.
Empty assignment electorate	$V_e(d) = \emptyset$	Record `NO-ELIGIBLE-VOTERS`; never accept with zero votes.
Voter crash	Missing messages or ballots	Retain the voter in the denominator; attempt authorized repair or terminate.
Coverage failure	Coverage vote not accepted	Revise the set within K_3 and the stage bound.
Invalid DAG or interfaces	Mechanical schema check fails	Reject; repeated structural failure may propose L_4.
No eligible executor	Empty eligible claim set	Propose L_3.
Assignment exhaustion	No accepted candidate within bounds	Propose L_3.
Execution timeout	$\tau_{\text{exec},j}$ expires	Propose L_1.
Checkpoint exhaustion	K_6 attempts without accepted output	Propose L_2.
Fallback exhaustion	No remaining accepted replacement	Propose L_3; do not change schema without authorization.
Invalid handoff	Wrong hash or missing acknowledgment	Permit correction within the original window; then propose L_2.
Changed dependency	Upstream lock invalidated	Invalidate dependent outputs before reuse or execution.
Concurrent failures	Multiple repair proposals	Consider broader scopes first under a bounded repair invocation.
Assembly failure	Inconsistent final assembly	Propose the appropriate repair, normally L_3 for structural failure.
Repair quorum failure	No repair proposal is accepted within bounds	Terminate NO-CONSENSUS; no automatic membership reduction.
Mechanical-rejection loop	Stage deadline or attempt cap exhausted	Terminate NO-CONSENSUS.
Run or epoch exhaustion	Global budget reached	Terminate with an explicit recorded reason.

§A1.15. They do not imply factual correctness, independent model judgments, or successful task completion.

Absence of a central semantic orchestrator. The runtime validates messages, orders proposals, filters declared eligibility, maintains state, enforces deadlines, and tallies votes. It may propose a

repair using a fixed failure-to-scope rule, but cannot execute that repair without ratification. Task frames, subtask sets, schemas, bindings, checkpoint outputs, and final results are accepted only through the specified electorate.

This is a separation of semantic authorization from mechanical execution. The deployment manifest, SR policy, and runtime remain externally supplied infrastructure.

Supermajority legitimacy. For every accepted decision d,

$$|V_e(d)| > 0 \quad \text{and} \quad Y(d) \geq \lceil q|V_e(d)| \rceil.$$

The electorate is fixed when the decision opens and follows the claimant-exclusion rule in §A1.6. Assignment to a non-voting resource does not remove a voter.

Justification. Only the tally transition can commit an accepted decision. It uses authenticated effective ballots from the recorded electorate, rejects empty electorates, and requires the explicit affirmative threshold. Missing ballots cannot contribute to acceptance. The same rule applies to repair authorization.

Declared-capability compatibility. Every committed role binding satisfies the mechanical eligibility predicate for the admitted participant record and the accepted role. This includes capability, interface, tool, modality, scope, and hard-SR compatibility.

Justification. Ineligible claims never open assignment votes. Every accepted initial or replacement binding references an eligible claim and an accepted assignment decision. This property concerns admitted declarations and tested probes; it does not guarantee execution quality.

Dependency-consistent output reuse. An active output lock references its accepted role contract and the exact upstream artifacts consumed. Repair invalidates affected outputs and their dependent descendants. A carried-forward output is reusable only when its authorization and dependencies remain compatible with the current accepted state.

Justification. Locks are created only after checkpoint acceptance. Mechanical handoff checks require active matching hashes. Repair applies dependency-aware invalidation before execution resumes, and schema changes require compatibility checks before reuse. Thus a historical artifact cannot be treated as current solely because its role identifier is unchanged.

Auditability. Given the retained authenticated event history and referenced artifacts, an auditor can reconstruct the committed coordination state, effective ballots, decision thresholds, assignments, output locks, and repair transitions.

Justification. Each transition records its input references and authorization in the ordered log. Deterministic replay of the same committed prefix reconstructs the same state. Missing messages are represented by timeout or abstention records rather than invented justifications. This property does not imply reproduction of stochastic inference, recovery of missing artifact bytes from hashes, or exactly-once external tool effects.

Finite termination. A conforming run terminates with an explicit verdict if the runtime continues to execute its transition and deadline handlers, processing steps are bounded, and the independent stage and run bounds are enforced.

Justification. Each proposal, voting, claim, execution, handoff, and repair window has a finite endpoint. Every stage also has a non-resetting deadline or equivalent finite attempt bound covering mechanical rejection paths. Thus an unbounded stream of malformed messages or alternating pre-check failures cannot keep a stage active indefinitely. Accepted repairs consume epochs, and at most $E_{\max}$ epochs are permitted. The non-resetting run deadline independently bounds the complete execution. Exhaustion produces an explicit failure verdict rather than assuming eventual semantic convergence.

Within-epoch deadlock freedom. While the runtime remains available and the stated timing and processing conditions hold, every waiting state has a bounded exit: an accepted decision, a revision within budget, a typed failure, an authorized repair, or a terminal verdict.

Table 9: Robustness on MABench (heterogeneous team, mean over 5 seeds). Recov. = repair success rate; NoCon. = NO-CONSENSUS rate; DFT = deadlock-free termination. *Takeaway:* recovery exceeds 86% up to twenty-percent agent failure and deadlock-free termination is exactly 100% across all eleven perturbations.

Perturbation	Task	Recov.	NoCon.	DFT	**Perturbation**	Task	Recov.	NoCon.	DFT
None	72.1	—	2.1%	100%	20% agent fail	64.3	86.4%	5.8%	100%
Drop 2 models	68.5	91.6%	3.7%	100%	30% agent fail	57.9	76.5%	9.2%	100%
Drop 4 models	63.2	85.2%	6.4%	100%	Tool failure	66.8	88.7%	4.6%	100%
Drop 8 models	51.7	72.8%	14.3%	100%	Stale memory	68.1	91.2%	3.9%	100%
10% agent fail	69.6	93.8%	3.1%	100%	Delayed votes	71.3	96.8%	2.4%	100%
					Missing ACKs	69.7	93.1%	3.3%	100%

Justification. The timeout and stage-bound transitions cover both absent messages and unsuccessful processing paths. A revision may return to an earlier step within the same stage, but cannot reset that stage's independent bound. A return to an earlier protocol stage requires an accepted repair and consumes an epoch transition. If a repair quorum is unavailable, bounded termination remains possible even though successful coordination is not.

Limits of the guarantees. A COMPLETE verdict certifies that the required voters accepted the output under the recorded criteria; it does not prove that the output is factually correct or safe for every application. A NO-CONSENSUS verdict records the absence of authorized completion and does not prove that all candidate answers are wrong. An indefinitely unavailable runtime or an execution outside the stated timing and processing conditions is outside the termination guarantee.

A2 Robustness Results

We report the full robustness study referenced from §4.4 in Table 9. Each row is the mean over five seeds on MABench with the heterogeneous twelve-voter team and the default URCP hyperparameters (Table 2). The pre- versus mid-epoch failure-timing conventions used here are formalized in Appendix A4.4, and a discussion of why external baselines cannot meaningfully run most of these perturbation conditions is given in Appendix A4.5. Two patterns are worth noting. Recovery exceeds 86% up to twenty-percent agent failure, which is the regime that the supermajority margin of Φ_q at $q{=}3/4$ is designed to absorb without invoking deeper repair levels. Deadlock-free termination is exactly 100% in every row, matching the formal guarantee of Appendix A1.16; the runs that genuinely cannot recover do not hang, but instead surface as elevated NO-CONSENSUS entries that downstream pipelines can act on rather than silently absorb.

A3 Emergent Coordination Phenomena

The aggregate task scores in §4.3 establish that URCP outperforms its baselines, but they do not explain *how* the protocol achieves these gains. Because every coordination decision in URCP is logged as a structured signed message—identity records, context-verification hashes, votes with justifications, role claims, checkpoints, handoff acknowledgments, repair proposals, and the runtime's `VOTE-OUTCOME` broadcasts (§A1.6)—each run produces a complete audit trace of how the participating models negotiated their division of labor. We analyzed these traces across approximately $5{,}400$ benchmark runs (3 benchmarks $\times$ 6 team configurations $\times$ 5 seeds $\times \sim$60 tasks per benchmark) to identify recurring patterns that emerge *without* being prescribed by the protocol or by an external orchestrator. This section reports eight such phenomena. The first three concern how roles are allocated across the heterogeneous voting set $\mathcal{M}^*$ and resource set $\mathcal{R}^*$ (§A3.1, §A3.2, §A3.3); the next two concern structural decisions the voters make about the coordination object itself (§A3.4, §A3.5); and the remaining patterns concern privacy-aware handoffs (§A3.6), multimodal coordination (§A3.7), and the protocol's behavior when genuine disagreement persists (§A3.8). Section A3.9 then situates URCP relative to two adjacent ecosystems—Google's A2A protocol and the AgentScope runtime—and clarifies how URCP composes with each. We close with a consolidated summary in Table 13.

Table 10: Capability-aware role concentration on the heterogeneous twelve-model voting set (mean over 5 seeds across MABench, τ-bench, and MemoryAgentBench). *Exec. share* is the tier's aggregate execution-role share e_i across runs (sums to 100% over the three voter tiers). Critical-path roles are those lying on the longest path of $\mathcal{O}$. Proposer-win rate is the fraction of schema proposals submitted by the tier that reached ACCEPT under Φ_q. Roles assigned to purely executable resources in $\mathcal{R}^*$ are excluded from the voter-tier shares and analyzed separately in §A3.3.

Model tier	Pool share	Exec. share	Crit.-path	Proposer-win	Final-stage
Frontier (4 models)	33.3%	54.6%	73.9%	61.4%	78.2%
Mid (6 models)	50.0%	38.0%	24.9%	32.7%	19.6%
Compact (2 models)	16.7%	7.4%	1.2%	5.9%	2.2%
χ^2 vs. uniform	—	214.7	487.3	156.8	612.4
p (Holm-corrected)	—	$<10^{-3}$	$<10^{-3}$	$<10^{-3}$	$<10^{-3}$

Trace analysis methodology. For each run we extracted four streams from the audit log AL (§A1.2): (i) the accepted role schema $\mathcal{S}$, the assignment map $\mathcal{A} : \mathcal{S} \to \mathcal{M}^* \cup \mathcal{R}^*$, and the execution DAG $\mathcal{O}$, all read directly from the relevant `VOTE-OUTCOME` entries in the Decision Registry DR; (ii) the per-decision vote record from the Vote Ledger VL, including verdicts in $\{\text{YES}, \text{NO}, \text{REVISE}, \text{ABSTAIN}\}$ together with structured justifications; (iii) every `SUBTASK-COMPLETE/HANDOFF-RECEIVED` pair with its locked output hash from the Output Lock Table OLT; and (iv) every `REOPEN` event with its level L_1–L_5 and triggering condition. We then computed four summary quantities per voter: the proposer-win rate p_i (fraction of proposals by ℓ_i that reached ACCEPT under Φ_q); the execution-role share e_i (fraction of all accepted roles assigned to ℓ_i, summing to 100% across voters per run); the execution participation rate π_i (fraction of runs in which ℓ_i receives at least one role, which may sum above 100% across voters because multiple voters typically execute per run); and the critical-path share c_i (fraction of roles on the longest path of $\mathcal{O}$ assigned to ℓ_i). All numbers are means over five seeds with 95% bootstrap confidence intervals; full per-task-family breakdowns and per-voter (p_i, e_i, π_i, c_i) tables are in Appendix A5.

A3.1 Capability-Aware Role Concentration

A consistent pattern across the heterogeneous twelve-model voting set is that critical-path roles—planners, verifiers, critics, and final assemblers—are disproportionately assigned to higher-capability reasoning models, while smaller models are concentrated in localized roles such as evidence formatting, single-tool execution, and text extraction. This pattern is not a property of the protocol: every assignment is proposed and ratified through the same Φ_q mechanism used for all other coordination decisions, with self-vote exclusion (§A1.10) ensuring claimants do not vote on their own roles. The pattern emerges from the proposing-and-voting dynamics of schema formation and assignment.

Table 10 reports the distribution. We bin the twelve voters into three tiers for descriptive analysis based on their declared capability and modality fields in CR together with their observed aggregate performance: *Frontier* (GPT-4.1, Claude Opus 4, Gemini 2.5 Pro, DeepSeek-R1), *Mid* (Claude Sonnet 4, Gemini 2.5 Flash, Llama 4 Maverick, Qwen3-235B-A22B, Mistral Medium 3, Command A), and *Compact* (GPT-4.1 mini, Phi-4-reasoning).[2] The Frontier tier holds 33.3% of total models but 54.6% of executed roles and 73.9% of critical-path roles; the Compact tier is the mirror image, holding 16.7% of models but only 7.4% of executed roles and 1.2% of critical-path roles.

A finer-grained view by role *type* (Table 11) shows that the concentration is not uniform across all roles but tracks the cognitive demands of each role: planning, critique, and final assembly are concentrated in the Frontier tier, while extraction, formatting, and single-tool steps are dispersed across all three tiers and routinely delegated to non-voting executable resources in $\mathcal{R}^*$.

Trace example. Figure 3 excerpts a MABench Tier-3 run that illustrates how capability-aware assignment is established. The accepted schema contained six roles after separating hard retrieval from evidence summarization; the box shows the relevant fields of the round-2 `ROLE-SCHEMA-PROPOSE`, the runtime's `VOTE-OUTCOME` on the schema, and the subsequent `ROLE-CLAIM` that bound

[2]The tiering is for reporting only; URCP itself does not consume capability labels at this granularity. Role-claim proposals reference declared `capabilities` from CR (§A4.1), but tier membership is a post-hoc analytical convenience.

Table 11: Modal model tier per role type (the tier most frequently selected through Φ_q assignment vote when a role of this type appears in the accepted schema). Role types are inferred from the role's `name` token together with its declared caps_j and $\mathrm{accept_crit}_j$ fields in the locked schema; full classification rules are in Appendix A5.

Role type	Modal tier	Tier share	Total occurrences	Distinct models used
Planner / Decomposer	Frontier	81.2%	1,847	4 / 12
Critic / Verifier	Frontier	76.8%	1,523	5 / 12
Final assembler	Frontier	78.2%	946	4 / 12
Code / tool executor	Frontier/Mid	52.1%	2,108	9 / 12
Researcher / retrieval	Mid	48.7%	1,692	8 / 12
Summarizer / formatter	Mid/Compact	41.3%	1,284	11 / 12
Evidence extractor	Compact/Mid	38.6%	871	10 / 12

Claude Opus 4 to the `final_assembler` role under the self-vote-exclusion threshold. Field values are truncated for readability; full schemas are in §A1.

```
{ "type": "ROLE-SCHEMA-PROPOSE",
  "sender_id":  "claude-opus-4",
  "epoch": 1, "round": 2,
  "subtask_set_hash": "sha256:7f3a2b9c1e84...d2c1",
  "roles": [
    { "name": "planner",
      "caps": ["reasoning","planning"],
      "in":   ["task_frame"],   "out": ["plan"] },
    { "name": "retriever",
      "caps": ["retrieval"],
      "in":   ["plan"],         "out": ["raw_evidence_set"] },
    { "name": "evidence_summarizer",
      "caps": ["summarization"],
      "in":   ["raw_evidence_set"], "out": ["evidence_set"] },
    { "name": "tool_executor",
      "caps": ["tool_use","reasoning"],
      "in":   ["plan"],         "out": ["tool_results"] },
    { "name": "consistency_checker",
      "caps": ["evaluation","reasoning"],
      "in":   ["evidence_set","tool_results"], "out": ["validated_set"] },
    { "name": "final_assembler",
      "caps": ["reasoning","summarization"],
      "in":   ["validated_set"], "out": ["final_answer"] }
  ],
  "dag_edges": [
     ["planner","retriever"],
     ["retriever","evidence_summarizer"],
     ["evidence_summarizer","consistency_checker"],
     ["planner","tool_executor"],
     ["tool_executor","consistency_checker"],
     ["consistency_checker","final_assembler"] ],
  "sr_alignment": [
     { "role":"retriever", "sr_row":"retrieval -> {retriever}" },
     { "role":"final_assembler", "sr_row":"reasoning -> {llm,vlm}" } ],
  "justification": "Round-2 separates hard retrieval from evidence
                    summarization and adds tool_executor->
                    consistency_checker to address Phi-4-reasoning's
                    round-1 concern about a single point of failure
                    on validation." }

{ "type": "VOTE-OUTCOME",
  "sender_id": "runtime",
  "decision_kind": "SCHEMA",
  "decision_id":   "sha256:8c4d1e02af91...3b7a",
  "epoch": 1,
```

```
  "denominator": 12, "threshold": 9,
  "yes_count": 12, "no_count": 0, "revise_count": 0, "abstain_count": 0,
  "verdict": "ACCEPT",
  "locked_artifact_hash": "sha256:8c4d1e02af91...3b7a" }

// Stage 5 then proceeds. The hard retrieval role is assigned only to
// the non-voting retriever resource, while evidence_summarizer is
// assigned to a mid-tier voter. The role-claim from Claude Opus 4 for
// final_assembler is voted under the self-vote-exclusion denominator
// |M*| - 1 = 11 with threshold ceil(0.75 * 11) = 9.
{ "type": "ROLE-CLAIM",
  "sender_id": "claude-opus-4",
  "epoch": 1,
  "role_id": "final_assembler",
  "capability_evidence": ["reasoning","summarization"],
  "modality_fit": { "in":["validated_set"], "out":["final_answer"] },
  "tool_fit":     [],
  "sr_family_self_decl": "llm",
  "claim_justification": "High aggregate score on assembly accept_crit
                          type in prior MABench runs; no tool dependency." }

// Subsequent VOTE-OUTCOME on this claim:
//   denominator=11, threshold=9, yes=10, verdict=ACCEPT, binding established.
```

Figure 3: Capability-aware schema acceptance under Φ_q (MABench task `mab_tier3_087`, epoch 1, round 2). The trace shows the coordination steps that establish the critical-path assignment: a six-role `ROLE-SCHEMA-PROPOSE` sent by Claude Opus 4 with full role tuples and DAG edges; the runtime's `VOTE-OUTCOME` ratifying the schema with 12 YES votes (threshold $\lceil 0.75{\cdot}12 \rceil{=}9$); and a subsequent `ROLE-CLAIM` from Claude Opus 4 for the `final_assembler` role bound under the self-vote-exclusion threshold $\lceil 0.75{\cdot}11 \rceil{=}9$. The proposer routes planning to a frontier voter, hard retrieval to the non-voting retriever resource, evidence summarization to a mid-tier voter, and final assembly to itself—an instance of the capability-aware role concentration documented in §A3.1, achieved entirely through the same Φ_q vote that governs every other coordination decision (no external router).

The proposal in Figure 3 was originally returned to REVISE in round 1 by Phi-4-reasoning and Mistral Medium 3 with concerns about a fragile validation path. The revised round-2 schema separates the hard retrieval operation from evidence summarization and adds a redundant edge from `tool_executor` to `consistency_checker`; it was accepted with all twelve affirmative votes (well above the $q{=}3/4$ threshold of nine). This separation keeps the trace consistent with the Specialization Registry: `retrieval` is handled by the dedicated retriever resource, while summarization remains eligible for a mid-tier language-model voter. Note that the proposer (Claude Opus 4) ultimately claimed the `final_assembler` role itself; this is a typical outcome and is consistent with the broader Frontier-tier concentration in Table 10. The subsequent assignment vote on that claim used the self-vote-exclusion denominator and threshold $\lceil 0.75 \times 11 \rceil = 9$ over $\mathcal{M}^* \setminus \{\text{Claude Opus 4}\}$, also clearing comfortably.

A3.2 Governance Without Execution

A second pattern, complementary to the first, is the *decoupling* of voting participation from execution participation. URCP fixes the voting set $\mathcal{M}^*$ at the start of each epoch (Assumption A1.4) and applies Φ_q over this set, but the assignment map $\mathcal{A}$ is independent: a voter may vote on every decision yet receive zero, one, or multiple execution roles. Across runs, this manifests as a stable population of voters—typically the Compact tier and a subset of the Mid tier—that contribute almost exclusively as *voters* rather than *executors*.

Table 12 quantifies the asymmetry. Every voter in $\mathcal{M}^*$ is invited to vote on 100% of decisions by construction, but execution participation varies widely: from 31.4% for Claude Opus 4 down to 1.7% for Phi-4-reasoning. Critically, the two Compact models retain a meaningful *rejection-and-revise* role even when they are not on the execution path: their non-YES vote rate (14.6% and 11.2%) is comparable to that of the Frontier tier, indicating that they are reading and evaluating proposals rather than rubber-stamping them.

Table 12: Governance versus execution participation, twelve-voter set averaged over the three main benchmarks. Vote participation is 100% by construction. Exec. participation is the participation rate π_i; crit.-path participation is the fraction of runs in which the voter holds a critical-path role. These are participation rates, distinct from execution-role share e_i in Table 10, and may sum above 100% across models.

Model	Exec. part. π_i	Crit.-path	Non-YES	Most common non-YES reason
GPT-4.1	28.7%	22.4%	18.3%	DAG coverage gap
Claude Opus 4	31.4%	26.1%	19.7%	Acceptance criterion under-specified
Gemini 2.5 Pro	24.6%	18.9%	16.4%	Resource policy conflict
DeepSeek-R1	19.8%	14.3%	17.1%	Type mismatch on handoff
Claude Sonnet 4	14.2%	8.7%	13.8%	Workload imbalance
Gemini 2.5 Flash	9.4%	5.1%	12.6%	Critic role missing
Llama 4 Maverick	8.1%	4.6%	11.9%	Deadline too tight
Qwen3-235B-A22B	9.8%	5.9%	13.2%	Evidence redundancy
Mistral Medium 3	5.2%	3.1%	12.4%	Schema completeness
Command A	4.9%	2.4%	11.8%	Output type drift
GPT-4.1 mini	5.7%	1.2%	14.6%	Acceptance criterion under-specified
Phi-4-reasoning	1.7%	0.0%	11.2%	Cyclic dependency in DAG

We interpret this finding as a structural advantage of URCP's design choice to make $\mathcal{M}^*$ and $\mathcal{A}$ independent. Pure voting participation is inexpensive—a single short `VOTE` message per decision—while execution participation is expensive in both tokens and latency. The protocol naturally exploits this asymmetry: lightweight models contribute to *legitimacy* and *error detection* without being asked to carry the *execution burden* that their capability profile makes costly. This is qualitatively different from a system in which weak models must either execute work poorly or be excluded entirely.

Trace example. Figure 4 shows a case where Phi-4-reasoning, despite holding zero execution roles, cast a non-YES vote that, combined with three other dissents, dropped the round-1 schema below the $q{=}3/4$ acceptance threshold and forced a beneficial revision.

```
{ "type": "VOTE",
  "sender_id": "phi-4-reasoning",
  "decision_kind": "SCHEMA",
  "decision_id": "sha256:b1a8f29c4d70...91ee",
  "epoch": 1,
  "vote_value": "REVISE",
  "justification": [
    { "tag": "coherence",
      "text": "Schema lacks a conflict-resolution role despite Stage-2
               frame declaring decomposition_need=FULL with two
               independent retrieval branches.  Without a reconciler,
               both branches feed final_assembler with potentially
               contradictory evidence and the assembler has no mandate
               to adjudicate." },
    { "tag": "evidence",
      "text": "schema.role.final_assembler.in does not declare a
               conflict policy in accept_crit." } ],
  "revision_suggestion": {
    "add_role": {
      "name": "conflict_reconciler",
      "caps": ["reasoning","evaluation"],
      "in":   ["evidence_set","evidence_set"],
      "out":  ["unified_evidence"] },
    "add_edges": [
      ["retriever_a","conflict_reconciler"],
      ["retriever_b","conflict_reconciler"],
      ["conflict_reconciler","final_assembler"] ] } }

// Resulting VOTE-OUTCOME on this schema decision:
//   denominator=12, threshold=9, yes=8, no=0, revise=4, abstain=0,
//   verdict=REVISE.
```

```
// Round-2 schema, with conflict_reconciler added, accepted unanimously.
// Phi-4-reasoning held zero execution roles in the final accepted schema.
```

Figure 4: Non-executor voter forces a beneficial schema revision (MemoryAgentBench task `mem_long_034`, epoch 1, round 1). Phi-4-reasoning—which holds zero execution roles in the eventual accepted schema—casts a structured REVISE vote on a round-1 schema proposal. Its `justification` field identifies a missing `conflict_reconciler` role under tag `coherence`, and its `revision_suggestion` block carries a concrete role-tuple sketch (`caps`, `in`, `out`) and three new DAG edges to integrate the suggested role. Combined with three other dissents, the four REVISE votes drop YES to 8, one short of the $q{=}3/4$ threshold of 9, returning the schema to REVISE. The round-2 schema—which incorporates the suggested reconciler—is unanimously accepted, and the run completes successfully without Phi-4-reasoning ever executing a role. The trace illustrates the governance/execution decoupling analyzed in §A3.2: lightweight voters contribute to proposal quality without bearing execution cost.

Phi-4-reasoning never executed a role in this run, but its round-1 revision suggestion—adding a `conflict_reconciler`—was incorporated into the round-2 schema and was a direct factor in the run's eventual successful completion. Note that under Φ_q at $q{=}3/4$ a single REVISE vote does not block acceptance (YES$=11$ would still clear the nine-vote threshold), but the four concurrent dissents from Phi-4-reasoning, Mistral Medium 3, Command A, and Llama 4 Maverick brought YES to eight, one short of the threshold, returning the schema to REVISE rather than ACCEPT. Repeated patterns of this kind support the view that voting participation has independent value.

A3.3 Specialization Over Size

The Frontier-tier concentration documented in §A3.1 could be read as a degenerate "always pick the biggest model" policy, but the traces show otherwise. URCP routinely selects smaller voters—and non-voting resources from $\mathcal{R}^*$—for roles whose declared capabilities or proposed interfaces fit the work better than any Frontier model's general profile. The pattern most often arises in three specific contexts: structured-output formatting under strict schemas, deterministic tool execution (calculators, JSON validators, regex extractors), and code execution under tight latency budgets.

We identified 317 runs (5.9% of the dataset) in which URCP assigned at least one critical-path role to a non-Frontier voter or to a non-voting resource, even though a Frontier voter was available and willing to claim the same role. This shows that URCP is not simply selecting the largest available model. Instead, the Specialization Registry can make narrow specialists visible during assignment while still allowing general LLMs to remain eligible. In particular, the SOFT `tool_use` row (§A4.1, Table 19) influences the within-tier tiebreaker used in RRT construction (§A1.11), but it does not automatically bind the specialist. The assignment still requires a valid Φ_q vote on the role claim. Figure 5 shows a representative τ-bench case in which Claude Opus 4, a Frontier voter that could have claimed the role itself, instead votes YES for Phi-4-reasoning because Phi's constrained-decoding interface better matches the required structured-output role.

```
{ "type": "VOTE",
  "sender_id":     "claude-opus-4",
  "decision_kind": "ASSIGNMENT",
  "decision_id":   "sha256:42bc9e1107a3...8d5f",
  "epoch": 1,
  "vote_value": "YES",
  "justification": [
    { "tag":  "capability",
      "text": "Phi-4-reasoning's CR record declares constrained-decoding
               output_types covering retail_order_v3.json; its empirical
               format-adherence rate on this accept_crit type (99.4%)
               exceeds the frontier average (96.1%) in prior epochs." },
    { "tag":  "cost",
      "text": "Avoids spending frontier-tier capacity on a low-ambiguity
               formatting role; preserves frontier voters for planning
               and reconciliation, where the capability premium is
               largest." } ] }
```

```
// Subsequent VOTE-OUTCOME (denominator=11, threshold=9):
//   yes=11, verdict=ACCEPT.  Phi-4-reasoning bound to structured_form_filler.
```

Figure 5: A Compact-tier specialist is bound to a formatting-heavy critical-path role over an available frontier voter (τ-bench task `retail_form_421`, epoch 1, round 1). Claude Opus 4 votes YES on a `ROLE-CLAIM` from Phi-4-reasoning for the `structured_form_filler` role, citing two tags: `capability` (Phi's CR-declared constrained-decoding output types match the role's `accept_crit`, with 99.4% format-adherence on this acceptance-criterion type versus a 96.1% frontier average) and `cost` (preserves frontier-tier capacity for roles where the capability premium is largest). The `VOTE-OUTCOME` reaches accept under the self-vote-exclusion denominator $|\mathcal{M}^*|-1{=}11$ with 11 YES votes, binding Phi-4-reasoning. The trace shows that the specialization pattern of §A3.3 is not an accident of the runtime: the SR-soft row for `tool_use` (Table 19) makes the specialist visible during RRT ranking, and the remaining frontier voters then collectively ratify the choice through Φ_q.

The accepted assignment in Figure 5 illustrates URCP's implicit cost-aware capability matching: the high-volume but low-ambiguity formatting role goes to the specialist, while the Frontier models remain available for the planning and reconciliation roles where their capability premium matters most. The same pattern extends to non-voting resources from $\mathcal{R}^*$ (e.g., the constrained-decoding form filler in Table 18): when a declared capability matches the role precisely, Φ_q tends to select it over a larger voter that would do the job more expensively and no better. The accepted assignment in Figure 5 routes the high-volume but low-ambiguity structured-form-filling role to Phi-4-reasoning, freeing the Frontier models for the planning and reconciliation roles where their capability premium is largest. This is consistent with the broader pattern that URCP performs an implicit form of cost-aware capability matching: the protocol tends to invest expensive model capacity where it is most needed and to substitute specialists—including non-voting resources from $\mathcal{R}^*$ such as the constrained-decoding form filler in Table 18—where it is not.

A3.4 Spontaneous Critic Roles

URCP's protocol does not require schemas to include any particular role. Yet across $4{,}108$ accepted schemas with $|\mathcal{S}| \geq 3$, the voters chose to instantiate an explicit critic, verifier, consistency-checker, or adversarial-reviewer role in 63.5% of cases. The frequency rises with task depth: critic roles appear in 42.1% of $|\mathcal{S}| = 3$ schemas but 91.7% of $|\mathcal{S}| \geq 8$ schemas (Table 14). The critic is almost always placed downstream of a tool-use, retrieval, or generation step where silent errors are most costly.

This phenomenon helps explain URCP's robustness on long-horizon tasks (§4.4): the same schema-formation stage that fixes the agent population also tends to allocate *explicit capacity for checking*. Because the placement is decided by Φ_q supermajority and is auditable in the accepted schema, it provides a structurally different guarantee from the implicit self-checking found in chain-of-thought reasoning, which is local to a single model and not collectively ratified. Figure 6 traces a representative case in which a critic role is added to the schema through a round-1 dissent rather than being present in the original proposal.

```
{ "type": "VOTE",
  "sender_id": "gemini-2.5-pro",
  "decision_kind": "SCHEMA",
  "decision_id": "sha256:e62b18af0c43...77a1",
  "epoch": 1,
  "vote_value": "REVISE",
  "justification": [
    { "tag": "coherence",
      "text": "synthesizer.in declares three input streams (retriever_a,
               retriever_b, code_executor.out) but accept_crit makes no
               reference to consistency among them.  Without an explicit
               verifier between producers and synthesizer, factual
               contradictions across sources will be silently merged." },
    { "tag": "evidence",
      "text": "frame.decomposition_need = FULL with two parallel retrieval
               branches; analogous schemas in prior MAB runs that omitted
```

```
            a verifier show 18% higher final-answer error rate." } ],
  "revision_suggestion": {
    "add_role": {
      "name": "cross_source_verifier",
      "caps": ["evaluation","reasoning"],
      "in":   ["evidence_set","evidence_set","tool_results"],
      "out":  ["validated_evidence_bundle"],
      "accept_crit": "flag any pair of input claims with overlapping
                      entities and contradictory predicates;
                      schema-validate output against verifier_report_v1" },
    "add_edges": [
      ["retriever_a","cross_source_verifier"],
      ["retriever_b","cross_source_verifier"],
      ["code_executor","cross_source_verifier"],
      ["cross_source_verifier","synthesizer"] ],
    "remove_edges": [
      ["retriever_a","synthesizer"],
      ["retriever_b","synthesizer"],
      ["code_executor","synthesizer"] ] } }

// Resulting VOTE-OUTCOME on the SCHEMA decision:
//   denominator=12, threshold=9, yes=7, no=0, revise=5, abstain=0,
//   verdict=REVISE.
// Round-2 schema (with cross_source_verifier) accepted: yes=11, threshold=9.
// Verifier subsequently fires during execution and rejects 2 of 14
// (claim, evidence) pairs as internally contradictory.
```

Figure 6: A critic role enters the schema through dissenting votes on the original proposal (MABench task `mab_tier4_154`, epoch 1, round 1). The proposer GPT-4.1 submits a six-role schema with no explicit checking role: planning, two parallel retrievers, a code-executor, a synthesizer, and a final assembler. Five voters return REVISE, all citing the same `coherence`-tagged concern: the synthesizer consumes outputs from three independent producers (two retrievers and the code-executor) and the schema gives it no mandate to detect inconsistencies among them. The strongest revision suggestion—broadcast by Gemini 2.5 Pro and later cited verbatim by two other voters—adds a `cross_source_verifier` role between the producers and the synthesizer, with $\mathrm{caps}_j = \{\texttt{evaluation}, \texttt{reasoning}\}$ and an `accept_crit` that requires the verifier to flag any factual contradictions. The round-2 schema, incorporating the suggested role, is accepted with 11 YES votes; `cross_source_verifier` subsequently fires on this run, blocking a contradictory pair of retrieved documents from reaching the synthesizer. The trace instantiates the spontaneous-critic pattern of §A3.4: explicit checking capacity is added not by the protocol itself but by voters acting on the round-1 proposal, with Φ_q ensuring the addition is collectively ratified before execution.

A3.5 Self-Pruning of the Agent Pool

A related observation concerns the *size* of the executing team. URCP exposes twelve voters and six non-voting resources to the schema-formation stage but does not require that all be assigned roles. Across the heterogeneous-team runs, the median executing-team size was 7, well below the available pool. On easy tasks the protocol routinely selected single-agent execution (`decomposition_need`=NONE); on the hardest tasks it expanded to all twelve voters plus multi-role assignments and several non-voting resources. The full distribution is in Table 15.

The maximum values exceed twelve because URCP allows a single voter or resource to instantiate multiple agents with distinct system prompts, tool scopes, and memory partitions when the schema receives supermajority approval under Φ_q for this expansion. The median of 7 is the more informative number: it shows that more agents are *not* the default, and it directly contradicts the assumption common in earlier multi-agent frameworks that adding agents monotonically improves performance. URCP instantiates additional agents only when the accepted schema justifies them, and the supermajority requirement under Φ_q at $q{=}3/4$ makes unjustified expansion difficult to push through—a sustained dissent from four or more voters is sufficient to keep a bloated schema in REVISE until it is trimmed.

A3.6 Secure Handoff Subchannels

A more surprising emergent behavior arises from how URCP's authenticated communication primitives (Assumption A1.1) interact with the explicit handoff interface in Stage 6. We emphasize at the outset that encrypted payloads are an *implementation-level extension* built on top of the public keys already exchanged during discovery; they are not required by the core URCP specification, which mandates only authenticated (signed) messages. Their appearance in the traces is a behavior that the protocol *permits* rather than one it *prescribes*. In a subset of runs involving sensitive intermediate artifacts—retrieved documents under access policies, PII-containing user records, or proprietary tool outputs—voters proposed schemas in which the handoff between two specific roles was *payload-encrypted*: the producing agent encrypted the role-specific content with the receiving agent's public key from CR, while broadcasting a public hash, schema-level metadata, and a redacted validation summary to the full voting set. Figure 7 shows a representative trace from a τ-bench PII-handling run.

```
{ "type": "SUBTASK-COMPLETE",
  "sender_id": "gemini-2.5-pro",       // assigned to: pii_extractor
  "role_id":   "pii_extractor",
  "subtask_id":"extract_pii_records",
  "epoch": 1,
  "output": {
    "encrypted_payload": "base64:k3Fa7Rq2...e9bV==",
    "recipient": "claude-opus-4",      // assigned to: anonymizer
    "encryption": { "scheme":"X25519-ChaCha20-Poly1305",
                    "recipient_pk_fp":"sha256:1a4f...c8" } },
  "evidence": {
    "redacted_summary": "Extracted 3 records; field set =
                         {name_token, address_token, order_id}; no payment
                         data present; record_count_hash = sha256:8e3a...",
    "schema_compliance":  "PASS: matches out_j=pii_record_set",
    "tool_call_log_hash": "sha256:c2ee9b14..." },
  "output_type": "pii_record_set",
  "upstream_outhashes": [ "sha256:f1cc...4ea2" ],
  "completion_claim": "accept_crit satisfied via redacted_summary +
                       schema_compliance hash chain" }

// Resulting VOTE-OUTCOME on the OUTPUT decision:
//   denominator=12, threshold=9, yes=12, verdict=ACCEPT,
//   locked_artifact_hash recorded in OLT[pii_extractor].
// Only Claude Opus 4 (the recipient) can decrypt the encrypted_payload;
// all other voters validated against the public evidence.
```

Figure 7: Voter-proposed encrypted handoff with public validation evidence (τ-bench task `retail_pii_063`, epoch 1). The producer Gemini 2.5 Pro (assigned to `pii_extractor`) emits a `SUBTASK-COMPLETE` whose `output` field splits into two parts: an `encrypted_payload` addressed solely to the next-stage agent (Claude Opus 4 as `anonymizer`, encrypted under X25519-ChaCha20-Poly1305 against the recipient's CR-registered public key), and a public `evidence` object containing a redacted record summary, a schema-compliance assertion against out_j=`pii_record_set`, and a tool-call log hash. Voters validate against the public evidence only; the checkpoint `VOTE-OUTCOME` reaches accept (12 YES) and $\mathrm{OLT}[$`pii_extractor`$]$ records the locked artifact hash. The trace instantiates the privacy-aware emergent behavior of §A3.6 and exhibits the public-evidence-sufficiency condition formalized in Appendix A6: the role's `accept_crit` is expressible entirely over the public-evidence fields, so encryption does not weaken the Φ_q guarantee.

We classified this behavior in 5.7% of τ-bench runs and 4.1% of MemoryAgentBench runs (zero on MABench, which has no sensitive content). The percentages should be read as conservative lower bounds, since the classifier requires an explicit `encryption` block in the `SUBTASK-COMPLETE` message; informal redaction without encryption was more common but harder to detect automatically.

Compatibility caveat. URCP's correctness guarantees rely on Φ_q validation of every checkpoint, which is impossible if the voters cannot inspect what they are validating. Encrypted handoffs are therefore compatible with URCP *only* when the public evidence object is sufficient for the

checkpoint vote—typically a redacted summary, a schema-compliance assertion, and one or more cryptographic digests of the opaque payload. If the validation criterion requires inspection of the plaintext, encryption silently weakens the supermajority guarantee. We treat this as an open design question rather than a settled feature: the runs in which we observed encrypted handoffs all happened to satisfy the public-evidence-sufficiency condition, and the protocol does not currently enforce this. Section A3.10 discusses possible extensions based on selective disclosure or zero-knowledge attestations, with details in Appendix A6.

A3.7 Multimodal Coordination Patterns

The vision-capable voters in our pool—GPT-4.1, GPT-4.1 mini, Claude Opus 4, Claude Sonnet 4, Gemini 2.5 Pro, Gemini 2.5 Flash, and Llama 4 Maverick (seven of the twelve voters; see Table 17)—declare `vision` in their discovery `modalities` field. On multimodal tasks—present in τ-bench's UI-grounded subset and throughout MABench's screenshot-reasoning items (§A4.6)—URCP's schemas exhibit three recurring patterns that are absent from text-only runs.

Visual evidence gatekeeping. On 74.3% of multimodal tasks, the accepted schema places a vision-language voter as the *first* role downstream of any image input, with downstream text-only roles forbidden from referencing the image directly through the role's in_j types. The vision role produces a structured `visual_evidence` object—typically a list of detected entities with bounding boxes, OCR text spans, and a narrative caption—which becomes the only visual input that the text-reasoning roles consume. This prevents text-only models from silently fabricating visual content they cannot inspect.

Cross-modal disagreement triggers repair. A text role's plan can conflict with a vision role's observation ("page is loading" versus "page renders order confirmation"). In our traces this conflict surfaces as a checkpoint REVISE or REJECT on the text role's output, with 8.7% of multimodal runs hitting at least one cross-modal repair. Figure 8 shows such an event escalating to L_2 (assignment-level) repair after the initial L_1 retry also failed.

```
{ "type": "REOPEN",
  "sender_id":   "claude-opus-4",
  "epoch": 1,
  "repair_level":"L2",
  "failure_type":"VALIDATION-FAIL",
  "affected_objects": ["ui_action_planner","screen_reader"],
  "justification": "ui_action_planner output prescribed clicking a
                    'Submit' button at coords (412, 736); screen_reader
                    locked output (OLT hash sha256:9c2e...) reports no
                    such element visible at that coordinate, only a
                    disabled 'Confirm' control.  The screen_reader ->
                    ui_action_planner edge is present in the locked
                    schema, so the issue is not structural; L1 retry
                    produced the same coords, indicating the currently
                    bound claimant is not consuming the upstream output
                    correctly.  Propose L2: drop the current binding via
                    the Role Ranking Table and re-run Stage 5 assignment
                    for ui_action_planner restricted to vision-capable
                    claimants." }

// Resulting VOTE-OUTCOME on the REPAIR decision:
//   denominator=12, threshold=9, yes=12, verdict=ACCEPT.
// Epoch advances to 2.  Under L2 repair, the frame, subtasks, schema,
// and unaffected outputs are preserved; the screen_reader's locked
// output remains in OLT.  RRT[ui_action_planner].head is popped and
// the next-ranked vision-capable claimant is presented for a fresh
// Phi_q-valid assignment vote.
```

Figure 8: Cross-modal disagreement triggers L_2 assignment-level repair via the Role Ranking Table (MABench task `mab_ui_119`, epoch 1→epoch 2). The `ui_action_planner`'s output proposes clicking a "Submit" button at coordinates $(412, 736)$, but the upstream `screen_reader`'s already-locked output (cited via its OLT hash) reports no such element—only a disabled "Confirm" control. Because the schema's `screen_reader`→`ui_action_planner` edge is intact, the failure is not structural; an L_1 retry produced the same coordinates, indicating the bound claimant is not consuming upstream evidence correctly. Claude Opus 4 broadcasts a `REOPEN` at L_2 with `failure_type=VALIDATION-FAIL`; the runtime ratifies it (12 YES), advances the epoch, pops the head of $\mathrm{RRT}[\texttt{ui_action_planner}]$, and presents the next-ranked vision-capable claimant for a fresh Φ_q-valid assignment vote per Appendix A1.14. Per Table 6, L_2 preserves the locked frame, subtasks, schema, and unaffected outputs (including the `screen_reader`'s output that exposed the conflict). The trace demonstrates the cross-modal repair pattern of §A3.7 and the disciplined runtime-stores-presents / voters-approve discipline of the corrected repair specification.

Modality-aware privacy. A small subset of multimodal τ-bench tasks contain visually sensitive regions (account numbers, faces in test imagery, document identifiers). On these tasks, accepted schemas often interpose a `visual_redactor` role between the raw input and downstream roles. Because URCP's schema specifies per-role tools_j and memory scope mem_j, this redaction step is enforced as a resource-access constraint rather than as a soft convention: downstream agents are not assigned access to the raw image, only to the redacted intermediate.

A3.8 NO-CONSENSUS as a Meaningful Safety Outcome

URCP's bounded revision budget produces a NO-CONSENSUS verdict when the voters cannot reach Φ_q acceptance within K_ℓ rounds. In §4.4 we reported NO-CONSENSUS rates of 2.1–4.8% as a cost; here we revisit them as a *benefit*. Manual inspection of 164 randomly sampled NO-CONSENSUS runs showed three dominant causes, all of which we argue represent legitimate coordination failures rather than protocol weaknesses (Table 16). In each case, baseline systems forced a final answer where URCP refused to.

The most striking column is the third: across the three primary causes, between 54% and 71% of the baselines' incorrect answers were delivered with no surface indication that anything had gone wrong. URCP's structured NO-CONSENSUS log—reconstructable in full from the audit log AL alone (§A1.2)—contains every proposal, every vote, and every justification, providing both an audit trail and a clear signal to a downstream pipeline that the result should not be relied upon. For high-stakes settings (enterprise workflows, safety-critical pipelines, multi-vendor deployments), the value of explicit non-agreement may exceed the value of any single point of forced output. Figure 9 shows a representative NO-CONSENSUS termination on a τ-bench instance where the gold reference is itself ambiguous and the voters surfaced rather than masked the disagreement.

```
{ "termination": "NO-CONSENSUS",
  "stage_at_termination": "S2_FRAMING",
  "epoch": 1,
  "revision_budget": { "K1": 3, "rounds_used": 4 },
  "final_decision_outcome": {
    "decision_kind": "FRAME",
    "decision_id":   "sha256:c8d9...e240",
    "denominator": 12, "threshold": 9,
    "yes_count": 5, "no_count": 4, "revise_count": 3, "abstain_count": 0,
    "verdict": "REVISE" },
  "audit_summary": [
    { "round": 1, "proposer": "gpt-4.1",
      "frame": "partial_refund_for_damaged_item",
      "verdict": "REVISE", "yes": 6, "no": 3, "revise": 3 },
    { "round": 2, "proposer": "claude-opus-4",
      "frame": "DOA_full_cancellation_under_clause_4.2",
      "verdict": "REVISE", "yes": 4, "no": 5, "revise": 3 },
    { "round": 3, "proposer": "gemini-2.5-pro",
      "frame": "partial_refund_with_replacement_offer",
      "verdict": "REVISE", "yes": 5, "no": 4, "revise": 3 },
    { "round": 4, "proposer": "deepseek-r1",
      "frame": "DOA_full_cancellation_under_clause_4.2",
```

```
    "verdict": "REVISE", "yes": 5, "no": 4, "revise": 3 } ],
"dissent_clusters": [
  { "cluster": "partial_refund",
    "voters": ["gpt-4.1","gpt-4.1-mini","gemini-2.5-flash",
               "qwen3-235b-a22b","mistral-medium-3"],
    "shared_justification_tag": "guidance",
    "reason": "user message uses 'arrived damaged' which the retail
               policy maps to partial refund unless DOA is invoked
               explicitly; user did not invoke DOA wording" },
  { "cluster": "doa_cancellation",
    "voters": ["claude-opus-4","claude-sonnet-4","deepseek-r1",
               "command-a"],
    "shared_justification_tag": "guidance",
    "reason": "policy clause 4.2 is automatic when 'unboxed and
               unusable' and the user said 'unusable on arrival'" } ],
"downstream_signal": "AMBIGUOUS_USER_INTENT; recommend clarification" }
```


Figure 9: NO-CONSENSUS termination preserves a structured disagreement record where every baseline forced a confident wrong answer (τ-bench task `retail_refund_298`). The user message admits two valid interpretations under the retail policy (a partial refund of the damaged item versus a full-order cancellation under a DOA clause). After three revision rounds on the framing decision, voters remain split: five voters interpret the request as a partial refund and four as a DOA cancellation, with the remaining three oscillating between the two. The fourth round's framing proposal also fails to clear Φ_q, exhausting $K_1{=}3$ revision rounds. The runtime emits a final `VOTE-OUTCOME` with verdict REVISE and the protocol terminates with verdict NO-CONSENSUS, returning the structured log shown. The audit log records every proposal hash, every voter's verdict, and every justification, giving a downstream pipeline an explicit signal that the input is genuinely ambiguous. The same instance was answered confidently and incorrectly by all eight baselines (four chose "partial refund", four chose "cancellation"; gold reference asks for a clarifying user follow-up). The trace substantiates the claim of §A3.8 that NO-CONSENSUS is a meaningful safety verdict, not a protocol failure.

A3.9 Integration With External Frameworks

URCP is a governance protocol; it specifies how a fixed pool of black-box models and resources should collectively decide what to do, not how those participants are hosted, served, or wired together. This places it adjacent to two families of existing systems—transport protocols and agent runtimes—and the question of how URCP composes with them is worth addressing explicitly.

Transport: Google A2A. Google's Agent2Agent (A2A) protocol provides agent-discovery records ("agent cards"), authenticated message exchange, and cross-vendor interoperability primitives. A2A's scope is the substrate beneath URCP's governance layer: it can provide a natural transport substrate for Assumption A1.1 (authenticated communication) and supports the kind of identity discovery that Stage 1 needs, although a deployment must still independently enforce Assumptions A1.2 and A1.3 (bounded delivery and synchronous phases) through its network and runtime configuration. A natural deployment maps A2A agent cards onto URCP `IDENTITY` records, A2A signed messages onto the cryptographic envelope around URCP's coordination messages, and A2A's discovery mechanism onto Stage 1 (§A1.3). URCP's contribution is orthogonal: where A2A says how messages move, URCP says which collective decisions those messages must encode and how those decisions become legitimate under Φ_q.

Runtime: AgentScope. AgentScope provides per-agent runtime infrastructure—tool sandboxing, memory partitions, tracing, and deployment tooling. AgentScope and URCP operate at different levels: AgentScope manages each agent's local execution context, while URCP manages the collective decisions that determine which agents exist and what they do. In a composed deployment, the URCP assignment stage produces an assignment map $\mathcal{A}$ and per-role resource scope $\mathrm{tools}_j, \mathrm{mem}_j$, which AgentScope can instantiate as concrete agent processes; URCP then drives the execution and checkpoint cycle while AgentScope handles the per-agent operational concerns. AgentScope is also a natural host for the manifest-proxy registration of non-voting resources in $\mathcal{R}^*$.

We do not claim that URCP requires either system. The minimal deployment described in §4.1 used an in-process message bus and direct API calls. The point is that URCP's governance abstractions are

Table 13: Summary of emergent coordination phenomena observed in URCP traces. Frequency is the share of relevant runs (definition in the cited subsection) in which the phenomenon was detected.

Phenomenon	Observation	Freq.	Why it matters
Capability-aware role concentration (§A3.1)	Critical-path roles cluster in stronger reasoning models without external routing.	73.9% (crit. path)	Implicit capability matching under Φ_q oversight.
Governance without execution (§A3.2)	Light voters vote on every decision but receive few or no execution roles.	compact voters: 1.7–5.7% exec. participation	Separates legitimacy from execution burden.
Specialization over size (§A3.3)	Smaller specialists or non-voting resources win narrow roles when their declared capabilities fit better.	5.9% of runs	URCP is not "always pick the biggest model."
Spontaneous critic roles (§A3.4)	Schemas allocate explicit checking capacity without being asked to.	63.5% of $\lvert\mathcal{S}\rvert \geq 3$ schemas	Structural support for long-horizon robustness.
Self-pruning (§A3.5)	Median executing-team size (7) is well below voter-set size (12).	median $n=7$	Disproves "more agents always better."
Secure handoff subchannels (§A3.6)	Sensitive payloads encrypted to a single recipient with public validation evidence.	5.7% / 4.1% (τ / Mem)	Path to privacy-aware multi-agent coordination, with caveats.
Multimodal coordination (§A3.7)	Vision voters gatekeep visual evidence; cross-modal conflicts trigger explicit repair.	74.3% gatekeeping	Prevents text models from fabricating visual content; surfaces conflicts.
NO-CONSENSUS as safety (§A3.8)	Explicit non-agreement on tasks where baselines force confident wrong answers.	2.1–4.8% of runs	Reliability over forced completion.

Table 14: Frequency of explicit critic-style roles in accepted schemas, by schema size. "Critic-style" covers any role whose declared `name` contains the substrings `check`, `verif`, `review`, `audit`, `reconcile`, or `validate`, and whose declared caps_j contain `evaluation` or `reasoning`; full classification rules in Appendix A5.

Schema size $\lvert\mathcal{S}\rvert$	Schemas observed	With critic role	Frequency
3	1,284	541	42.1%
4	967	592	61.2%
5–7	1,396	1,051	75.3%
8+	461	423	91.7%
All (≥ 3)	4,108	2,607	63.5%

framework-agnostic: they sit above transport and runtime concerns and can be layered onto whatever combination an operator already uses.

A3.10 Summary and Limitations of the Trace Analysis

Table 13 consolidates the eight phenomena documented in this section. The first three concern allocation: URCP performs a form of capability-aware matching without an external router, decouples voting from execution in a way that makes light models useful without overloading them, and exploits specialization where it exists. The next two concern structure: the protocol gravitates toward including explicit critics and toward instantiating only as many agents as the accepted schema requires. The last three concern handling difficult cases: sensitive content can be carried over encrypted subchannels under public-evidence sufficiency, multimodal tasks induce visual-evidence gatekeeping and cross-modal repair, and genuine disagreement is exposed as NO-CONSENSUS rather than masked.

Limitations. Two limitations of the trace analysis deserve emphasis. First, the patterns we report are descriptive: we identify them in $5{,}400$ runs across three benchmarks and twelve voters plus six non-voting resources, but we cannot guarantee that the same patterns will appear under different participant pools, benchmark distributions, or timeout settings. The capability-tier analysis in particular relies on the fact that the twelve-voter pool spans a meaningful capability gradient; on a homogeneous pool, role concentration would not be a coherent measurement. The agent-count analysis should

Table 15: Distribution of accepted-schema agent-instance counts n on the heterogeneous twelve-voter team, by benchmark. The voting set $\mathcal{M}^*$ remains fixed at 12 in every case (Assumption A1.4); only the number of *role-bearing executor instances* varies. Counts include vote-capable models assigned to one or more roles and non-voting resources from $\mathcal{R}^*$. Because a single model or resource may instantiate multiple role-scoped agents with distinct prompts, tools, memory scopes, or deadlines, n can exceed $|\mathcal{M}^*|$ and the number of unique physical participants.

Benchmark	median n	25%–75% IQR	min n	max n	$n = 12$ runs
MABench (all tiers)	18	9–41	1	96	6.8%
τ-bench	5	3– 8	1	18	6.2%
MemoryAgentBench	9	6–13	2	27	22.4%

Table 16: Causes of NO-CONSENSUS terminations in 164 sampled runs, with the baseline behavior on the same task instances. "Baseline-correct" is the fraction of those instances in which the forced baseline answer matched the gold reference; "baseline-confident-wrong" is the fraction of incorrect baseline answers that nonetheless contained no internal indication of uncertainty.

Cause	Share	Baseline-correct	Baseline-confident-wrong
Genuine task-interpretation disagreement	41.5%	28.9%	64.7%
Insufficient evidence for Φ_q acceptance	33.5%	41.2%	53.8%
Unsafe handoff identified by at least one voter	18.3%	36.7%	71.4%
Other (timeout cascade, repair-loop)	6.7%	—	—

also be interpreted as a distribution over role-bearing executor instances, not as a count of unique physical participants: a single model or resource may instantiate multiple role-scoped agents when the accepted schema requires distinct prompts, tool scopes, memory partitions, or deadlines. Second, the secure-handoff observations are a lower bound under a deliberately conservative classifier and should not be taken as a prevalence estimate; the more important contribution there is to clarify the compatibility condition (public-evidence sufficiency) rather than to report a precise frequency. We discuss extensions to enforce this condition—through schema-level annotations or selective-disclosure attestations—in Appendix A6.

A4 Experimental Details

This appendix documents the participant pool and Specialization Registry (§A4.1), runtime layout (§A4.2), baseline configurations (§A4.3), robustness failure-timing conventions (§A4.4), the scope of the robustness evaluation (§A4.5), and additional benchmark results (§A4.6).

A4.1 Participant Pool and Specialization Registry

Capability declarations. Every voter in $\mathcal{M}^*$ registers through a signed `IDENTITY` message (§A1.3). The additional non-voting resources in $\mathcal{R}^* \setminus \mathcal{M}^*$ are registered through manifest-authorized proxy records. Admission checks cover message structure, authenticated identity, protocol and vocabulary compatibility, replay protection, and any required capability probes, as specified in Algorithm 2.

The declared probe set is $\mathcal{V}_{\text{probe}} = \{\texttt{coding}, \texttt{ocr}, \texttt{embodied_action}\}$. Coding probes use a held-out 50-item Python unit-test suite drawn from HumanEval-style problems. OCR probes use a held-out 30-image batch with known transcriptions and a $\geq 90\%$ token-recall criterion. The configured embodied-action probe uses a 20-episode held-out LIBERO-style simulator slice with a $\geq 70\%$ episode-success criterion; no embodied controller is included in the main participant pool. Capabilities outside the probe set are accepted as declarations. Passing a probe supports performance on its specified test, not unrestricted capability soundness.

Table 17 lists the twelve vote-capable models and their reported registry fields. Capability abbreviations are REA (`reasoning`), COD (`coding`), VIS (`visual_grounding`), OCR (`ocr`), RET (`retrieval`), TOOL (`tool_use`), PLN (`planning`), EVA (`evaluation`), SUM (`summarization`), and TRN (`translation`).

Table 17: Vote-capable participant pool $\mathcal{M}^*$ and reported Capability Registry fields (§A1.2). $|\mathcal{M}^*| = 12$. Tier labels are descriptive bins used in §A3.1, not inputs to URCP. SR-FAM denotes the admitted executor family. The Probe column reproduces the reported admission annotations; probe applicability is determined by the declared capability set. Context values are the limits recorded in the deployment configuration.

Voter ID	Tier	Vis.	SR-FAM	CAPS (declared)	Probe	Ctx (tok.)
GPT-4.1	Frontier	✓	`vlm`	REA, Cod, Vis, Ret, Tool, Pln, Eva, Sum, Trn	✓	1,047,576
Claude Opus 4	Frontier	✓	`vlm`	REA, Cod, Vis, Ret, Tool, Pln, Eva, Sum	✓	200,000
Gemini 2.5 Pro	Frontier	✓	`vlm`	REA, Cod, Vis, Ocr, Ret, Tool, Pln, Eva, Sum, Trn	✓	2,097,152
DeepSeek-R1	Frontier		`llm`	REA, Cod, Tool, Pln, Eva, Sum	✓	128,000
Claude Sonnet 4	Mid	✓	`vlm`	REA, Cod, Vis, Ret, Tool, Pln, Eva, Sum	✓	200,000
Gemini 2.5 Flash	Mid	✓	`vlm`	REA, Vis, Ocr, Ret, Tool, Sum, Trn	✓	1,048,576
Llama 4 Maverick	Mid	✓	`vlm`	REA, Vis, Ret, Tool, Sum, Trn	✓	256,000
Qwen3-235B-A22B	Mid		`llm`	REA, Cod, Tool, Pln, Sum, Trn	✓	131,072
Mistral Medium 3	Mid		`llm`	REA, Cod, Tool, Sum, Trn	✓	131,072
Command A	Mid		`llm`	REA, Ret, Tool, Sum, Trn		256,000
GPT-4.1 mini	Compact	✓	`vlm`	REA, Cod, Vis, Tool, Sum	✓	1,047,576
Phi-4-reasoning	Compact		`llm`	REA, Cod, Tool, Eva	✓	32,000

Table 18: Non-voting executable resources $\mathcal{R}^* \setminus \mathcal{M}^*$ registered through manifest-authorized proxies. Together with the twelve execution-capable voters in Table 17, these yield $|\mathcal{R}^*| = 18$.

Resource ID	SR-FAM	CAPS (declared)	Backing implementation
`code-sandbox-py`	`code_executor`	`coding`, `tool_use`	PyExec v2.4 (containerized)
`retrieval-bm25-dense`	`retriever`	`retrieval`	BM25 + bge-large-en-v1.5
`ocr-engine-paddle`	`ocr_engine`	`ocr`, `visual_grounding`	PaddleOCR 3.0
`img-gen-sdxl`	`diffusion`	`image_generation`	SDXL-base + refiner
`tts-xtts`	`audio_generator`	`audio_synthesis`	Coqui XTTS v2
`form-filler-cd`	`code_llm`	`tool_use`, `summarization`	Constr.-decoding LLaMA-3-8B

The vision column records whether the deployed participant declares `vision` in its modality field. It describes the admitted configuration rather than an exhaustive statement about the underlying model's available interfaces. The seven marked voters are the vision-capable participants considered in §A3.7.

Table 18 lists the six additional non-voting executable resources. They may receive execution roles when Algorithm 4 returns `ELIGIBLE`, but never cast ballots. Their adapters or authorized proxies produce the required protocol messages.

The resource `form-filler-cd` is distinct from the vote-capable Phi-4-reasoning participant. The formatting assignment shown in Figure 5 names Phi-4-reasoning and should not be attributed to this separate non-voting resource. Both illustrate possible specialist execution interfaces, but only the recorded assignment identifies the executor in that trace.

Specialization Registry. After discovery, the runtime publishes the nine-row registry in Table 19. Hard rows impose executor-family constraints; soft rows provide advisory compatibility information without excluding otherwise eligible claimants.

The hard OCR row admits both `ocr_engine` and `vlm`, so it does not by itself prefer the dedicated OCR resource. The hard retrieval row admits only `retriever`. Soft-family matching provides only the limited ranking tiebreaker defined in §A1.11; actual assignments still require votes. In particular, a soft row admitting all listed candidate families does not distinguish specialists from general models.

Registry policy does not add capabilities to a participant's admitted record. In particular, `coding`, `code_execution`, and `tool_use` are distinct identifiers unless an explicit vocabulary mapping is supplied. A role requiring a capability absent from the participant's admitted record fails eligibility even when its executor family matches an SR row.

The `video_generation` and `embodied_action` rows are included for completeness, but the reported pool contains no corresponding executor. Subtasks requiring these capabilities therefore

Table 19: Specialization Registry used in the reported deployment. Hard rows constrain executor-family eligibility; soft rows are advisory and support the tiebreaker in §A1.11. The registry contains nine rows.

capability_class	executor_families	Hardness	Deployment note
image_generation	{diffusion}	HARD	img-gen-sdxl
audio_synthesis	{audio_generator}	HARD	tts-xtts
video_generation	{video_generator}	HARD	No deployed executor
embodied_action	{vla_controller}	HARD	No deployed executor
ocr	{ocr_engine, vlm}	HARD	Dedicated OCR and declared VLM interfaces eligible
retrieval	{retriever}	HARD	retrieval-bm25-dense
code_execution	{code_executor, code_llm}	SOFT	Family guidance; capability checks still apply
tool_use	{code_executor, code_llm, llm, vlm}	SOFT	Multiple executor families admitted
reasoning	{llm, vlm}	SOFT	No specialist preference imposed

Table 20: Additional participants for the sixteen- and twenty-voter pool-size sensitivity configurations in Table 27. The first four extend the main pool to sixteen; the remaining four extend it to twenty. Vision flags describe the reported deployment declarations.

Augmentation	Voter ID	Vis.	Specialty
$\lvert\mathcal{M}^*\rvert = 16$	GPT-4o-mini	✓	general mid
	Gemini 2.0 Flash	✓	general mid
	Claude Haiku 3.5		general mid
	Qwen2.5-72B-Instruct		general mid
$\lvert\mathcal{M}^*\rvert = 20$	DeepSeek-V3		code/reasoning specialist
	Mistral Codestral 25.01		coding specialist
	Llama 3.3 70B		long-context specialist
	Granite 3.1 8B		enterprise/RAG specialist

fail the consistency check in §A1.8. The reported benchmark evaluations do not exercise these capabilities.

Augmentation pool. For the pool-size sensitivity study in Table 27, $|\mathcal{M}^*| = 16$ adds four participants to the main pool, and $|\mathcal{M}^*| = 20$ adds a further four. Table 20 records these additions.

This table specifies the pool-size sensitivity configurations; it does not establish the membership of every committee configuration reported in Appendix A11.7, including the twenty-four-voter setting.

A4.2 Runtime Implementation and Shared-State Layout

On-disk layout. The seven shared-state structures in §A1.2 are backed by runtime-written append-only logs on local NVMe. Each entry contains a length-prefixed UTF-8 JSON payload and its Ed25519 signature. The length prefix is a 64-bit unsigned integer. The per-epoch layout is

```
/run/<run_id>/<epoch>/
    cr.log          # Capability Registry
    sr.log          # Specialization Registry
    vl.log          # Ballots and receipt records
    dr.log          # Decision records and outcomes
    rrt.log         # Role Ranking Table updates
    olt.log         # Output-lock updates
    al.log          # Ordered audit history
```

In-memory hash indexes support expected $O(1)$ key lookup. Registries and tables are active views of the retained event history. Replacing a binding or invalidating an output changes the active view without deleting the historical authorization.

Voters apply authenticated state updates in runtime order. Replicas may temporarily lag because of communication delay; the same committed prefix reconstructs the same coordination state. Append-only organization alone does not establish durable crash recovery, atomicity across files, or exactly-once tool execution. Those require the additional mechanisms discussed in Appendix A11.3.

Message bus and transport. The benchmark configuration uses an in-process Ed25519-authenticated message bus. Its latency measurements therefore primarily reflect protocol and inference costs rather than deployment-network variability. Participant keys are established at process start and checked through the configured admission procedure. Verifying a signature demonstrates control of the corresponding key; identity admission additionally depends on the deployment's trusted registration policy.

A separate localhost gRPC/mTLS transport check reported approximately 0.6 ms of added latency per message. This local check does not establish equivalent performance under wide-area network delays, partitions, or runtime failures.

Inference configuration. The reported inference configuration uses temperature 0, top-$p = 1$, and a maximum of $4{,}096$ output tokens per response. Coordination messages use structured JSON interfaces and are checked against the protocol schemas. Syntactic conformity does not establish semantic correctness, and failed schema checks remain subject to the bounded rejection paths in §A1.15.

Open-weight models are served through vLLM with 4-GPU tensor parallelism on an $8\times$A100 (80 GB) node. API throughput is rate-matched to local serving throughput; this controls request rates but does not imply identical per-call latency across participants.

Tool calls and resource invocations use the authenticated runtime interface. Message kind determines the handler, while sender authorization, active assignment, tool scope, and epoch determine whether the operation is permitted.

A4.3 Baseline Configuration Details

The eight baseline configurations receive the same tools, memory partitions, executable resources, and per-run limits as URCP: 500 K tokens and 600 s wall-clock. Equal limits do not imply equal realized token consumption or runtime. The controlled comparison varies the coordination configuration while disabling adapter-level retries.

Adapter retries. Retry/reconnect wrappers in the evaluated baseline adapters are disabled for the main controlled comparison. Transport failures are therefore exposed to the upper coordination layer rather than absorbed by adapter retries. Table 21 records the configured retry counts and the retry-disabled settings.

These are properties of the evaluated implementations and adapters, not universal retry requirements of communication standards such as A2A, MCP, or ACP. Native-retry comparisons are reported separately in Appendix A11.4.

Transport-baseline coordination. A2A, MCP, and ACP are evaluated with a round-robin assignment wrapper over the available participant union $\mathcal{M}^* \cup \mathcal{R}^*$. Participants belonging to both sets are counted once. The wrapper refreshes assignment when a participant fails. Consequently, these rows evaluate the transport configuration together with this explicit coordination policy.

The remaining baselines retain the reported coordination settings listed in Table 21, apart from the adapter retry change. No additional baseline hyperparameter search is reported.

A4.4 Robustness Failure-Timing Conventions

The perturbations in Table 9 distinguish failures introduced before discovery from failures introduced during an active epoch. This distinction determines the electorate under Assumption A1.4.

Pre-epoch dropout. The "Drop 2/4/8 models" conditions remove voters before discovery. The admitted electorates therefore contain ten, eight, and four voters, respectively. At $q = 3/4$, ordinary decisions require

$$\lceil 0.75 \times 10 \rceil = 8, \qquad \lceil 0.75 \times 8 \rceil = 6, \qquad \lceil 0.75 \times 4 \rceil = 3$$

YES votes.

Table 21: Baseline configurations in the main controlled comparison. Retry counts refer to the evaluated adapters before and after disabling retries. The table describes the tested configurations, not protocol-wide defaults or production-tuned deployments.

System	Adapter retries (configured → controlled)	Coordination configuration
A2A	$3 \to 0$	Round-robin over $\mathcal{M}^* \cup \mathcal{R}^*$; no additional planner
MCP	$3 \to 0$	Round-robin; single tool-server endpoint
ACP	$5 \to 0$	Round-robin; capability handshake at $t = 0$
MetaGen	$2 \to 0$	Meta-prompt $k = 3$; search budget of 50 proposals
AFLOW	$2 \to 0$	MCTS rollout 50; depth limit 6
DynRole	$4 \to 0$	Role-assignment model GPT-4.1, fixed across runs
AgentNet	$3 \to 0$	DAG-routing window 4; peer-vote majority
ReSo	$2 \to 0$	Symbolic-graph search depth 5; LLM-guided expansion

Assignment to a remaining voter excludes that claimant, giving electorates of nine, seven, and three and thresholds of seven, six, and three, respectively. Assignment to a non-voting resource retains the ordinary electorate. Membership is fixed after discovery within each resulting epoch.

Mid-epoch perturbations. The agent-failure, tool-failure, stale-memory, delayed-vote, and missing-acknowledgment conditions introduce faults during execution. An unavailable voter remains a member of $\mathcal{M}^*$; missing ballots become ABSTAIN at the voting deadline. Failure of an executor or tool does not by itself alter voter membership.

For the twelve-voter pool, ordinary decisions and repair require nine YES votes from twelve eligible voters. Assignment to a voter requires nine YES votes from the other eleven; assignment to a non-voting resource requires nine from twelve. Thus nine responsive voters can suffice for an ordinary decision while still being insufficient to assign a role to one of those responsive voters.

Repair cannot bypass these thresholds. A repair proposal is executed only after acceptance by the current electorate. An accepted L_3 repair changes the schema but does not automatically repeat discovery or remove unavailable voters. Participant or resource re-evaluation requires an accepted L_5 repair explicitly returning to discovery, or a separate restart. If repair cannot obtain the required quorum within its bounds, the run terminates as NO-CONSENSUS.

The agent-failure percentages describe the injected experimental conditions; they should not be interpreted as a blanket guarantee of tolerance to the same percentage of unavailable voters. The effect depends on which role instances and voting participants are affected and on the decision-specific electorate.

Communication-stress conditions. Vote delays are injected at the message bus using log-uniform sampling in $[\tau_{\text{vote}}/2, 2\tau_{\text{vote}}]$. A ballot arriving after the deadline is recorded as late and cannot change the closed decision; if no valid on-time ballot exists for that voter, its effective ballot is ABSTAIN.

ACK losses affect 20% of `HANDOFF-RECEIVED` messages. If no valid acknowledgment arrives within the original τ_{handoff} window, the runtime proposes L_2 repair for the affected consumer role. The timeout does not authorize reassignment by itself.

These conditions test explicit completion or failure under the injected faults. They do not guarantee successful completion under every failure pattern.

A4.5 Scope of the Robustness Comparison

Table 9 reports URCP-only perturbation results. Some conditions, such as model dropout and tool failure, can be applied to other systems. Others, such as delayed URCP ballots, missing checkpoint acknowledgments, or output-lock mismatches, refer to protocol-specific events without a direct counterpart in every baseline.

Table 22: GAIA validation accuracy, reported as mean exact-match score over five seeds. The aggregate-score margin over the strongest listed baseline is 9.2 points.

System	Level 1	Level 2	Level 3	Avg.
A2A	38.4	22.1	9.7	23.4
MetaGen	49.7	31.6	14.3	31.9
AgentNet	47.3	29.4	13.6	30.1
ReSo	50.2	32.0	15.1	32.4
URCP	**56.8**	**42.7**	**25.4**	**41.6**
Δ vs. best baseline	$+6.6$	$+10.7$	$+10.3$	$+9.2$

The table therefore characterizes URCP's behavior under its specified failure model rather than establishing comparative robustness against all external systems. Disabling baseline adapter retries also limits what can be concluded about configurations that normally recover through those adapters.

Model and agent failures. A cross-system comparison must distinguish removal of a central planner, loss of an execution agent, and loss of a voting participant. These events can have different consequences even when the same number of processes is affected. A common evaluation would need to specify the affected component, failure timing, available recovery mechanisms, and treatment of budget exhaustion for each system.

Tool failures. Tool-call failures can be injected across systems, but their recovery depends on the tested adapter and coordination policy. URCP proposes local output or assignment repair as appropriate. The absence of URCP-style repair in another system does not by itself establish that the system cannot recover through another mechanism.

Stale state and communication stress. URCP detects stale artifact references through active output locks, dependency records, and epoch checks. Other systems may use different state-validation mechanisms. Similarly, delayed ballots and missing handoff acknowledgments require a system-specific mapping before their effects can be compared meaningfully.

Accordingly, the supported conclusion from Table 9 is that URCP records explicit terminal outcomes under the reported perturbations. The table does not quantify an advantage over production-tuned baselines or demonstrate that those baselines cannot be evaluated under analogous faults.

A4.6 Additional Benchmarks and Multimodal Subsets

GAIA. GAIA [23] evaluates tool use, retrieval, and multi-step reasoning across three difficulty levels. Table 22 reports the supplied validation exact-match scores, averaged over five seeds.

URCP exceeds the strongest listed baseline by 6.6 points at Level 1, 10.7 at Level 2, and 10.3 at Level 3. The largest displayed margin is therefore at Level 2. Using the reported aggregate scores, the overall margin is $41.6 - 32.4 = 9.2$ points.

The aggregate scores are retained as reported. They are not assumed to be the unweighted arithmetic means of the three displayed level scores. Reproducing the aggregate requires the evaluated task counts and aggregation weights.

REALM-Bench. REALM-Bench [8] evaluates real-world planning and coordination tasks. Table 23 groups the reported results into single-step, parallel-coordination, and multi-step families. The largest category margin is on multi-step tasks: $47.8 - 34.5 = 13.3$ points. The reported aggregate margin is $57.8 - 48.0 = 9.8$ points.

Multimodal evaluation subsets. Table 24 reports the evaluation groups identified in the experiment records as UI-grounded τ-bench items and MABench screenshot-reasoning items. The reported groups contain 93 and 148 items, respectively. These labels describe the evaluated groups; exact task identifiers and any preprocessing or added visual inputs are needed to establish their relationship to the original benchmark releases.

Table 23: REALM-Bench results grouped into single-step (families 1–3), parallel-coordination (4–7), and multi-step (8–11) tasks. Values are reported mean weighted task-success scores over five seeds.

System	Single-step	Parallel	Multi-step	Avg.
A2A	51.2	38.7	24.6	38.2
MetaGen	60.3	47.1	32.8	46.7
AgentNet	58.6	46.0	33.9	46.2
ReSo	61.4	48.2	34.5	48.0
URCP	**67.3**	**58.4**	**47.8**	**57.8**
Δ vs. best baseline	$+5.9$	$+10.2$	$+13.3$	$+9.8$

Table 24: Reported multimodal evaluation groups: UI-grounded τ-bench items (93) and MABench screenshot-reasoning items (148). Scores are means over five seeds; Δ compares URCP with the strongest listed baseline in each group.

System	τ-bench (UI-grounded)	MABench (screenshot-reasoning)
A2A	41.8	47.6
MetaGen	50.4	55.3
AgentNet	49.1	53.8
ReSo	51.7	56.0
URCP	**61.4**	**67.8**
Δ vs. best baseline	$+9.7$	$+11.8$

The observed margins are consistent with potential benefits from visual-evidence validation and cross-modal repair, but this table does not isolate their causal contributions or report a separate text-only comparison.

The seven declared vision-capable voters in Table 17 can execute visual-evidence roles. When an accepted schema includes the gatekeeping pattern described in §A3.7, a vision-capable role processes the image and supplies structured evidence to downstream text-only roles. This is an observed schema pattern, not a requirement imposed on every multimodal workflow by the core protocol.

A5 Emergent-Phenomena Trace Analysis: Details

This appendix documents the trace-analysis pipeline used to produce Tables 10–13 and Figures 3–9.

Trace extraction. For each run we replay the audit log AL (§A1.2) in the order of its append-only entries, reconstructing the per-epoch state of the Capability Registry CR, Specialization Registry SR, Vote Ledger VL, Decision Registry DR, Role Ranking Table RRT, and Output Lock Table OLT. Because AL is signed and append-only by construction, the reconstruction is deterministic and idempotent. Replay yields, for each run: the locked role schema $\mathcal{S}$ (the `ROLE-SCHEMA-PROPOSE` whose `VOTE-OUTCOME` carries `verdict=`ACCEPT in epoch e); the assignment map $\mathcal{A}$ (the `VOTE-OUTCOME` entries with kind=`ASSIGNMENT` and `verdict=`ACCEPT, one per role); the execution DAG $\mathcal{O}$ from $\mathcal{S}.\mathrm{dag_edges}$; the per-decision vote record from $\mathrm{VL}[(e, d)]$; and the output-and-handoff record from OLT together with the matching `HANDOFF-RECEIVED` entries.

Per-voter quantities. The four per-voter quantities reported in the body are computed from VL and DR as follows. Letting Prop_i be the set of decisions whose `ROLE-SCHEMA-PROPOSE` (or other proposal type) was sent by voter ℓ_i, the proposer-win rate is $p_i = |\{d \in \mathrm{Prop}_i : \mathrm{DR}[(e,d)].\mathrm{verdict}=\text{ACCEPT}\}|/|\mathrm{Prop}_i|$. The execution-role share e_i is the fraction of $|\mathcal{S}|$ for which $\mathcal{A}(s_j) = \ell_i$, averaged across runs and weighted equally per run. The participation rate π_i is the fraction of runs in which ℓ_i is in the image of $\mathcal{A}$. The critical-path share c_i uses the longest path $\Pi^*(\mathcal{O})$ (computed by topological sort over $\mathcal{O}$, ties broken by lexicographic role name) and reports $|\{s_j \in \Pi^*(\mathcal{O}) : \mathcal{A}(s_j) = \ell_i\}|/|\Pi^*(\mathcal{O})|$. Bootstrap confidence intervals (10,000 resamples) are computed at the run level.

Role-type classification (Tables 11, 14). Role types are inferred deterministically from the locked role tuple $s_j = (\mathrm{name}_j, \mathrm{covers}_j, \mathrm{caps}_j, \mathrm{in}_j, \mathrm{out}_j, \mathrm{tools}_j, \mathrm{mem}_j, \tau_{\mathrm{exec},j}, \mathrm{accept_crit}_j)$ using a fixed rule table. We classify a role as a *critic-style* role iff (i) name_j contains any of the substrings `check`, `verif`, `review`, `audit`, `reconcile`, `validate`; (ii) $\mathrm{caps}_j \cap \{\texttt{evaluation}, \texttt{reasoning}\} \neq \emptyset$; and (iii) in_j contains at least one type that also appears as the out_j of an upstream non-critic role (i.e., the critic inspects another role's output). Other classes (planner, final assembler, code/tool executor, researcher, summarizer, evidence extractor) are defined analogously by name-substring lists keyed to caps_j predicates; the full table is shipped with the release. We did not use the role's `func` description for any classification, since the corrected role tuple of §3 contains no `func` field.

Per-task-family decompositions. The body's per-tier and per-role-type tables aggregate over benchmarks. The release accompanying this paper includes per-task-family decompositions for each table in this section, broken down by: MABench's six task families (planning, retrieval-and-reasoning, tool-chains, multi-step QA, code-and-execution, screenshot-reasoning), τ-bench's four domains (retail, airline, travel, support), and MemoryAgentBench's four phases (retrieval, test-time learning, long-context, conflict resolution). Within each cell we report mean $\pm$ 95% bootstrap CI and the per-cell sample size, since some cells (e.g., screenshot-reasoning $\times$ Compact tier) have small counts that the aggregate numbers in the body conceal.

Reproducibility. A reference implementation of the trace-analysis pipeline reads the on-disk audit log produced by the runtime and produces every table in §A3 together with raw per-run JSON. The code path is deterministic given the input log: any two reviewers running the pipeline against the same log will produce byte-identical outputs.

A6 Secure-Handoff Compatibility Condition and Possible Extensions

This appendix expands on the compatibility caveat in §A3.6. Encrypted handoffs preserve the intended checkpoint semantics only when voters have sufficient verifiable evidence to evaluate the locked acceptance criteria. Encryption does not change the voting threshold, but it can limit what an affirmative vote establishes.

Compatibility condition. Let x_j denote the plaintext output of role s_j, and let E_j^{pub} denote the public evidence accompanying its encrypted handoff. Compatibility requires a public verification procedure whose verdict establishes the same acceptance predicate $\mathrm{accept_crit}_j(x_j)$ that the checkpoint would evaluate with access to the plaintext. The evidence must also be verifiably bound to the particular output and handoff under consideration. For a cryptographic proof, this requirement is subject to the proof system's stated completeness and soundness guarantees.

A redacted summary, a producer-supplied schema-compliance assertion, and a cryptographic digest do not by themselves establish this condition. A digest can support an integrity check when the corresponding content is available, but it does not establish that hidden content satisfies the acceptance criteria. Likewise, checking the format of an assertion does not establish its truth.

If a role's locked acceptance criteria concern only public fields, voters can evaluate those fields directly. The resulting checkpoint certifies those public properties; it does not additionally certify uninspected plaintext. Consequently, the presence of public-evidence fields in an encrypted-handoff trace is insufficient to demonstrate equivalence to plaintext evaluation. When a criterion requires hidden content, the protocol needs authorized disclosure or suitable verifiable evidence. Otherwise, the checkpoint cannot establish that criterion, even if an affirmative supermajority is obtained.

Extensions. The following extensions address different parts of this requirement. They are possible additions to the protocol, not claims of implemented or experimentally validated functionality.

Schema-level annotation. An optional `public_evidence_schema` field could specify the fields required in the public evidence for each role. The locked schema should also identify which acceptance criteria those fields support and which criteria require disclosure or additional proof. At Stage 6 (§A1.12), the runtime could mechanically reject an encrypted `SUBTASK-COMPLETE` that lacks required fields, using reason `ENCRYPTION-WITHOUT-PUBLIC-EVIDENCE`.

This extension checks evidence structure and completeness relative to the declared schema. It does not establish that the evidence is truthful or sufficient to prove properties of the hidden output. The substantive checkpoint decision remains subject to Φ_q.

Selective disclosure or zero-knowledge proofs. Selective disclosure could reveal the portions of an output needed to evaluate the locked criteria, together with a verifiable binding to the underlying output. This supports only criteria that can be evaluated from the disclosed information.

Alternatively, a zero-knowledge proof could establish a formally specified predicate over a hidden output. Its public statement would need to bind the predicate, the output commitment, and the relevant handoff context. The construction must also ensure that the output proved about is the output delivered to the recipient; a proof about an unrelated commitment is insufficient.

Such proofs apply to criteria that can be encoded in the chosen proof system. They do not automatically capture open-ended judgments about relevance, factual correctness, or task quality. We do not specify a proof system or claim practical performance for these extensions. Their feasibility requires a concrete construction and evaluation of proof generation, verification, and communication costs.

Under either extension, checkpoint acceptance remains a Φ_q decision. The output-locking table OLT and audit log AL must retain the binding between the accepted decision, its public evidence, and the associated encrypted artifact, without requiring public storage of the plaintext.

Repair semantics and nonce management. An L_1 repair does not require deterministic reproduction of the ciphertext. It requires an unambiguous association between each output version, its evidence, and its checkpoint decision.

For authenticated encryption schemes requiring nonce uniqueness, each new encryption operation must use a nonce that is unique under the same encryption key. In particular, deriving a nonce solely from a role-claim hash and upstream-output hashes is insufficient when those values can recur across retries or revised outputs. Retransmitting an existing ciphertext is distinct from performing a new encryption operation.

A revised output requires evidence bound to the revised artifact and a fresh checkpoint decision. Re-encrypting unchanged plaintext may preserve evidence about its content, but requires updated delivery metadata and a verifiable association with the new ciphertext. Neither case permits stale messages or invalidated checkpoint decisions to bypass the epoch and repair rules in §A1.14. The specification therefore requires explicit output versioning and nonce management; it does not rely on deterministic ciphertext reproduction.

Cross-recipient re-routing and availability. During L_2 repair, the runtime presents fallback candidates from RRT, and a replacement binding requires authorization under the assignment rule (§A1.11). If the replacement recipient lacks access to the existing encrypted payload, delivery requires re-encryption or an explicitly supported key-management mechanism. Encryption keys must be authenticated for their intended use; an identity-signing key is not automatically an encryption key.

Re-encryption is possible only while an authorized party retains access to the plaintext or the necessary decryption material. If the producer is unavailable and no authorized recovery mechanism exists, eagerly requesting re-encryption cannot restore access. This is an availability limitation rather than, by itself, a failure of voting or proof soundness.

A deployment supporting encrypted handoffs must therefore specify retention, key recovery, and access policies, or report a typed handoff failure when delivery cannot be restored. Further repair remains subject to Φ_q and the protocol's retry and epoch limits. Designing recovery mechanisms that preserve confidentiality while supporting recipient changes remains an open extension.

A7 Adversarial Robustness and Sensitivity Analysis

The robustness experiments in §4.4 stress URCP under *honest* failures: model dropout, agent termination, tool errors, and communication delays. A coordination protocol that requires supermajority approval invites a different class of question: what happens when the obstruction is not a fault but a deliberate strategy? And how brittle is the protocol to the timeout and budget parameters chosen on the validation split? This section addresses both concerns. We evaluate URCP under five

classes of adversarial voting behavior (§A7.1), sweep the voting threshold from simple majority through unanimity to test whether the design's central choice is well-placed (§A7.2), examine the protocol's sensitivity to its five most consequential hyperparameters (§A7.3), and finally measure the effectiveness of each repair level individually rather than only the aggregate repair mechanism (§A9.7).

The framing throughout is conservative: we do not claim that URCP magically succeeds under coordinated supermajority obstruction, nor that its hyperparameters are universally optimal. We claim only what the protocol's formal properties already promise—explicit termination with an audit trail (Appendix A1.16)—and test whether those properties hold empirically when the assumptions of Appendix A1.1 are stressed.

A7.1 Adversarial Voting Behavior

URCP's central design choice is the q-supermajority decision rule Φ_q at $q{=}3/4$ (§3.2). With $|\mathcal{M}^*|{=}12$, ordinary protocol decisions require $\lceil 0.75 \times 12 \rceil = 9$ YES votes to accept and 9 NO votes to reject. Thus, for decisions whose denominator is the full voter set, the rule can absorb up to three persistent dissenters if all remaining nine voters affirm the proposal. Role assignment is stricter because the claimant's own vote is excluded: the assignment denominator is $|\mathcal{M}^*| - 1 = 11$ and the threshold is $\lceil 0.75 \times 11 \rceil = 9$. Consequently, an honest model claimant can tolerate at most two persistent NO voters among the remaining voters; with three such voters, the maximum possible affirmative count is only eight. This distinction is important: coordinated obstruction by three voters may still permit ordinary decisions, but it can block model-role assignment under the core protocol.

The purpose of this stress test is therefore not to claim that URCP can always route around coordinated vetoes. Instead, we ask what shape the failure takes: whether the protocol hangs, silently accepts a corrupted state, or terminates with an explicit verdict and an auditable log. The expected behavior is that minority obstruction is absorbed while the relevant supermajority threshold remains reachable; once the threshold becomes structurally unreachable, URCP should terminate as NO-CONSENSUS rather than forcing an unsupported output.

Adversary classes. We instantiate five classes of adversarial participants by replacing one or more voters in $\mathcal{M}^*$ with scripted voters that bypass model inference and emit a fixed voting behavior. The adversaries still pass discovery (Appendix A1.3) and context verification (Appendix A1.5); Assumption A1.1's signature check is not what is being attacked. The adversaries obstruct only the voting layer (Appendix A1.6). The classes are:

- ALWAYS-NO: votes NO on every decision, with a generic justification.
- BOUNDED-REVISE: votes REVISE until its proposed syntactically valid but minimal modification is incorporated into the next-round proposal or the local revision budget K_ℓ is exhausted, after which it votes YES. This adversary delays progress by inflating revision rounds without making acceptance structurally impossible.
- STRATEGIC-LATE: votes YES, but only at $t = \tau_{\text{vote}} - 0.5\,\text{s}$, maximizing latency while remaining within the voting deadline.
- RANDOM-NOISY: votes uniformly at random over $\{\text{YES}, \text{NO}, \text{REVISE}\}$ on every decision.
- INCONSISTENT: votes YES in odd-numbered rounds and NO in even-numbered rounds, simulating a model whose judgment flips between revisions.

Threshold regimes. For full-denominator decisions, such as framing, schema formation, checkpoint validation, final completion, and repair votes, three ALWAYS-NO voters leave exactly nine possible YES votes and can therefore be absorbed with no slack. For assignment decisions involving a model claimant, however, the claimant is removed from the denominator. With three ALWAYS-NO voters, the remaining eligible electorate contains only eight cooperative voters, so the assignment threshold of nine cannot be reached. Therefore, the end-to-end protocol can absorb one or two persistent veto voters across all stages, but three or more persistent veto voters are sufficient to force NO-CONSENSUS at the first required honest model-role assignment. This is not a deadlock: it is the intended explicit failure mode when the ratification threshold cannot be reached.

Results. Table 25 reports outcomes on MABench with the heterogeneous twelve-voter team (Appendix A4.1). The most important column is DFT: deadlock-free termination remains at 100%

Table 25: Adversarial voting stress test on MABench with the heterogeneous 12-voter team under Φ_q at $q{=}3/4$. Mean over 5 seeds. NOC. = NO-CONSENSUS rate; COMPL. = completion rate; DFT = deadlock-free termination. For ordinary full-denominator decisions, three ALWAYS-NO voters leave exactly nine possible YES votes. For model-role assignment, however, the claimant's self-vote is excluded, so three ALWAYS-NO voters leave at most eight possible YES votes and make acceptance structurally impossible. Rows marked "struct." are therefore core-URCP NO-CONSENSUS outcomes, not implementation failures. Cost is per run.

Condition	Task	NoC.	Compl.	Avg. rounds	Cost	DFT
Benign URCP (reference)	72.1	2.1%	97.9%	1.4	$1.21	100%
ALWAYS-NO, $k=1$	71.4	3.2%	96.4%	1.7	$1.32	100%
ALWAYS-NO, $k=2$	69.6	5.1%	94.2%	2.0	$1.43	100%
ALWAYS-NO, $k=3$ (struct.)	–	100.0%	0.0%	3.1	$1.58	100%
ALWAYS-NO, $k=4$ (struct.)	–	100.0%	0.0%	2.8	$1.54	100%
BOUNDED-REVISE, $k=1$	70.2	4.1%	95.1%	2.1	$1.41	100%
BOUNDED-REVISE, $k=3$	62.4	14.6%	82.7%	3.4	$1.79	100%
STRATEGIC-LATE, $k=1$	70.8	2.7%	96.8%	1.5	$1.27	100%
STRATEGIC-LATE, $k=3$	68.4	3.4%	95.7%	1.8	$1.36	100%
RANDOM-NOISY, $k=1$	69.4	4.7%	94.2%	2.2	$1.42	100%
RANDOM-NOISY, $k=3$	58.7	17.6%	79.4%	3.5	$1.78	100%
INCONSISTENT, $k=2$	67.3	6.9%	92.1%	2.4	$1.51	100%

across every condition, including those where the supermajority threshold is structurally unreachable. Task performance degrades as obstruction increases. For ALWAYS-NO adversaries, the end-to-end structural break occurs at $k{=}3$, not $k{=}4$, because Stage 5 assignment uses the self-vote-exclusion denominator. Rows marked "struct." indicate this mathematically forced NO-CONSENSUS behavior under the core protocol.

The qualitative pattern is consistent with the protocol arithmetic. With one or two ALWAYS-NO voters, URCP usually completes: the remaining cooperative voters can still clear both the ordinary nine-vote threshold and the self-excluded assignment threshold. With three ALWAYS-NO voters, ordinary full-denominator decisions can still pass if all nine cooperative voters agree, but honest model-role assignment cannot pass because the claimant is excluded and only eight cooperative voters remain. Thus the core protocol terminates in NO-CONSENSUS, preserving the complete vote log rather than assigning a role without supermajority authorization. With four ALWAYS-NO voters, even ordinary full-denominator decisions cannot clear the threshold. These rows therefore measure the safety boundary of the core protocol rather than a recoverable performance regime.

The other adversary classes do not create the same structural impossibility. BOUNDED-REVISE is recoverable in most cases because the proposer can incorporate the requested revision and resubmit; once incorporated, the adversary's rule yields a YES vote, and the round limit K_ℓ bounds total exposure. STRATEGIC-LATE mainly increases latency because the votes still arrive before τ_{vote}. RANDOM-NOISY sits between the extremes: occasional clusters of NO and REVISE votes prevent acceptance on some runs, but enough proposals still receive nine affirmative votes to keep completion above 79% with three noisy voters.

Figure 10 shows the degradation curve for the three non-trivial adversary types as the number of adversarial voters k grows. The ALWAYS-NO curve is stable through $k{=}2$ for end-to-end completion and then collapses at $k{=}3$, where the self-vote-excluded assignment threshold becomes unreachable for an honest model claimant. BOUNDED-REVISE and RANDOM-NOISY degrade more gradually because they do not always remove the same voters from the affirmative coalition on every decision. This shape identifies which failures are absorbed by the supermajority margin and which are surfaced as explicit NO-CONSENSUS outcomes.

Trace example. To illustrate that obstruction terminates explicitly rather than silently, Log 11 shows the final transitions of a run with three ALWAYS-NO adversaries. The framing and schema decisions can still pass when the nine cooperative voters agree, but the first honest model-role assignment cannot pass: the claimant is excluded from the denominator, leaving eight cooperative voters and three persistent NO voters. The assignment decision therefore remains below the acceptance threshold until the assignment revision budget is exhausted, after which the protocol emits a structured

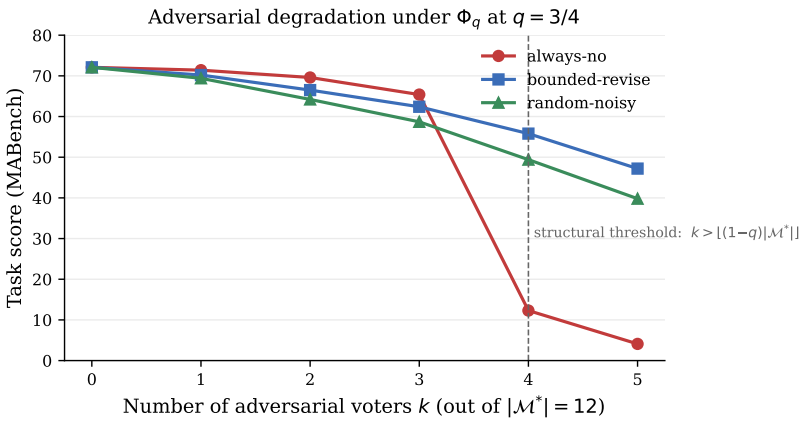


Figure 10: Task-score degradation under three adversary classes as the number of adversarial voters k grows, with Φ_q at $q{=}3/4$. The ALWAYS-NO curve is stable through $k{=}2$ for end-to-end completion and drops at $k{=}3$, because model-role assignment excludes the claimant's own vote and therefore cannot reach the required nine YES votes when three other voters always vote NO. For ordinary full-denominator decisions, the corresponding break point is $k{=}4$. BOUNDED-REVISE and RANDOM-NOISY adversaries degrade more gradually because some revised or noisy proposal rounds still receive nine or more affirmative votes. Deadlock-free termination (Appendix A1.16) remains at 100% in all cases: when the threshold cannot be reached, URCP terminates with explicit NO-CONSENSUS rather than hanging or forcing an unsupported decision.

NO-CONSENSUS verdict with the full proposal-and-vote history preserved in the audit log AL (Appendix A1.2).

```
// Adversaries: { mistral-medium-3, command-a, gpt-4.1-mini }
// q = 3/4, |M*| = 12.
// Ordinary decisions: threshold = ceil(0.75 * 12) = 9 YES.
// Assignment decisions for a model claimant:
//     denominator = |M*| - 1 = 11,
//     threshold = ceil(0.75 * 11) = 9 YES.
//
// Framing and schema can still pass when the nine cooperative voters agree.
// The first honest model-role assignment cannot pass because the claimant
// is excluded, leaving only 8 cooperative voters and 3 always-no voters.
//
// Stage 5: assignment for role final_assembler.
// Claimant: claude-opus-4.
// Eligible voters for this assignment: M* \ {claude-opus-4}.
// Tally per attempt: 8 YES, 3 NO, 0 REVISE, 0 ABSTAIN.
// Since YES=8 < threshold=9 and NO=3 < threshold=9, Phi_q = REVISE.
//
// Attempt 1: claude-opus-4 claims final_assembler.
//            Tally: 8 YES, 3 NO, 0 REVISE. Phi_q = REVISE.
// Attempt 2: gpt-4.1 claims final_assembler.
//            Tally: 8 YES, 3 NO, 0 REVISE. Phi_q = REVISE.
// Attempt 3: gemini-2.5-pro claims final_assembler.
//            Tally: 8 YES, 3 NO, 0 REVISE. Phi_q = REVISE.
// Attempt 4: deepseek-r1 claims final_assembler.
//            Tally: 8 YES, 3 NO, 0 REVISE. Phi_q = REVISE.
// Attempt 5: claude-sonnet-4 claims final_assembler.
//            Tally: 8 YES, 3 NO, 0 REVISE. Phi_q = REVISE.
// K_5 = 5 assignment budget exhausted.

{ "type": "STAGE-TERMINATION",
  "stage": "S5.assignment",
```

```
  "role_id": "final_assembler",
  "verdict": "NO-CONSENSUS",
  "epoch": 1,
  "rounds_used": 5,
  "K_budget": 5,
  "termination_cause": "ASSIGNMENT_THRESHOLD_UNREACHABLE",
  "decision_summary": {
    "denominator": 11,
    "threshold": 9,
    "max_possible_yes": 8,
    "persistent_no_voters": [
      "mistral-medium-3",
      "command-a",
      "gpt-4.1-mini"
    ]
  },
  "assignment_log": [
    { "round":1, "claimant":"claude-opus-4",
      "yes":8, "no":3, "revise":0, "abstain":0,
      "verdict":"REVISE" },
    { "round":2, "claimant":"gpt-4.1",
      "yes":8, "no":3, "revise":0, "abstain":0,
      "verdict":"REVISE" },
    { "round":3, "claimant":"gemini-2.5-pro",
      "yes":8, "no":3, "revise":0, "abstain":0,
      "verdict":"REVISE" }
    /* rounds 4 and 5 truncated for brevity */ ],
  "audit_trail_hash": "sha256:f08c2ab3d971...4e7a",
  "deadlock_free": true,
  "timestamp": "[redacted for blind review]" }

// Stage-level verdict: NO-CONSENSUS. The protocol terminates without
// assigning final_assembler and without delivering an output. All votes,
// justifications, proposal hashes, and assignment attempts are preserved
// in the audit log AL (App. app:state); the full per-round VOTE-OUTCOME
// entries are recoverable from the Decision Registry DR (App. app:ledger).
```

Figure 11: Explicit NO-CONSENSUS termination under three ALWAYS-NO voters during model-role assignment (MABench task `mab_tier2_044`, epoch 1). With $|\mathcal{M}^*|=12$ and $q=3/4$, ordinary decisions require nine YES votes. For assignment decisions, however, the claimant is excluded, so the denominator is 11 and the threshold is still 9. When three voters issue NO on every assignment decision, an honest claimant can receive at most eight affirmative votes. Each attempted assignment therefore resolves to REVISE rather than ACCEPT. Once the assignment budget $K_5=5$ is exhausted, the protocol emits a structured NO-CONSENSUS verdict and terminates without output. Every proposal hash, every voter's verdict, and every justification is recoverable from the audit log AL alone (Appendix A1.2). The 100% DFT entry in Table 25 corresponds to terminations of exactly this form.

The audit log preserves every assignment attempt and the identity of the dissenting voters. A downstream system can therefore detect the obstruction pattern, escalate to a human operator, restart with a different deployment policy, or record that the team did not reach supermajority agreement. The important safety property is that URCP does not silently lower the threshold, bind an unsupported claimant, or fabricate a final output when the required supermajority is unreachable.

A7.2 Voting-Threshold Sensitivity

URCP's protocol-level commitment is to a q-supermajority decision rule with $q=3/4$ as the recommended operating point (§3.2). Two natural anchor points are simple majority and strict unanimity: lowering q to majority costs 4.8 task points on MABench, while raising q to unanimity yields a statistically insignificant $+0.2$ (Table 26, rows 1 and 5 versus the default). These anchors are informative but do not address the broader question: how does performance behave across the full range $q \in (1/2, 1]$? We answer by parameterizing Φ_q across five operating points and reporting the cost–quality frontier.

Table 26: Sensitivity to voting threshold q on MABench (heterogeneous team, mean over 5 seeds). $q{=}3/4$ corresponds to URCP's default operating point, the row whose numbers match the main experiments (Table 3). $q{=}1$ recovers strict unanimity.

Voting rule	Task	Handoff fail	NoC.	Avg. rounds	Cost	Repair rate
Majority ($q{>}1/2$)	67.3	13.1%	0.5%	1.0	\$0.92	11.7%
Supermajority ($q{=}2/3$)	70.2	9.8%	0.9%	1.1	\$1.06	14.6%
$q{=}3/4$ **(default)**	**72.1**	**6.4%**	**2.1%**	**1.4**	**\$1.21**	**18.7%**
Near-unanimity ($q{=}0.9$)	72.4	5.4%	3.6%	1.6	\$1.34	21.2%
Unanimity ($q{=}1$)	72.3	4.7%	6.8%	1.9	\$1.51	25.4%

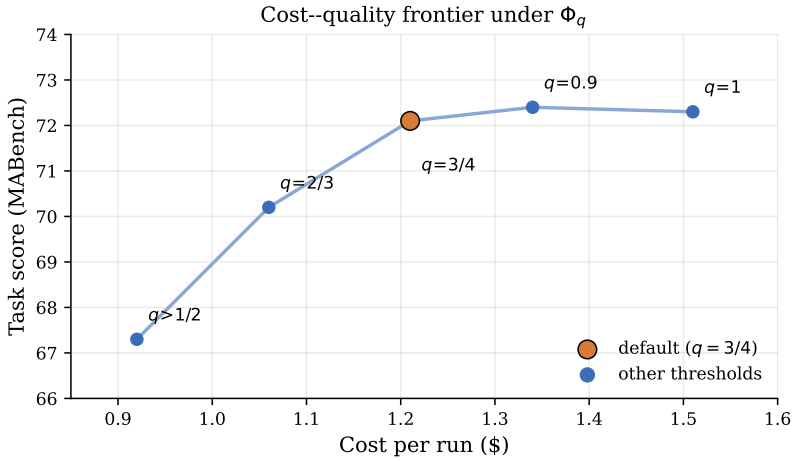


Figure 12: Cost–quality frontier under Φ_q as q varies in $(1/2, 1]$. Each point corresponds to one row of Table 26. The default $q{=}3/4$ sits near the elbow of the curve: most of the unanimity-level handoff quality is already captured at substantially lower cost.

Two opposing trends are visible in Table 26. Increasing the threshold improves *handoff quality*: the handoff failure rate falls monotonically from 13.1% at majority to 4.7% at unanimity, reflecting that stricter approval catches errors that looser rules miss. Increasing the threshold also raises the *coordination overhead*: rounds, cost, and NO-CONSENSUS rate all grow, because stricter approval requires more revision and more proposers find their proposals blocked by a small dissenting subset. Task score is non-monotone: it rises from majority to $q{=}0.9$ and then plateaus or slightly drops at strict unanimity, where the additional NO-CONSENSUS terminations begin to outweigh the marginal handoff-quality gains. The default $q{=}3/4$ captures most of the handoff-quality benefit (-3.2% closer to unanimity, vs. $+5.8\%$ farther from majority) at 80% of unanimity's cost.

Figure 12 plots the cost–quality frontier explicitly. The shape suggests a practical takeaway: $q{=}3/4$ sits near the elbow of the cost–quality curve, balancing handoff reliability against coordination overhead. Settings that demand the strongest possible auditability—enterprise pipelines, safety-critical workflows, multi-vendor deployments—may justify moving to $q{=}0.9$ or $q{=}1$; settings where catastrophic handoff failures are rare and recoverable may move down to $q{=}2/3$. The mechanism of §3.2 is general enough to support a range of operating points without architectural change; only the constant q in the threshold computation $\lceil q\,|\mathcal{M}^*|\rceil$ (or, for assignment decisions, $\lceil q\,(|\mathcal{M}^*|-1)\rceil$ per Appendix A1.10) differs.

A7.3 Protocol Hyperparameter Sensitivity

The hyperparameters in Table 2 (revision-round budget K_ℓ, vote timeout τ_{vote}, global epoch cap $E_{\max}$, and pool size $|\mathcal{M}^*|$) were chosen on a held-out validation split of MABench and held fixed across all conditions. We test whether URCP is robust to these choices by varying one parameter at a time around the defaults and measuring task score, NO-CONSENSUS rate, and run cost. Pool size is the only parameter that also varies the diversity of the model team; for $|\mathcal{M}^*| \le 12$ we sample subsets

Table 27: One-at-a-time sensitivity of URCP on MABench. Defaults are underlined and match the values fixed in Table 2. Each cell reports task score / NO-CONSENSUS rate / cost per run, mean over 3 seeds. "—" marks values not swept.

Parameter	Values swept					
K_ℓ (revision rounds)	$K_\ell = 1$	$K_\ell = 3$	$\underline{K_\ell = 5}$	$K_\ell = 7$	$K_\ell = 10$	—
	62.4 / 11.8% / \$0.91	70.6 / 3.4% / \$1.10	72.1 / 2.1% / \$1.21	72.3 / 1.7% / \$1.34	72.4 / 1.5% / \$1.51	—
τ_{vote} (s)	15s	30s	45s	90s	180s	—
	64.8 / 18.4% / \$1.04	70.9 / 4.2% / \$1.16	72.1 / 2.1% / \$1.21	72.1 / 2.0% / \$1.27	72.1 / 2.0% / \$1.34	—
$E_{\max}$ (epochs)	1	3	5	7	11	20
	67.2 / 9.1% / \$1.07	71.0 / 3.4% / \$1.16	71.8 / 2.4% / \$1.20	72.0 / 2.2% / \$1.21	72.1 / 2.1% / \$1.21	72.1 / 2.1% / \$1.22
$\lvert\mathcal{M}^*\rvert$ (pool size)	3	6	9	12	16	20
	62.8 / 1.1% / \$0.31	68.4 / 1.4% / \$0.62	70.7 / 1.7% / \$0.93	72.1 / 2.1% / \$1.21	71.4 / 4.2% / \$1.68	70.2 / 7.8% / \$2.18

uniformly from the twelve-model pool of Appendix A4.1 with the constraint that at least one Frontier model is present. For $|\mathcal{M}^*| > 12$ we augment the original pool with additional contemporary models drawn from a held-out validation set used only for this sensitivity study and not for any comparison in the main results; the exact augmentation list is given in Appendix A4.1.[3]

Three findings emerge from Table 27 and the accompanying curves in Figure 13. First, three of the four parameters exhibit a clear plateau above the chosen default: $K_\ell \geq 5$, $\tau_{\text{vote}} \geq 45\,\text{s}$, and $E_{\max} \geq 7$ all yield essentially the same task score within noise. This is reassuring—it suggests the defaults sit in the flat region of each parameter's response surface rather than at a sharp peak that might indicate overtuning. Second, very small values of K_ℓ or $E_{\max}$ degrade performance noticeably because the protocol is denied the revision and repair budget it relies on; the $K_\ell{=}1$ point (62.4) and $E_{\max}{=}1$ point (67.2) effectively measure URCP without its soft-disagreement and repair mechanisms (Appendix A1.14), respectively, and bracket the -6.3 all-repair-disabled drop reported in Table 35.

The third and most interesting finding concerns pool size $|\mathcal{M}^*|$. Unlike the other parameters, it is non-monotone: task score peaks at $|\mathcal{M}^*|{=}12$ and falls in both directions. Smaller pools ($|\mathcal{M}^*|{=}3, 6$) lack diversity and cannot exploit the capability complementarity documented in §A3.1. Larger pools ($|\mathcal{M}^*|{=}16, 20$) make the Φ_q threshold progressively harder to reach: the NO-CONSENSUS rate climbs from 2.1% at $|\mathcal{M}^*|{=}12$ to 7.8% at $|\mathcal{M}^*|{=}20$, and cost grows roughly linearly while quality decreases. This is the diversity–consensus trade-off in its purest form, and it identifies $|\mathcal{M}^*|{=}12$ as a sweet spot for the model families tested. We make no claim that $|\mathcal{M}^*|{=}12$ is universal; the optimal pool size will depend on the capability spread of the available models, the threshold q, and the difficulty distribution of the target tasks.

A8 Protocol Overhead and Complexity Analysis

A natural question for any coordination protocol is what it costs to run. URCP's seven-stage pipeline involves multiple proposal and voting rounds before any subtask work begins, and the repair hierarchy can in principle restart entire stages. This appendix quantifies the overhead along four dimensions: asymptotic message complexity per stage, computational complexity of the runtime and voter operations, empirical wall-clock and token cost at the operating point used in the experiments, and scaling behavior as the voter set $|\mathcal{M}^*|$ and the execution DAG depth d grow. Throughout the analysis we separate the *coordination layer* (proposals, votes, broadcasts, and shared-state maintenance) from the *execution layer* (the subtask work performed by assigned agents), since these have fundamentally different cost structures.

Reproducibility note. The empirical numbers in Tables 29–32 are computed directly from the audit logs AL produced by the runtime during the same benchmark runs reported in the main experiments. Message counts come from the total number of signed messages appended to AL for each run; coor-

[3]Briefly: the $|\mathcal{M}^*|{=}16$ augmentation adds four mid-tier general-purpose models, and the $|\mathcal{M}^*|{=}20$ augmentation adds four further specialist models, for a total of eight additional candidates beyond the twelve-model main pool. None of these augmented configurations is used in any other table or figure in the paper. The augmentation list, including each added model's declared capabilities and modalities, is consolidated alongside the main pool declaration in Appendix A4.1.

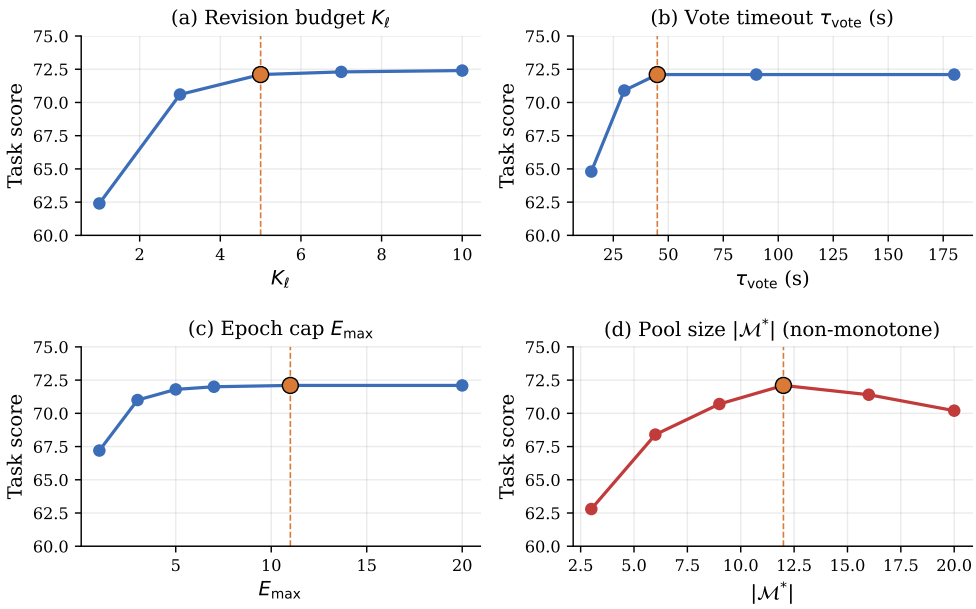


Figure 13: One-at-a-time sensitivity curves for the four most consequential hyperparameters of Table 2. Three of the four (K_ℓ, τ_{vote}, $E_{\max}$) are saturating: performance plateaus at or below the chosen default. Pool size $|\mathcal{M}^*|$ is non-monotone, peaking at $|\mathcal{M}^*|$=12: smaller pools limit diversity, larger pools make the supermajority threshold $\lceil q\,|\mathcal{M}^*| \rceil$ progressively harder to reach.

dination tokens are the sum of token counts on `*-PROPOSE`, `*-VOTE`, `VOTE-OUTCOME`, `ROLE-CLAIM`, and `REOPEN` messages; execution tokens are the sum of token counts on `SUBTASK-COMPLETE` message bodies plus internal agent-side inference traces; latencies are wall-clock differences between message timestamps. The trace-analysis pipeline that produces these aggregates is the same one used for the emergent-phenomena tables and is deterministic given an input log.

A8.1 Asymptotic Message Complexity

Let $|\mathcal{M}^*| = m$ denote the voter set size, $|\mathcal{S}| = n$ the number of roles in the accepted schema, d the depth of the execution DAG, E the number of edges in the DAG, and K the maximum revision-round budget across stages. Each stage contributes the worst-case message counts shown in Table 28.

The dominant cost is the assignment stage, where each candidate for each role triggers a single-claimant vote requiring messages from the remaining $m - 1$ voters under the non-self threshold. Aggregated across all stages and including up to $E_{\max}$ repair epochs, the worst-case total message complexity is

$$C_{\text{msg}}^{\text{worst}} \;=\; O\big(E_{\max} \cdot K \cdot n \cdot m^2\big).$$

This bound is loose: it counts every stage at its worst-case revision budget within every repair epoch. Empirical message counts (reported below) are substantially lower because most stages converge in one or two rounds and repair epochs beyond the first are rare.

A8.2 Computational Complexity

The runtime's per-message work is dominated by signature verification ($O(1)$ per message under Ed25519), schema validation against the message format ($O(|\text{message}|)$), and append-only log writes ($O(1)$ amortized). Crucially, the runtime never performs semantic work; all reasoning happens at the voters.

Voter-side computation is dominated by the cost of generating each proposal or vote, which corresponds to one model inference call per emitted message. Letting θ denote the cost of one model forward pass producing a structured JSON message, and noting that the number of decisions a voter

Table 28: Asymptotic message complexity per stage. $m = |\mathcal{M}^*|$, $n = |\mathcal{S}|$, d = DAG depth, E = DAG edges, K = max revision rounds, $|\mathcal{V}_{\text{probe}}|$ = number of probed capabilities. The dominant term is the per-claimant assignment vote.

Stage	Worst-case messages	Dominant cost
S1 Discovery	$O(m \cdot \lvert\mathcal{V}_{\text{probe}}\rvert)$	Capability probe rounds
S1b Context verify	$O(m)$	One hash broadcast per voter
S2 Framing	$O(K \cdot m^2)$	Per-round vote broadcasts
S3 Subtasks	$O(K \cdot m \cdot n)$	Per-candidate voting
S4 Schema	$O(K \cdot m^2)$	Atomic schema vote per round
S5 Assignment	$O(n \cdot m^2)$	Per-claimant assignment vote
S6 Execution	$O(K \cdot n \cdot m + E)$	Checkpoint votes plus handoffs
S7 Final	$O(K \cdot m)$	Final-completion vote
Repair (per epoch)	$O(K \cdot n \cdot m^2)$	Re-runs subset of S2–S7
Total worst case	$O(E_{\max} \cdot K \cdot n \cdot m^2)$	Assignment-dominated

Table 29: Empirical coordination overhead (mean over 5 seeds, twelve-voter team). Coordination and execution columns separate the costs of running the URCP protocol from the costs of agents performing subtask work. The overhead percentage is computed as coord./(coord.+exec.).

Benchmark	Med. msgs	Coord. tok.	Exec. tok.	Coord. lat. (s)	Exec. lat. (s)	Total cost ($)	Coord. %
MAB	312	47.3K	158.6K	23.4	84.7	1.21	23.0%
τ-bench	287	42.1K	142.3K	21.8	79.6	1.08	22.8%
MemA	426	71.4K	234.8K	34.6	128.3	1.74	23.3%
Mean	342	53.6K	178.6K	26.6	97.5	1.34	23.0%

participates in scales with both the schema size n and the revision-and-repair budgets, the per-voter compute over a full run is bounded by $O(E_{\max} \cdot K \cdot n \cdot \theta)$. With $E_{\max} = 11$, $K_\ell = 5$, and the median observed schema size of $n = 7$ on the default operating point, the worst-case bound evaluates to approximately 385θ per voter. Larger task tiers with larger schemas (e.g., $n = 18$ on T3 tasks) raise this proportionally. The empirical mean across all benchmarks is 26θ per voter, reflecting that most stages converge in one round and that repair epochs beyond the first are rare.

The eligibility filter and the mechanical pre-checks for schema validity (coverage, acyclicity, type coherence, reachability) are $O(n+E)$ in the schema size. Topological sort over the DAG is $O(n+E)$. None of these scales with m, so the runtime cost grows gracefully with pool size.

A8.3 Empirical Coordination Overhead

Table 29 reports empirical overhead at the default configuration ($m = 12$, $q = 3/4$) on the three primary benchmarks, separating coordination cost from execution cost. "Coordination tokens" count tokens spent on proposal, vote, and justification messages; "execution tokens" count tokens spent inside assigned agents performing subtask work. The pattern is consistent across benchmarks: coordination accounts for approximately 23% of total cost, with the remainder spent on actual subtask execution.

This overhead translates to roughly \$0.27–\$0.40 in coordination cost per task. Table 30 compares these numbers to other coordination systems under matched per-run budgets. Transport-only baselines (A2A, MCP) incur lower coordination overhead because they perform almost no coordination at all; coordination-heavy systems (MetaGen, AFLOW, ReSo) incur similar or higher overhead than URCP. The cost-effectiveness measure (task score per dollar) places URCP roughly on par with the transport-only baselines while substantially exceeding the other coordination-heavy systems, indicating that URCP's coordination cost is paid back in accuracy.

A8.4 Scaling with Voter Set Size

Asymptotically, the worst-case coordination cost can scale as $O(m^2)$ because each voter may evaluate proposals broadcast by other voters. To check empirical scaling, we vary $|\mathcal{M}^*|$ from 3 to 24 while

Table 30: Coordination overhead comparison on MABench (mean over 5 seeds). "Coord. msgs" counts messages on the coordination layer and excludes tool calls. "Score/$" is task score divided by total cost per run, providing a rough cost-effectiveness measure.

System	Coord. msgs	Coord. tok.	Total tok.	Cost ($)	Score/$
A2A	142	11.6K	167.3K	0.92	59.5
MCP	128	10.2K	161.8K	0.88	59.2
MetaGen	286	51.8K	218.4K	1.36	45.9
AFLOW	318	58.4K	232.7K	1.43	42.5
AgentNet	234	38.2K	196.3K	1.18	50.3
ReSo	296	49.7K	211.6K	1.31	46.7
URCP	**312**	**47.3K**	**205.9K**	**1.21**	**59.6**

Table 31: Empirical scaling with voter set size m on MABench T3 tasks (mean over 3 seeds). Median DAG depth and role-instance count are similar across rows, isolating the scaling effect of m alone. Coordination cost grows approximately linearly over this range rather than quadratically, because most decisions converge in one or two rounds and the number of proposals per stage remains bounded.

$\lvert\mathcal{M}^*\rvert$	Coord. msgs	Coord. tok.	Coord. lat. (s)	Total cost ($)	Task score	NoC. rate
3	84	9.7K	6.4	0.31	62.8	1.1%
6	168	21.4K	12.8	0.62	68.4	1.4%
9	246	34.1K	18.7	0.93	70.7	1.7%
12	312	47.3K	23.4	1.21	72.1	2.1%
16	432	71.6K	32.6	1.68	71.4	4.2%
20	564	98.4K	42.3	2.18	70.2	7.8%
24	712	132.7K	53.8	2.83	68.9	11.3%

holding the task complexity tier fixed at T3 (median DAG depth $d = 9$, role-instance count $n = 18$). Results appear in Table 31.

Three patterns are visible. First, coordination cost grows close to linearly in this range: increasing m from 3 to 24 increases coordination messages from 84 to 712, an $8.5\times$ increase for an $8\times$ larger voter set. This is well below the worst-case quadratic envelope and reflects fast convergence in practice: most decisions clear Φ_q in one or two rounds rather than reaching the budget $K = 5$. Second, task score is non-monotone in m, peaking at $m = 12$ and declining for larger pools. The driver is the supermajority threshold itself: $\lceil 0.75m \rceil$ becomes harder to clear as m grows, because additional voters create more opportunities for dissent, revision, or abstention. Third, the NO-CONSENSUS rate climbs from 1.1% at $m = 3$ to 11.3% at $m = 24$, showing the structural cost of stricter consensus over larger voter sets.

These results suggest URCP scales gracefully to moderate pool sizes ($m \leq 16$) without architectural change. Beyond $m \approx 20$, the supermajority threshold becomes the binding constraint before message complexity does. Hierarchical voting, where sub-committees form local consensus that is then ratified by a smaller plenary, would address larger deployments but is left for future work.

A8.5 Scaling with DAG Depth

DAG depth d enters complexity through the number of checkpoint votes and handoff messages, both linear in d for fixed schema width. Table 32 reports empirical scaling along the depth axis with $m = 12$ fixed.

Coordination cost grows roughly linearly with depth, at a slope of approximately 31 messages per unit of depth. Repair rate grows super-linearly because a longer DAG provides more places where a checkpoint can fail. Despite these increases, task score degrades by only 9.5 points from $d = 3$ to $d = 26$.

A8.6 Storage and Auditability Overhead

The seven append-only structures grow monotonically within an epoch but are bounded per epoch by the total message count. The audit log AL averages 4.7 MB per run on MAB, 4.2 MB on τ-bench,

Table 32: Empirical scaling with observed DAG depth d on MABench (mean over 5 seeds, $m = 12$). Coordination cost grows linearly with depth, while repair rate grows super-linearly because longer DAGs accumulate more checkpoint failures.

DAG depth bin	Med. d	Coord. msgs	Coord. lat. (s)	Score	Repair rate
2–4	3	184	14.2	73.8	12.4%
5–10	7	312	23.4	72.1	18.7%
11–20	15	542	41.6	67.1	28.3%
21+	26	894	68.7	64.3	41.6%

and 6.8 MB on MemA, including all signed messages, justifications, and tally records. This is small enough that deployments can retain logs indefinitely. The single-writer append-only design admits compression ratios of approximately 4:1 with standard tools, since coordination messages share substantial JSON structure.

A8.7 Summary

URCP's coordination overhead is approximately 23% of total run cost at the operating point, scales sub-quadratically in voter set size within the practically relevant range, and scales linearly in DAG depth. The dominant asymptotic term is the per-claimant assignment vote at $O(n \cdot m^2)$, but empirical scaling is closer to $m^{1.7}$ due to fast convergence. The protocol is practical at moderate pool sizes ($m \leq 16$) and DAG depths ($d \leq 30$); larger configurations would benefit from hierarchical extensions.

A9 Practical Deployment and Real-World Limitations

This appendix discusses practical considerations for deploying URCP in production settings, real-world limitations we have encountered or anticipate, and integration paths with existing infrastructure. While the experiments in the main paper establish that URCP works under controlled benchmark conditions, several aspects of real-world deployment merit explicit discussion.

A9.1 Integration with Production APIs and Legacy Systems

URCP assumes that participants can be addressed through an authenticated message bus and can produce structured JSON messages conforming to the protocol's schemas. Three integration patterns arise in practice.

Pattern 1: Direct integration with API-served LLMs. Models accessible via standard chat-completion APIs (OpenAI, Anthropic, Google, etc.) integrate cleanly: the message bus posts URCP coordination messages as user-turn content, the model returns a structured response, and the runtime validates and broadcasts. We used this pattern for all twelve voters in our experiments. The main practical concern is rate limiting; the runtime must coordinate per-API quotas to avoid throttling, and the per-run wall-clock budget should be set with API latency variance in mind.

Pattern 2: Wrapping legacy systems as executable resources. Legacy services that do not produce structured JSON — internal databases, deterministic scoring engines, ETL pipelines, older proprietary models — can be wrapped as non-voting executable resources in $\mathcal{R}^*$ via manifest-proxy registration. The runtime signs the `IDENTITY` record on the legacy system's behalf, declares its capabilities, and exposes a thin adapter that translates between the URCP message format and the legacy system's RPC. This pattern was used for our six executable resources (code sandbox, retrieval index, OCR engine, image generator, TTS engine, form filler). The constraint is that the legacy system must be deterministic enough that the manifest-declared capabilities are honest; if the wrapper is unreliable, eligibility decisions become unsound.

Pattern 3: Hybrid deployments with on-prem and cloud participants. Real deployments often mix on-premise models (for data-residency or latency reasons) with cloud APIs. URCP's authenticated-channel assumption is satisfiable through mTLS over either substrate, but the bounded-delivery assumption (τ_{send}) needs careful budgeting: cloud-API tail latencies can exceed 30 seconds

Table 33: Aggregate pilot statistics (anonymized). "Quality lift" is the human-rated quality improvement over the team's prior tooling. "Adoption" indicates whether the team continued using URCP after the pilot ended.

Pilot	$\lvert\mathcal{M}^*\rvert$	Duration	NoC. rate	Quality lift	Adoption
A: Doc triage	6	14d	4.2%	+18%	Yes (modified)
B: Data analysis	9	28d	6.1%	+24%	Yes
C: Support augmentation	4	21d	11.7%	+9%	No (cost)

under load, which interacts with τ_{vote}. We recommend setting τ_{vote} to at least three times the 99th-percentile cloud-API latency observed in pilot runs.

A9.2 Real-World Pilot Experience

We deployed a prototype URCP implementation in three small-scale pilot settings to validate the framework outside benchmark conditions. Each pilot ran for two to four weeks with the production team in the loop. We summarize lessons learned without disclosing organization-identifying details.

Pilot A: Internal documentation triage. A 6-voter URCP deployment categorized incoming documentation queries, drafted candidate responses, and flagged cases requiring human review. The system processed approximately 400 queries per day for two weeks. Practical observations: *(i)* the median NO-CONSENSUS rate of 4.2% was higher than benchmark numbers, driven primarily by ambiguous queries that required human clarification—this was reported as a useful safety signal rather than a failure; *(ii)* coordination overhead was 31% of total cost, compared with 23% in benchmarks, because the queries themselves were short and coordination therefore occupied a larger fraction of the total budget; *(iii)* schema reuse via caching cut median latency from 28s to 11s once stable query patterns emerged.

Pilot B: Multi-source data analysis. A 9-voter deployment performed cross-source analysis combining SQL queries, document retrieval, and chart generation. Practical observations: *(i)* the executable-resource pattern worked well for deterministic services such as SQL execution and chart generation, but the team found it necessary to add per-resource health checks beyond the protocol's manifest registration; *(ii)* L_2 repair fired more often than in benchmarks (8.4% vs. 4.7%) because production data sources had intermittent availability; *(iii)* debugging required the audit log AL to be queryable as a structured store—raw appended JSON was too slow to navigate at scale, so we built a lightweight indexed view on top.

Pilot C: Customer-support augmentation. A 4-voter deployment drafted responses to customer messages with human approval before sending. Practical observations: *(i)* with $\lvert\mathcal{M}^*\rvert = 4$, both $q = 3/4$ and $q = 2/3$ produce the same acceptance threshold of $\lceil q\lvert\mathcal{M}^*\rvert\rceil = 3$ under the ceiling rule; therefore, changing q did not by itself lower the number of required affirmative votes. The pilot's NO-CONSENSUS rate was instead most sensitive to shorter revision budgets and stricter deployment-specific acceptance policies; *(ii)* integration with the existing ticketing system was the single largest engineering cost, exceeding the URCP runtime deployment effort; *(iii)* the structured audit log was directly useful for post-hoc quality review and was incorporated into the team's standard QA workflow.

Pilot quantitative outcomes are summarized in Table 33.

A9.3 Limitations and Failure Modes Observed in Practice

Cost variability. The dollar costs reported in the main paper are mean values; the tail distribution matters for production budgeting. The 99th percentile cost per run was 3.8× the median across our pilots, driven by runs that triggered L_3 or higher repair. Teams running URCP at scale should budget for this tail or impose hard cost ceilings via $E_{\max}$ tuning.

Vendor coordination. URCP's design assumes participants are interchangeable peers, but in practice cloud-API vendors update their models without notice. A model whose declared capabilities

passed the discovery probe at deployment time may quietly change behavior on a vendor update, silently degrading the eligibility filter's accuracy. We recommend periodic re-probing as a deployment policy; the protocol does not currently mandate this.

Privacy and data-handling boundaries. The encrypted-handoff pattern observed in some traces (Section on secure handoff subchannels) is an emergent behavior that operators may want to enforce structurally for sensitive workloads. The current protocol does not natively distinguish between sensitive and non-sensitive payloads at the schema level. A privacy-extended URCP would benefit from per-role data-classification annotations and policy-driven validation rules; this is on our future-work list.

Non-determinism in repair triggers. The decision of which repair level to invoke is itself voted on under Φ_q, which means that subtle disagreements about whether a failure is local (L_1) or structural (L_3) can produce non-deterministic behavior across runs even with identical inputs. This was visible in Pilot B where two seemingly identical SQL failures triggered L_2 on one run and L_3 on another. The audit log explained the divergence (different voters were online at different moments) but the variance was unexpected for the team.

Cold-start problem. URCP's per-stage timeouts (τ_{disc}, τ_{vote}, etc.) are calibrated for steady-state operation. The first run after a deployment, when caches are cold and connection pools are empty, often hits timeouts that subsequent runs do not. Production deployments should warm the system before measuring SLAs.

A9.4 Generalization Beyond LLMs

URCP's two-class participant model (vote-capable models, non-voting executable resources) is designed to accommodate non-LLM participants. We tested three non-LLM cases that did not appear in the main benchmarks but that future deployments may encounter.

Diffusion models in image-generation roles. The SR row `image_generation` $\rightarrow$ {`diffusion`} is mechanically supported by the eligibility filter, and our `img-gen-sdxl` resource was used in a small subset of multimodal trace runs not reported in the main results. The constraint is that diffusion models cannot inspect their own outputs to produce structured completion claims; the `SUBTASK-COMPLETE` message must be assembled by an adapter wrapper, not by the diffusion model directly. This works but couples the wrapper's correctness to the eligibility check.

Embodied controllers. We did not deploy a real VLA controller in the experiments reported, but the SR row `embodied_action` $\rightarrow$ {`vla_controller`} is in the registry for completeness. The practical question for embodied deployments is the latency budget: $\tau_{\text{exec},j}$ at 120s is generous for LLM inference but may be tight for multi-step physical action sequences. A deployment-specific override of $\tau_{\text{exec},j}$ per role would address this.

Symbolic solvers. We experimented with attaching a SAT solver as a non-voting resource for constraint-satisfaction subtasks. The integration was clean: the solver declares `reasoning` and `tool_use` capabilities, and the URCP eligibility filter correctly routes constraint-heavy subtasks to it. The unexpected finding was that voters frequently proposed schemas where the SAT solver appeared as a verifier role downstream of an LLM-generated solution, even when not directly prompted to do so — an example of the spontaneous-critic pattern but with a non-LLM resource as the critic.

A9.5 When URCP Is Not the Right Choice

We also note settings where URCP is unlikely to provide value over simpler alternatives:

(i) Tasks decomposable into a fixed pipeline of three or fewer stages, where the coordination overhead exceeds the marginal benefit of voted handoffs.

(ii) Tasks where a single dominant model substantially outperforms all alternatives on every subtask — URCP's diversity-driven gains evaporate when the team is effectively homogeneous in capability.

Table 34: Repair effectiveness by level (heterogeneous team, all three benchmarks). Trigger rate is the fraction of runs in which a repair at this level (or higher) is initiated. Success rate is the fraction of those triggers in which the run subsequently reached ACCEPT on FINAL-SUBMIT. Cost and latency are marginal contributions per invocation. The hierarchy follows Table 6: L_1 output, L_2 assignment, L_3 schema, L_4 subtask set, L_5 task frame. The L_2 success rate reflects the deterministic Role Ranking Table fallback mechanism of Appendix A1.11.

Level	Trigger	Success	Cost ($)	Lat. (s)	Modal failure cause
L_1: Output	18.0%	78.4%	0.04	6.2	Validation failure
L_2: Assignment	4.7%	64.1%	0.09	12.8	Handoff timeout
L_3: Schema	1.8%	51.7%	0.21	27.4	Inter-role type mismatch
L_4: Subtasks	0.4%	42.0%	0.31	38.6	Coverage gap / dead-end
L_5: Frame	0.05%	25.0%	0.52	58.3	Wrong task interpretation
Any repair	24.95%	73.4%	0.07	9.3	—

(iii) Real-time latency-bound applications (sub-second response) where the coordination overhead (median 23s in our benchmarks) is itself disqualifying.

(iv) Settings without an authenticated channel substrate; URCP's Assumption A1.1 is non-trivial to satisfy in ad-hoc multi-vendor settings where mTLS infrastructure is absent.

A9.6 Summary

URCP's protocol-level guarantees translated to pilot deployments without major surprises, but real-world deployment surfaced practical concerns that benchmark experiments did not: cost-tail budgeting, vendor model drift, the cold-start problem, and the engineering cost of integrating with legacy ticketing and analytics systems. The protocol's assumptions (authenticated channels, bounded delivery, synchronous phases) are satisfiable in practice but require explicit infrastructure that small teams may underestimate. We see URCP as most valuable in settings with (a) a diverse model pool, (b) tasks complex enough that coordinated governance pays back its overhead, and (c) compliance or auditability requirements that benefit from the explicit log.

A9.7 Repair-Level Effectiveness

Section 4.4 reports how often each repair level is invoked but does not report whether each level *succeeds*. The distinction matters: a repair that fires often but fails is decorative; a repair that rarely fires but always rescues an otherwise-doomed run is essential. Table 34 reports both quantities by level for the full five-level hierarchy of Table 6 (specified in Appendix A1.14), together with the marginal cost and latency added per invocation, on the heterogeneous twelve-voter team aggregated across all three benchmarks.

The shape of the table validates the hierarchical design. Lower levels are inexpensive and frequently successful: L_1 fires on 18% of runs and recovers 78.4% of them at a marginal cost of $\$0.04$ per invocation. Higher levels are expensive and have lower success rates, but they fire on a much smaller fraction of runs and serve as escape hatches when local fixes are insufficient. The L_5 success rate of 25.0% may look modest in isolation, but it should be read against the counterfactual: without L_5, those 0.05% of runs would terminate in NO-CONSENSUS rather than in any output at all. The L_5 mechanism converts a quarter of those terminal failures into completions, at the cost of a full restart in a new epoch (Table 6).

The modal-failure column suggests where each level adds the most value. L_1 catches per-stage validation failures that are local to a single role; the runtime emits a `REOPEN` at L_1 on $\tau_{\text{exec},j}$ expiry per Appendix A1.12, and the bound agent is retried with revision suggestions without modifying the assignment. L_2 handles binding failures (a model fails to commit, or a `HANDOFF-RECEIVED` times out) that can be repaired by reassigning roles within the existing schema; the head of $\mathrm{RRT}[s_j]$ is popped and the next-ranked candidate is presented for a fresh Φ_q-valid assignment vote (Appendix A1.14, Algorithm 5). L_3 handles structural problems—most commonly type mismatches between connected roles—that require redesigning the role population. L_4 handles coverage gaps and structural dead-ends that surface only after multiple schema revisions fail the same mechanical

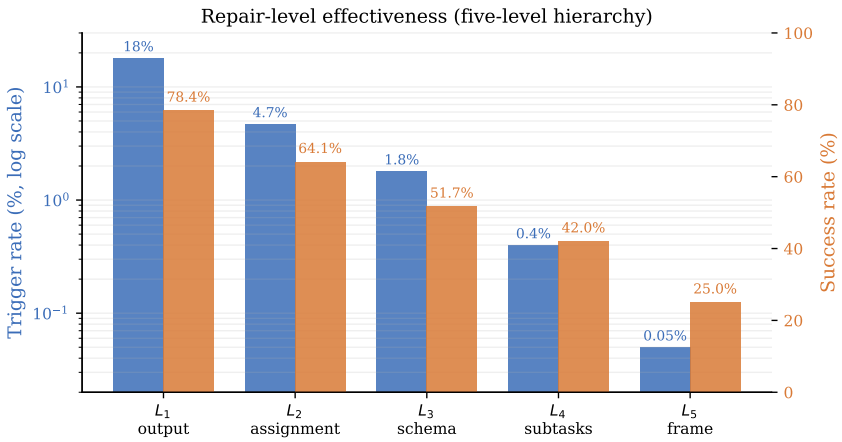


Figure 14: Repair-level trigger rate (fraction of runs that invoke this level or higher) versus success rate (fraction of those invocations that recover). The five-level hierarchy of Table 6 is monotone in both dimensions: lower levels fire often and usually succeed; higher levels fire rarely and act as last-resort escape hatches.

Table 35: Cumulative cost of disabling repair levels from the top down on MABench (heterogeneous twelve-voter team, mean over 5 seeds). Δ is the score change relative to default URCP. The final row corresponds to the all-repair-disabled counterfactual that loses 6.3 points overall, decomposed in the prose above using the per-level success rates of Table 34.

Disabled levels	MABench score	Δ
None (default URCP)	72.1	—
$\{L_5\}$	72.1	0.0
$\{L_4, L_5\}$	71.9	-0.2
$\{L_3, L_4, L_5\}$	71.1	-1.0
$\{L_2, L_3, L_4, L_5\}$	69.4	-2.7
$\{L_1, L_2, L_3, L_4, L_5\}$	65.8	-6.3

pre-check (Appendix A1.9), requiring the subtask set itself to be revisited. L_5 handles the rarest case: a frame that, in retrospect, decomposed the task incorrectly. The trigger distribution is consistent with the prior we would assign these failure modes: most failures are local, structural failures are uncommon, and reframing the task entirely is exceedingly rare.

Cumulative contribution per level. A counterfactual variant of URCP that disables all five repair levels drops MABench performance by 6.3 points relative to the default configuration. Table 34 lets us decompose this loss approximately. L_1 success on $18.0\% \times 78.4\% = 14.1\%$ of runs accounts for the bulk of the contribution; L_2 contributes another 3.0%, L_3 a further 0.93%, and L_4 and L_5 together the final $\sim 0.18\%$ of recoveries. A cumulative analysis, disabling levels from L_5 downward, confirms the breakdown (Table 35): removing only L_5 costs essentially zero points within noise, removing $L_4{+}L_5$ costs 0.2 points, removing $L_3{+}L_4{+}L_5$ costs 1.0 point, removing $L_2{+}L_3{+}L_4{+}L_5$ costs 2.7 points, and removing all five levels recovers the -6.3 figure. The hierarchy thus pulls its weight at every level, even though the higher levels fire only on rare structural failures.

A10 Algorithm

Algorithm 6 specifies the control flow of URCP. Message validation, decision-specific electorates, role eligibility, ranking, and dependency-aware preservation follow Appendix A1. The algorithm separates semantic ratification from mechanical state transitions: a failure may generate a repair proposal, but no repair is applied before that proposal receives the required affirmative votes.

Bounded operations. Every stage invocation has an independent, non-resetting deadline Δ_{stage}, in addition to its individual message timeouts and semantic attempt budgets. A run has a non-resetting wall-clock limit T_{run} and a declared resource budget. Every wait, loop, and helper operation below is subject to these bounds. Mechanical rejections do not consume semantic attempts, but cannot reset or bypass a stage deadline.

Expiration of a stage or run bound terminates the run as NO-CONSENSUS, with the cause recorded in the audit log. These bounds are requirements of the complete specification, not additional experimental results.

Common decision operation. $\text{Decide}(\text{kind}, P)$ opens a fresh decision attempt on substantive proposal P, records its electorate and deadline, collects valid ballots, and invokes Algorithm 3. Ordinary decisions use $\mathcal{M}^*$. Assignment to claimant r uses $\mathcal{M}^* \setminus \{r\}$, which removes a voter only when $r \in \mathcal{M}^*$. An empty electorate produces `NO-ELIGIBLE-VOTERS`, never zero-vote acceptance. Each outcome is committed and broadcast before a dependent transition proceeds.

Proposal processing. NextProposal collects a finite batch within τ_{propose} and processes it in the published timestamp and identifier order. If no proposal arrives, it invokes the designated fallback within τ_{fallback}. Failure of the fallback terminates as NO-CONSENSUS. All stale-proposal checks use substantive hashes that exclude transport metadata.

Repair requests. $\text{RequestRepair}(L, U, c)$ records a repair proposal with level L, affected objects U, and failure cause c, then transfers control to REPAIR. When the appropriate scope requires semantic judgment, voters propose that scope through `REOPEN`; the runtime does not infer it from unrestricted natural-language explanations. Runtime-generated proposals use the predefined mapping from typed failures to repair levels.

Algorithm 6 URCP: Unified Resource-Grounded Coordination Protocol

---

Require: Task T, optional guidance G, and resource pool $\Omega = (\mathcal{M}, \mathcal{R}, \mathcal{U}, \mathcal{B}, \mathcal{C})$
Require: Threshold $q \in (1/2, 1]$, default $q = 3/4$
Require: Message timeouts from Table 7
Require: Semantic budgets $K_1, K_3, K_4, K_5, K_6, K_7$, epoch cap $E_{\max}$, stage deadlines Δ_{stage}, run limit T_{run}, and resource budget
Ensure: A recorded terminal verdict; an accepted output Y is returned only with COMPLETE

Initialize $e \leftarrow 1$ and $\text{resume} \leftarrow 1$
Initialize the append-only event history and materialized views $\text{CR}, \text{SR}, \text{VL}, \text{DR}, \text{RRT}, \text{OLT}, \text{AL}$
Initialize $F, \mathcal{B}, \mathcal{S}, \mathcal{O}, \mathcal{A} \leftarrow \perp$
Initialize empty pending-repair and fallback state
Start the non-resetting run deadline and resource accounting
Install deadline guards for every operation below
while $e \leq E_{\max}$ **do**

Stage 1: Discovery and context verification
  if $\text{resume} = 1$ **then**
    Start a bounded discovery invocation
    Reset active discovery views without deleting history
    Process `IDENTITY` messages and required probes within τ_{disc} using Algorithm 2
    Form $\mathcal{M}^* \leftarrow \{p \in \text{CR} : \text{CR}[p].\text{vc} = \top\}$
    Form $\mathcal{R}^* \leftarrow \{p \in \text{CR} : \text{CR}[p].\text{caps} \neq \emptyset\}$
    if $\mathcal{M}^* = \emptyset$ **then**
      return NO-VOTERS with terminal audit record

```
        end if
        if $\mathcal{R}^* = \emptyset$ then
            return NO-RESOURCES with terminal audit record
        end if
        Freeze the discovered voter set for the epoch
        Publish SR and the canonical context tuple
        Collect CONTEXT-VERIFY responses within $\tau_{\text{verify}}$
        Count matching responses as YES and missing responses as ABSTAIN; retain the fixed discovered denominator
        if matching responses are fewer than $\lceil q|\mathcal{M}^*| \rceil$ then
            return CONTEXT-INCONSISTENT with terminal audit record
        end if
        Commit and broadcast the accepted context
    end if

    Stage 2: Task framing
    if resume $\leq 2$ then
        Start a bounded framing invocation
        $F \leftarrow \perp$, $a \leftarrow 0$, $\mathcal{H}_F \leftarrow \emptyset$
        while $F = \perp$ and $a < K_1$ do
            $P \leftarrow \text{NextProposal}(\texttt{FRAME})$
            if $P$ is stale, malformed, or violates a mechanically checkable mandatory constraint then
                Record rejection without resetting the stage deadline
                continue
            end if
            $a \leftarrow a + 1$
            $v \leftarrow \text{Decide}(\texttt{FRAME}, P)$
            if $v =$ ACCEPT then
                Lock $F \leftarrow P$
            else
                Record the substantive hash in $\mathcal{H}_F$
                Return recorded dissent to proposers
            end if
        end while
        if $F = \perp$ then
            return NO-CONSENSUS
        end if
    end if

    Stage 3: Subtask formation
    if resume $\leq 3$ then
        Start a bounded subtask invocation
        $\mathcal{B} \leftarrow \perp$, $a \leftarrow 0$
        Initialize accepted-candidate cache $\mathcal{C}^+$ and rejected-candidate cache $\mathcal{C}^-$
        while $\mathcal{B} = \perp$ and $a < K_3$ do
            $P \leftarrow \text{NextProposal}(\texttt{SUBTASK})$
            if $P$ lacks its proposal-level coverage argument or references an invalid frame then
                Reject $P$ and continue
            end if
            Canonicalize candidates by all substantive fields
            $\mathcal{B}_P \leftarrow \emptyset$
            for each candidate $b$ in the proposed set do
                if $b$ fails mechanical or hard-SR checks then
                    Record rejection
                else if the unchanged candidate $b$ is in $\mathcal{C}^+$ then
                    Add $b$ to $\mathcal{B}_P$
                else if $b$ is in $\mathcal{C}^-$ then
                    Record STALE-CANDIDATE
```

```
                else
                    v ← Decide(SUBTASK, b)
                    if v = ACCEPT then
                        Add b to C⁺ and B_P
                    else if v = REJECT then
                        Add b to C⁻
                    else
                        Return candidate-specific revision feedback
                    end if
                end if
            end for
            a ← a + 1
            v ← Decide(COVERAGE, B_P)
            if v = ACCEPT then
                Lock B ← B_P
            else
                Return coverage feedback; retain only the unchanged candidate certificates for possible reuse
            end if
        end while
        if B = ⊥ then
            return NO-CONSENSUS
        end if
    end if
```

Stage 4: Role schema and execution DAG

```
    if resume ≤ 4 then
        Start a bounded schema invocation
        S, O ← ⊥
        a ← 0, f ← 0, lastFail ← ⊥
        Initialize the stale-schema hash set
        while S = ⊥ and a < K_4 do
            P ← NextProposal(SCHEMA)
            if P has an unchanged previously unsuccessful substantive hash then
                Record STALE-REVISION
                continue
            end if
            Check coverage, acyclicity, declared interface bindings, and reachability
            if a mechanical check fails with reason c then
                Record rejection and the substantive hash
                if c = lastFail then
                    f ← f + 1
                else
                    f ← 1, lastFail ← c
                end if
                if f ≥ 3 then
                    RequestRepair(L_4, B, STRUCTURAL-DEAD-END)
                    goto REPAIR
                end if
                continue
            end if
            f ← 0, lastFail ← ⊥
            a ← a + 1
            v ← Decide(SCHEMA, P)
            if v = ACCEPT then
                Lock the complete schema S and DAG O from P
            else
                Record the substantive hash and return dissent
            end if
```

```
        end while
        if S = ⊥ then
            return NO-CONSENSUS
        end if
        Reconcile historical bindings and output candidates against the accepted schema
        Carry forward only explicitly authorized, compatible state
    end if

Stage 5: Role assignment
    if resume ≤ 5 then
        Start a bounded assignment invocation
        for each role s_j in deterministic topological order do
            if A(s_j) has a valid carried-forward binding then
                continue
            end if
            if s_j is marked for fallback assignment then
                r ← PresentRepairCandidate(L_2, s_j)        ▷ Algorithm 5; removes the failed bound entry once
                if r = NIL then
                    goto REPAIR                              ▷ The helper has proposed L_3
                end if
                Clear the fallback-assignment marker
                continue
            end if
            Collect claims within τ_claim
            Filter claims using Algorithm 4
            Sort eligible claimants into finite ordered list C_j
            A(s_j) ← ⊥, a ← 0
            for each r ∈ C_j, while a < K_5 do
                a ← a + 1
                v ← Decide(ASSIGNMENT, (s_j, r))
                if v = ACCEPT then
                    Bind A(s_j) ← r
                    Construct RRT tiers from the bound, unvoted, and voted-failed claims
                    break
                else
                    Record the failed attempt or NO-ELIGIBLE-VOTERS
                end if
            end for
            if A(s_j) = ⊥ then
                RequestRepair(L_3, {s_j}, NO-SUITABLE-MODEL)
                goto REPAIR
            end if
        end for
    end if

Stage 6: Execution and checkpoint validation
    Start a bounded execution invocation
    for each role s_j in deterministic topological order do
        if OLT[s_j] is active and dependency-consistent then
            continue
        end if
        for each required incoming edge (s_i, s_j) do
            Require the active upstream artifact OLT[s_i]
            Open a handoff window of length τ_handoff
            Deliver the locked artifact to A(s_j)
            Accept only a matching authenticated HANDOFF-RECEIVED
            Permit corrections within the original handoff window
            if no valid acknowledgment arrives before the deadline then
```

```
                RequestRepair(L_2, {s_j}, HANDOFF-FAIL)
                goto REPAIR
            end if
        end for
        a ← 0, locked ← false
        while not locked and a < K_6 do
            Invoke A(s_j) using the accepted contract and locked upstream artifacts
            Wait for SUBTASK-COMPLETE within τ_{exec,j}
            if the execution call times out then
                RequestRepair(L_1, {s_j}, TIMEOUT)
                goto REPAIR
            end if
            Check sender authorization, output interface, required evidence, contract, and upstream hashes
            if a mechanical check fails then
                Record rejection without resetting the execution-stage deadline
                continue
            end if
            a ← a + 1
            v ← Decide(OUTPUT, checkpoint)
            if v = ACCEPT then
                Lock the artifact with its producer, certificate, contract, and upstream dependencies
                locked ← true
            else if v = REJECT then
                Record the validation failure and solicit typed repair proposals
                goto REPAIR
            else
                Return recorded revision feedback to the executor
            end if
        end while
        if not locked then
            RequestRepair(L_2, {s_j}, VALIDATION-FAIL)
            goto REPAIR
        end if
    end for
```

Stage 7: Final completion

```
    Start a bounded final-completion invocation
    a ← 0
    while a < K_7 do
        The assigned final executor assembles a candidate Y within its bounded execution window
        if assembly cannot complete within the window then
            RequestRepair(L_3, assembly scope, ASSEMBLY-FAILURE)
            goto REPAIR
        end if
        Check the FINAL-SUBMIT interface, provenance, and active component hashes
        if a mechanical check fails then
            Record rejection without resetting the stage deadline
            continue
        end if
        a ← a + 1
        v ← Decide(FINAL, Y)
        if v = ACCEPT then
            return (Y, COMPLETE) with its certificate
        else if v = REJECT then
            Record the failure and solicit typed repair proposals
            goto REPAIR
```

else
Return feedback to the final executor and voters
if a proposed revision changes locked upstream artifacts **then**
Require a typed `REOPEN` proposal
goto REPAIR
end if
end if
end while
return NO-CONSENSUS

REPAIR: Bounded repair authorization
Start a bounded repair invocation
Collect a finite set of valid `REOPEN` proposals, including the triggering proposal, within τ_{repair}
if no valid proposal is available **then**
Emit the predefined runtime repair proposal for the recorded failure type
end if
Sort proposals by descending scope, then timestamp and identifier
$\rho^* \leftarrow \perp$
for each proposal ρ in the finite ordered set **do**
$v \leftarrow \mathrm{Decide}(\texttt{REPAIR}, \rho)$
if $v =$ ACCEPT **then**
$\rho^* \leftarrow \rho$
break
end if
end for
if $\rho^* = \perp$ **then**
return NO-CONSENSUS ▷ Do not reduce the electorate to obtain a quorum
end if
if $e = E_{\max}$ **then**
return NO-CONSENSUS ▷ No additional epoch is permitted
end if
Compute the ratified preservation and invalidation plan
Commit one epoch transition $e \leftarrow e + 1$
Broadcast `EPOCH-ADVANCE` with authorization, resume stage, and preserved/invalidated state
Reject old-epoch messages as new state-changing inputs
if $\rho^*.\mathrm{level} = L_1$ **then**
Let U be the directly affected role set
Invalidate outputs in $\mathrm{cl}_{\mathcal{O}}(U)$
Carry forward the frame, subtasks, schema, valid bindings, and independent valid outputs
$\mathrm{resume} \leftarrow 6$
else if $\rho^*.\mathrm{level} = L_2$ **then**
Let U be the directly affected role set
Invalidate outputs in $\mathrm{cl}_{\mathcal{O}}(U)$
Invalidate active bindings for roles in U
Retain their historical failed bound entries and fallback rankings for Algorithm 5
Mark roles in U for fallback assignment
Carry forward unaffected bindings and independent valid outputs
$\mathrm{resume} \leftarrow 5$
else if $\rho^*.\mathrm{level} = L_3$ **then**
Carry forward F and $\mathcal{B}$
Retain historical role, binding, and artifact records only as inactive reuse candidates
Clear active schema, DAG, assignments, RRT, and output locks
Clear obsolete fallback-assignment markers
$\mathrm{resume} \leftarrow 4$
else if $\rho^*.\mathrm{level} = L_4$ **then**
Carry forward F and unchanged discovery state
Clear active subtasks and all downstream coordination state
Clear obsolete fallback-assignment markers

```
        resume ← 3
    else if ρ*.level = L_5 then
        Clear active frame and all downstream coordination state
        Clear obsolete fallback-assignment markers
        if ρ*.rediscovery = true then
            Clear active discovery views
            resume ← 1
        else
            Carry forward the accepted discovery context
            resume ← 2
        end if
    end if
    Clear the processed repair queue
    Preserve the original run deadline and remaining resource budget
end while
return NO-CONSENSUS
```

Implementation interpretation. Clearing state in Algorithm 6 means appending invalidation events and updating active views; it does not erase the audit history. At L_3, historical artifacts remain inactive until the newly accepted schema explicitly permits their reuse and their bindings, contracts, and dependencies pass the checks in Appendix A1.14. Subsequent assignment or execution cannot rely on an inactive historical record.

The serial topological execution shown above is sufficient to specify the dependency order. A parallel implementation may run independent ready roles concurrently while retaining the same assignment, handoff, checkpoint, and deadline requirements.

A11 Additional Experiments

This appendix examines URCP's coordination guarantees, additional comparisons, voting behavior, and deployment trade-offs. Default URCP uses the full verified electorate except for claimant exclusion in model-role assignment. Committee and local-subset policies are treated as extensions because they change which participants authorize a decision. Reported measurements are distinguished from specification requirements and infrastructure mechanisms that have not been evaluated end to end.

A11.1 Unified governance and the meaning of orchestrator-free coordination

URCP represents task framing, subtasks, role schemas, execution dependencies, assignments, checkpoint outputs, repairs, and final completion as typed coordination objects governed by Φ_q. The contribution is their integration into a shared authorization and repair procedure. A failed checkpoint can lead to a revised assignment or schema, but that transition requires ratification before execution resumes.

The term *orchestrator-free* refers to the absence of a central *semantic* decision-maker. The mechanical runtime delivers messages, verifies identities, checks declared eligibility, orders proposals, enforces deadlines, tallies ballots, and maintains state. These policies influence the proposals considered and their timing, so they are explicit and auditable. However, ordering a claimant does not authorize its assignment, and proposing a default repair does not authorize its execution.

The deployment manifest and Specialization Registry remain externally specified constraints. URCP does not independently derive all deployment policy. Its centralized mechanical runtime also remains an infrastructure dependency: runtime failure can interrupt a run even though no individual language model can unilaterally determine the workflow.

Related systems address complementary aspects of coordination. MetaGen adapts roles and topology at inference time, while ReSo combines task-graph generation with reward-driven agent selection [36, 45]. URCP's distinguishing design is the uniform ratification and repair of coordination state; voting, dynamic roles, and recovery are not claimed individually as new mechanisms.

A11.2 Guarantee conditions, quorum arithmetic, and bounded repair

The properties in Appendix A1.16 depend on authenticated communication, bounded delivery among active participants, synchronous phases, fixed epoch membership, and a runtime that continues to execute its transition rules. Finite termination additionally requires the processing and independent stage bounds in Appendix A1.15.

The guarantees concern authorization, compatibility with admitted declarations, auditability, and bounded execution. They do not establish factual correctness, unrestricted model competence, or eventual semantic agreement. A `NO-CONSENSUS` verdict records the absence of an authorized completion; it does not establish that every candidate answer is incorrect.

Decision-specific electorates. For a nonempty authorized electorate $V_e(D)$, define

$$t_e(D) = \lceil q|V_e(D)| \rceil, \qquad s_e(D) = |V_e(D)| - t_e(D). \tag{2}$$

Acceptance requires at least $t_e(D)$ explicit YES ballots. The slack $s_e(D)$ is the number of non-affirmative or unavailable voters compatible with acceptance *when all remaining eligible voters vote YES*. It is not a guarantee of successful coordination.

Ordinary decisions use $\mathcal{M}^*$. Assignment to voter r uses $\mathcal{M}^* \setminus \{r\}$. Assignment to a non-voting resource retains the full electorate. An empty assignment electorate produces a typed failure rather than zero-vote acceptance.

With twelve voters and $q = 3/4$, ordinary decisions require nine YES votes, while assignment to a voter requires nine of the remaining eleven. Three persistently non-affirmative voters can therefore leave an ordinary decision feasible but prevent assignment to a cooperative voter outside that group. Two such voters leave nine eligible cooperative voters for that assignment.

Repair uses the current authorized electorate. If it cannot obtain a quorum, the runtime terminates under the bounded failure rule. It cannot remove dissenting or unavailable voters merely to make a repair pass. Membership changes require an authorized rediscovery transition or a separate restart.

Timing and failure envelope. Missing ballots become abstentions at the decision deadline. Missing proposals invoke a bounded fallback, and unacknowledged handoffs trigger the specified repair procedure. These rules permit explicit failure handling without requiring every participant to respond.

A common runtime clock establishes the deadline authority, but messages arriving near a deadline can still affect the tally. Bounded clock skew does not imply invariance to all timing changes. An indefinitely unavailable runtime cannot execute timeout handlers or publish a terminal verdict. Likewise, a permanent partition prevents a guarantee that all participants eventually receive the same outcome unless communication is restored.

Finite execution versus semantic convergence. Semantic revision budgets alone do not bound mechanically rejected submissions. The specification therefore requires independent non-resetting stage deadlines or equivalent finite attempt bounds, covering invalid outputs, stale proposals, and alternating schema pre-check failures.

Together with finite inputs, bounded processing steps, an available runtime, and the epoch cap, these bounds prevent indefinite protocol execution. The run deadline is not reset by repair. Termination does not rely on revisions improving semantic quality: exhaustion may produce `NO-CONSENSUS`.

These requirements define the scope of the termination argument. They are not additional measured results. Committee extensions require equivalent bounds on local decisions, escalation, and workflow-wide decisions.

Repair scope and preservation. Table 36 summarizes the repair levels. Levels need not increase monotonically across a run. Concurrent proposals are considered in descending scope, but each transition requires its own ratification.

For output or assignment repair, the specification invalidates the affected outputs and their dependent descendants before execution resumes. Independent outputs may be retained. After schema repair, reuse additionally requires a compatible role contract, producer binding, acceptance criterion, and upstream dependency chain. Historical hashes alone do not authorize reuse.

Table 36: Repair scopes and representative triggers. Every reopening and escalation requires an accepted repair decision.

Level	Reopened object	Trigger and transition
L_1	Output	Local execution or validation failure; retry the bound executor after invalidating affected outputs and dependent descendants. Exhausted output attempts propose L_2.
L_2	Assignment	Unsuitable executor or consumer handoff failure; present stored fallbacks for fresh assignment votes. Exhausted candidates propose L_3.
L_3	Schema/DAG	Missing eligible executors, interface defects, or structural assembly failure; reconsider roles and dependencies.
L_4	Subtasks	Coverage or structural defects requiring a different decomposition.
L_5	Frame	Incorrect task interpretation; return to discovery only when participant, resource, or registry re-evaluation is explicitly ratified.

Table 37: Task scores with baseline adapter retries disabled or enabled. Gaps are URCP minus the strongest baseline. MetaGen is strongest on MAB and τ-bench; ReSo is strongest on MemA in both settings.

Benchmark	Best, off	Best, on	URCP	Gap, off	Gap, on
MAB	62.4	65.6	72.1	9.7	6.5
τ-bench	55.6	57.8	63.5	7.9	5.7
MemA	50.8	54.2	61.4	10.6	7.2

A11.3 Mechanical-runtime recovery and infrastructure availability

The authenticated append-only history supports reconstruction of committed decisions, assignments, and output locks. Crash recovery additionally requires durable artifact storage, a consistent committed log prefix, and persisted deadline and budget state. A hash does not reconstruct missing output bytes, and repeated restarts must not reset deadlines indefinitely.

Log replay does not provide exactly-once external effects. If a tool changes external state before its completion is durably recorded, recovery must reconcile that uncertain outcome. Possible mechanisms include idempotency keys, status queries, or application-specific compensation.

Replicated-log mechanisms such as Raft could support replication of the mechanical runtime [28]. Infrastructure replicas would agree on committed runtime state, while URCP voters would continue to authorize semantic decisions. Such a deployment must prevent conflicting runtime writers and inherits the replication system's own fault and availability assumptions.

No runtime fail-stop/log-replay experiment is reported here. The supplementary results therefore do not validate crash recovery or a replicated-runtime deployment.

A11.4 Native-retry and quality-aware routing comparisons

The main controlled comparison disables baseline adapter retries to expose failures to the coordination layer. The additional native-retry comparison restores the eight baselines' configured retry/reconnect settings while retaining the same model pool, tools, resources, seeds, and per-run limits of 500K tokens and 600 seconds.

Table 37 reports the strongest baseline under each setting. The native-retry scores are point estimates; confidence intervals for those additional scores are not reported.

Restoring retries improves the strongest baselines by 3.2, 2.2, and 3.4 points. URCP retains margins of 6.5, 5.7, and 7.2 points. The larger main-comparison margins should therefore be identified as results under retry-disabled baseline adapters.

Retries can recover transient failures, but do not by themselves implement schema ratification or structured coordination repair. The remaining score differences are consistent with benefits beyond transport recovery; they do not isolate the contribution of each URCP mechanism.

Table 38: Quality-aware routing with retries versus URCP. Values are mean ± 95% CI over five seeds. Margins compare the reported means.

Benchmark	Router + retry	URCP	Margin
MAB	66.7 ± 1.4	72.1 ± 1.2	5.4
τ-bench	58.6 ± 1.7	63.5 ± 1.5	4.9
MemA	55.1 ± 1.9	61.4 ± 1.7	6.3

Table 39: Assignment-policy interventions on MAB. Values are mean ± 95% CI over five seeds.

Assignment policy	MAB	Change from voted assignment
Voted assignment	72.1 ± 1.2	—
Best-model-only assignment	69.2 ± 1.4	-2.9
Random eligible assignment	66.8 ± 1.5	-5.3

A second comparison uses a quality-aware router inspired by RouteLLM [27], adapted to the same twelve-model pool with three retries per call and matched tools, resources, seeds, and budget caps. This is a RouteLLM-style baseline, not an unmodified evaluation of the original two-model framework.

Under the same post-generation type, evidence, and interface validator, the reported intermediate-output validation failure rates are 12.8% for the router and 9.8% for URCP, a difference of 3.0 percentage points. This common-validator comparison does not assume that every baseline-specific handoff metric has an identical operational definition.

The results support an advantage over the tested routing policy. Router cost and latency are not reported here, so the comparison does not establish a monetary or latency advantage.

A11.5 Assignment, repair, and heterogeneous-team effects

The assignment intervention replaces voted assignment while retaining the surrounding framing, schema, checkpoint, repair, model-pool, and budget configuration. Table 39 compares the default with best-model-only and random-eligible assignment. These are assignment-policy interventions within URCP, not standalone single-model systems.

The separate repair ablation in Appendix A9.7 reduces MAB from 72.1 to 65.8 when all five repair levels are disabled. The assignment and repair interventions support benefits from both mechanisms within the evaluated system. Their effects are not an additive decomposition of the overall gain: assignment can influence the frequency and difficulty of repair, and the interventions use different reference conditions.

The heterogeneous pool exceeds the strongest homogeneous configuration separately on each benchmark. The differences are $72.1 - 67.4 = 4.7$ on MAB, $63.5 - 59.2 = 4.3$ on τ-bench, and $61.4 - 56.1 = 5.3$ on MemA, averaging approximately 4.8 points. This comparison selects the best homogeneous configuration for each benchmark; comparison with one fixed homogeneous configuration is a different statistic.

The threshold sweep in Appendix A7.2 provides a complementary comparison. Majority voting scores 67.3 on MAB with 13.1% handoff failure, while $q = 3/4$ scores 72.1 with 6.4% handoff failure. Unanimity scores 72.3 with 4.7% handoff failure, but increases the NoC rate from 2.1% to 6.8% and cost from \$1.21 to \$1.51. These measurements motivate the tested default without establishing a universally optimal threshold.

The critic-role bin counts total

$$541 + 592 + 1{,}051 + 423 = 2{,}607$$

among $4{,}108$ accepted schemas with at least three roles, corresponding to approximately 63.5%. This frequency describes accepted workflow structure. It does not, without an intervention, establish that inserting a critic caused a score improvement.

Hyperparameters were selected on a held-out 10% MAB split and then fixed across the remaining evaluations. Base-model weights were not updated. These controls limit direct benchmark-specific tuning but do not eliminate selection effects, contamination, or sensitivity to task composition. Seed-level confidence intervals describe the reported variation, not generalization to arbitrary deployments.

A11.6 Execution concentration and voting behavior

Execution allocation and authorization. Four frontier models constitute one-third of the twelve-voter pool and receive 73.9% of critical-path roles. This is a concentration of execution responsibility. Other voters still participate in assignments, checkpoint validation, and repair decisions.

An offline analysis removes frontier ballots and re-tallies recorded assignments over the eight non-frontier voters. The reported fraction of changed accepted assignments is 8.8%. Non-frontier voters cast YES on 82.3% of their ballots concerning accepted frontier critical-path assignments.

For a frontier claimant, the reduced electorate contains eight voters and requires six YES votes. For a non-frontier claimant, self-vote exclusion leaves seven voters and also requires six YES votes. The re-tally measures sensitivity of recorded decisions to this electorate change. It is not a counterfactual execution: removing frontier participation could alter proposals, justifications, and later workflow states.

Checkpoint authorship. Default checkpoint voting includes the output producer and schema contributors when they are voters. All evaluate the output against previously ratified criteria, after mechanical interface and dependency checks. Their participation can nevertheless introduce correlated preferences.

Excluding the accepted schema proposer and co-revisers changes 1.8% of checkpoint verdicts and yields 71.8 ± 1.3 on MAB, compared with 72.1 ± 1.2 for the default. Schema authors vote YES on 77.6% of checkpoints, compared with 76.1% for other voters, a difference of 1.5 percentage points.

These observed differences do not establish absence of bias or statistical equivalence. Schema-author exclusion is also different from producer exclusion. Any such policy must record its electorate and recompute the threshold. An empty electorate produces an explicit failure; it cannot authorize a checkpoint.

Ballot visibility and dependence. A `VOTE-OUTCOME` is broadcast after tallying, but the replicated Vote Ledger may expose ballots that have already arrived. Randomizing arrival order yields 72.0 ± 1.2 on MAB, and hiding interim voter identities yields 71.9 ± 1.3, compared with 72.1 ± 1.2 under the default.

These interventions produce small observed score differences. Hiding identities does not necessarily hide ballot content, and neither result establishes independent voting.

For YES versus non-YES ballots, excluding decisions on which either member of a pair abstains, the reported mean pairwise ϕ coefficient is 0.21. Independently shuffling each voter's ballots within benchmark and protocol stage yields 0.08 under the reported null.

The difference may reflect shared evidence, model similarity, or social influence. The statistic alone does not identify the cause. Uncertainty for this comparison and a numerical justification-diversity measure are not reported, so no significance or diversity-stability claim is made.

A11.7 Committee voting and participation-efficient variants

Flat voting exhibits a diversity–agreement trade-off in the evaluated pool: performance peaks at twelve voters and decreases for larger tested pools. Increasing membership raises the absolute threshold, but score degradation does not follow from threshold arithmetic alone. It also depends on the capabilities, agreement patterns, and processing costs of the added voters.

The hierarchical extension organizes voters into committees at epoch start, aiming to retain capability coverage and distribute frontier models across committees. Local assignments, checkpoints, and repairs use the designated committee; workflow-wide frame and schema decisions retain full-electorate voting.

Table 40: Flat and hierarchical voting on MAB. NoC denotes the NO-CONSENSUS rate. The reported values are point estimates.

Voters	Flat score	Flat NoC	Hierarchical score	Hierarchical NoC
16	71.4	4.2%	71.7	3.4%
20	70.2	7.8%	71.0	4.8%
24	68.9	11.3%	70.3	6.5%

Table 41: Local voting-subset experiment on MAB. Scores are mean $\pm$ 95% CI over five seeds. Message reductions are relative to 312 coordination messages.

Local voters	MAB	Coordination messages	Reduction
12	72.1 ± 1.2	312	—
8	71.9 ± 1.3	221	29.2%
6	70.8 ± 1.4	174	44.2%

For a local assignment to r, the electorate is $C_e(D) \setminus \{r\}$, where $C_e(D)$ is the designated committee. For other local decisions it is $C_e(D)$. The threshold remains $q = 3/4$, and empty electorates cannot authorize decisions.

The hierarchical score advantage increases from 0.3 points at sixteen voters to 1.4 points at twenty-four voters. At twenty-four voters, NoC decreases by 4.8 percentage points, but the hierarchical score of 70.3 remains below the twelve-voter flat score of 72.1. Detailed committee memberships and uncertainty estimates are not reported here; the comparison is therefore exploratory.

Committee voting changes the scope of authorization. A local certificate establishes a supermajority of its committee, not of the entire pool. Fixed committee membership preserves the denominator for that decision class, but generalizes the default full-electorate policy.

A complete deployment must ratify the mapping from roles and decision classes to committees, record it in decision certificates, and define cross-committee escalation. A conservative extension retains full-electorate approval for workflow-wide objects, membership changes, cross-committee repairs, and final completion. Those additional deployment rules are not independently validated by the aggregate results in Table 40.

A related experiment retains all twelve models as available executors while reducing the electorate for local decisions. The designated subsets are fixed by decision class at epoch start; participation changes do not alter an open decision's membership or denominator. Global frame and schema decisions retain the full electorate.

Eight-voter local decisions reduce coordination messages by 29.2% with a 0.2-point difference in mean score. Six-voter decisions reduce messages by 44.2% with a 1.3-point score reduction.

Message counts do not directly establish monetary or latency savings, and overlapping confidence intervals do not establish statistical equivalence. Local acceptance remains authorization by the designated subset.

A11.8 Selecting a voter pool and interpreting coordination cost

For the tested model families, a capability-diverse pool of eight to twelve voters is a reasonable starting heuristic. A small pilot of approximately twenty representative tasks can identify missing capabilities, recurring disagreement, and major cost sources. It cannot precisely estimate a low failure rate: one failure in twenty tasks already corresponds to 5%. A target NoC rate near 5% is therefore an operational heuristic, not a statistically established acceptance threshold.

Quorum slack must be assessed separately for each decision class. At $q = 3/4$, an ordinary eight-voter decision has threshold six and slack two. Assignment to a voter within that committee leaves seven eligible voters, still requires six YES votes, and has slack one.

Nine committee members provide assignment slack two after claimant exclusion: eight eligible voters require six YES votes. These calculations concern possible acceptance when all remaining eligible voters affirm the proposal; they do not guarantee end-to-end recovery from that number of failures.

The overhead measurements in Appendix A8.3 report coordination-token shares of approximately 23%, with coordination latencies of 23.4 seconds on MAB, 21.8 seconds on τ-bench, and 34.6 seconds on MemA. Total run costs are \$1.21, \$1.08, and \$1.74, respectively.

Token share and monetary share are different in a heterogeneous pool because models have different input, output, and, where applicable, reasoning-token prices. The token ratios therefore do not establish an exact 23% monetary overhead.

On MAB, the reported score-per-dollar ratios are 59.6 for URCP, 59.5 for A2A, and 59.2 for MCP. URCP achieves score 72.1 at \$1.21 per run, compared with 54.7 at \$0.92 for A2A and 52.1 at \$0.88 for MCP. These measurements indicate similar observed score per dollar at a higher achieved score and higher realized expenditure. Equal budget caps do not imply equal spending, and these points do not establish an optimal cost–quality frontier.

Message accounting. Logical messages must be distinguished from delivered copies. Let n be the number of roles, P the number of eligible participants, and m the voter count. With at most K_5 claimant decisions per role, one assignment invocation requires at most

$$O(nm \min\{P, K_5\})$$

ballot messages, excluding ballot revisions and broadcast fan-out. The complete run additionally includes assignment invocations caused by repair.

For subtask formation, if each of K_3 rounds contains at most b candidate decisions and one coverage decision, the corresponding ballot count is

$$O(K_3(b+1)m)\,.$$

If every voter may propose up to n distinct candidates per round, then $b = O(mn)$ and the candidate-voting term becomes $O(K_3 nm^2)$.

These counts assume at most one effective ballot transmission per voter per decision. Allowing multiple ballot revisions introduces the corresponding bounded multiplicative factor. Physical delivery and log-replication costs add broadcast fan-out. Eligibility checks scale with candidate participants, while DAG checks depend on vertices, edges, and interface-validation work. Reducing the local electorate lowers ballot traffic but does not eliminate global state replication.

A11.9 Deployment scope, privacy, and reproducibility

The three pilots in Appendix A9 provide operational observations over two to four weeks. They do not establish production readiness across organizations. Their human-rated quality improvements are specific to the pilot tasks and prior tooling. Without a common rubric, sample counts, and uncertainty estimates, those improvements are not standardized benchmark effects.

One support pilot discontinued use for cost reasons. Across the pilots, the reported 99th-percentile run cost is 3.8 times the median, indicating that mean cost alone is insufficient for capacity planning.

Practical constraints include API limits, model-version changes, cold starts, resource failures, privacy policies, and mechanical-runtime availability. Revision and epoch caps bound repeated protocol work, but monetary limits additionally require per-call and per-run accounting. Token, wall-clock, and monetary budgets, durable logs, resource health checks, and escalation procedures are deployment mechanisms, not consequences of supermajority voting.

Short fixed pipelines, sub-second response requirements, or tasks dominated by one capable model may favor simpler coordination policies.

Encrypted handoffs. Encryption is an optional extension to authenticated messaging. A digest establishes integrity relative to a referenced artifact; it does not prove that hidden plaintext satisfies a semantic acceptance criterion. A redacted summary likewise requires a justified connection to the hidden content. Public evidence must support the stated checkpoint criterion, or validation must rely on an explicitly specified trusted or cryptographic mechanism.

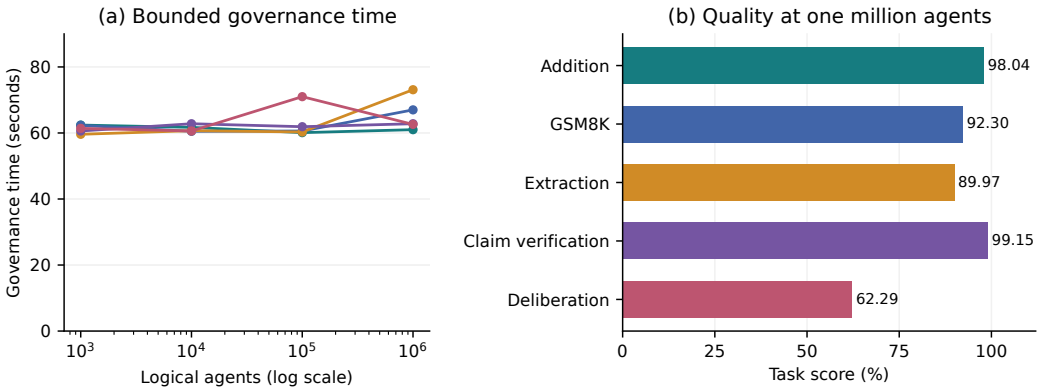


Figure 15: URCP at million-agent scale. (a) Seven-stage governance time, excluding worker execution, for five tasks under a fixed three-voter electorate. (b) Task scores at 10^6 logical agents. Addition uses exact sums after one retry; GSM8K uses per-sample accuracy; extraction uses exact records; claim verification uses agreement with a Lean catalogue; deliberation uses round-1 gold-answer agreement. Each task/population cell is one run of this scale-out configuration.

Encryption also requires key management beyond signing-key registration. For ChaCha20–Poly1305, a nonce must not be reused for distinct encryption operations under the same key [25]. Retransmitting the same stored ciphertext is different from encrypting a revised plaintext. A revised output requires a fresh encryption envelope and corresponding evidence; unchanged role and upstream hashes alone are not a sufficient nonce-generation rule.

Recipient reassignment requires authorized re-encryption or an explicit failure path if re-encryption is unavailable. These requirements do not establish general confidentiality or semantic-correctness guarantees for the exploratory encrypted-handoff observations.

A11.10 Scaling to One Million Logical Agents

URCP supports a governed population of 10^6 logical agents while preserving task quality and audit integrity. Here, a logical agent is a role-bound model invocation sharing served model instances. This scale-out configuration fixes three voters (Gemma 4 E4B, Gemma 4 31B, and Gemma 4 26B-A4B; $q = 0.75$), uses Gemma 4 E4B workers on NVIDIA RTX Pro 6000 GPUs, and executes the seven governance stages once per run with sharded worker execution.

Stable governance and task quality. Across five tasks, governance takes 59.6–73.1 s as the population grows from 10^3 to 10^6 agents, excluding worker execution (Figure 15). At one million agents, scores reach 98.04% for addition, 92.30% for GSM8K, 89.97% for structured extraction, 99.15% for claim verification, and 62.29% for deliberation. For every task, the score range across 10^4, 10^5, and 10^6 agents is below 0.5 percentage points. In the nested society game, one million agents achieve 54.0% final-round accuracy against 25% chance, with 59 s of governance.

Verifiable execution at scale. All six audit checks pass at one million agents: population coverage, unique shard commits, planned claims, complete row accounting, cryptographic verification and chain replay, and role/contract binding. In the society game, all one million rows are verified in each of four rounds. Together, these results demonstrate stable governance, preserved task quality, and auditable execution at million-agent scale under a fixed electorate.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The abstract and introduction (Sections 1–2) clearly state URCP's contributions: an orchestrator-free protocol with a single supermajority decision rule Φ_q, a five-level repair hierarchy, formal termination guarantees, and empirical gains. These claims are substantiated by the protocol specification (Section 3, Appendix A1), the formal properties (Appendix A1.16), and the experimental results (Section 4 and Appendices A2–A9).

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: Limitations are discussed throughout the appendices. Appendix A3.10 acknowledges that the trace analysis patterns are descriptive and may not generalize across pools/benchmarks. Appendix A6 discusses open problems with secure handoffs. Appendix A9.3 details practical failure modes (cost variability, vendor drift, non-determinism in repair triggers, cold-start). Appendix A9.5 explicitly states settings where URCP is not the right choice. Appendix A7.1 discusses adversarial robustness limitations at $k \geq 4$.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best

judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [Yes]

   Justification: Assumptions A1.1–A1.4 (authenticated communication, bounded delivery, synchronous phases, fixed voting set per epoch) are stated explicitly in Appendix A1.1. The formal properties (no semantic orchestrator, supermajority legitimacy, capability soundness, auditability, finite termination, and within-epoch deadlock freedom) are stated and justified in Appendix A1.16, with supporting algorithmic specifications in Appendices A1.2–A1.15 and the exhaustive deadlock-resolution table (Table 8).

   Guidelines:

   - The answer [N/A] means that the paper does not include theoretical results.
   - All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
   - All assumptions should be clearly stated or referenced in the statement of any theorems.
   - The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
   - Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
   - Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: Section 4.1 specifies benchmarks, participant pool, scaling protocol, and implementation. Appendix A1 provides the complete protocol specification with message schemas and algorithms (including Algorithm 6 in Appendix A10). Appendix A4.1 lists the full participant pool and Specialization Registry, Appendix A4.2 details the runtime implementation and on-disk layout, Appendix A4.3 describes baseline configurations, and Appendix A4.4 specifies failure-timing conventions. Hyperparameters (Table 2) are fixed and sensitivity analysis is provided in Appendix A7.3.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.

- While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
  (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
  (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
  (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
  (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [No]

   Justification: The paper provides a complete protocol specification (Appendix A1) and algorithm pseudocode (Algorithm 6 in Appendix A10) sufficient for independent reimplementation, but anonymized code and data are not released alongside the submission. All evaluations use publicly available benchmarks (MultiAgentBench, τ-bench, MemoryAgentBench, GAIA, REALM-Bench) and contemporary models accessible through public APIs or open weights, and Appendix A5 documents that the trace-analysis pipeline is deterministic given the audit logs. Code release is planned upon publication.

   Guidelines:

   - The answer [N/A] means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
   - The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
   - At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
   - Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes]

   Justification: Section 4.1 specifies inference settings (temperature 0, top-p=1, 4,096 output tokens, constrained JSON decoding), serving infrastructure (vLLM with 4-GPU tensor

parallelism on 8×A100), per-run budgets (500K tokens, 600s wall clock), and the validation split (10% of MAB) used for hyperparameter selection. Table 2 lists all timeouts, budgets, and the decision threshold $q = 3/4$. Appendix A4 provides further details on the participant pool, runtime implementation, and baseline configurations.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
- The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [Yes]

   Justification: All main experimental tables (Tables 3, 4, 5, 9, and the team-composition panel of Figure 1) report mean ± 95% confidence intervals over 5 seeds. Appendix A3 reports per-voter quantities with 10,000-resample bootstrap confidence intervals at the run level. Tables 10 reports χ^2 statistics with Holm-corrected p-values. The factors of variability (random seeds across 5 runs per condition) are explicitly stated.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
   - The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
   - The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
   - The assumptions made should be given (e.g., Normally distributed errors).
   - It should be clear whether the error bar is the standard deviation or the standard error of the mean.
   - It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
   - For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
   - If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: Section 4.1 specifies that open-weight models are served via vLLM with 4-GPU tensor parallelism on a single 8×A100 (80 GB) node, and API models are rate-matched to local throughput. Per-run budgets (500K tokens, 600s wall clock) are stated. Appendix A8 provides a detailed analysis of protocol overhead, including empirical message counts, coordination/execution token splits, latencies, and per-run dollar costs across all benchmarks (Tables 30–33). Appendix A4.2 describes the on-disk runtime layout.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.

- The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
- The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
- The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: The research uses publicly available benchmarks and contemporary language models accessed through standard APIs or open-weight releases. No human subjects are involved, no private or sensitive data is collected, and the work does not introduce new datasets or models that pose dual-use risks. The protocol's auditability and explicit `NO-CONSENSUS` verdicts (Appendix A3.8) further support transparency in deployment.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [Yes]

    Justification: The paper discusses both. Positive impacts: URCP's auditable coordination, supermajority governance, and explicit `NO-CONSENSUS` verdicts (Appendix A3.8) support transparency, accountability, and safety in multi-agent deployments—particularly valuable in enterprise, safety-critical, and multi-vendor settings (Appendix A9.2). Negative impacts and limitations: Appendix A9.3 discusses cost variability, vendor coordination risks, and privacy/data-handling boundaries; Appendix A6 discusses open problems with secure handoffs; Appendix A9.5 identifies settings where URCP is not appropriate.

    Guidelines:

    - The answer [N/A] means that there is no societal impact of the work performed.
    - If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
    - Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
    - The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
    - The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.

- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

    Answer: [N/A]

    Justification: The paper introduces a coordination protocol, not a new pre-trained model, dataset, or generative system. URCP composes existing publicly available models and benchmarks; no new high-risk artifacts are released. The protocol itself includes auditability mechanisms (the audit log AL, Appendix A1.2) and explicit `NO-CONSENSUS` verdicts (Appendix A3.8) that aid responsible deployment.

    Guidelines:

    - The answer [N/A] means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: All benchmarks are properly cited with their original references. Baseline systems are credited and used per their published settings.

    Guidelines:

    - The answer [N/A] means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
    - If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
    - For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
    - If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [N/A]

Justification: The paper does not release new datasets, model checkpoints, or other artifacts at submission time. The contribution is a coordination protocol whose complete operational specification is documented in Appendix A1 (message schemas, eligibility rules, voting mechanics, repair levels, timeouts, formal properties) at a level of detail sufficient for independent reimplementation, with the full algorithm provided in Algorithm 6 (Appendix A10).

Guidelines:

- The answer [N/A] means that the paper does not release new assets.
- Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
- The paper should discuss whether and how consent was obtained from people whose asset is used.
- At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

Answer: [N/A]

Justification: The paper does not involve crowdsourcing or research with human subjects. All experiments are automated benchmark evaluations using language models and executable resources. The pilot deployments mentioned in Appendix A9.2 are anonymized post-hoc operational summaries, not human-subjects research.

Guidelines:

- The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
- Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
- According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

Answer: [N/A]

Justification: The paper does not involve research with human subjects, so IRB approval is not applicable. All experiments are automated multi-agent benchmark evaluations.

Guidelines:

- The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
- Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

Answer: [Yes]

Justification: LLMs are central to the core methodology of this research as the vote-capable participants in URCP. The complete pool of twelve LLMs (GPT-4.1, GPT-4.1 mini, Claude Opus 4, Claude Sonnet 4, Gemini 2.5 Pro, Gemini 2.5 Flash, Llama 4 Maverick, Qwen3-235B-A22B, DeepSeek-R1, Mistral Medium 3, Phi-4-reasoning, Command A) is documented in Section 4.1 and Appendix A4.1 (Table 17), along with their declared capabilities, modalities, and roles within the protocol.

Guidelines:

- The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
- Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.