When 20 Agents Fail to Sort: The Distributed Sorting Benchmark for Scalable Multi-Agent Systems

Xin Yang[1] **Junhao Wang**[2] **Bintao Tang**[3]
Xuxin Cheng[4] **Cao Liu**[4] **Ke Zeng**[4] **Wenyuan Jiang**[5*]

[1] School of Mathematical Sciences, Zhejiang University, China
[2] Chinese University of Hong Kong, China [3] Tongji University, China
[4] Meituan LongCat Interaction Team [5] ETH Zürich, Switzerland

Abstract

Current LLM-based multi-agent systems remain fragile under scaling, even on algorithmically trivial tasks. We introduce MAS-BENCH, a distributed-sorting benchmark that isolates coordination under explicit communication constraints: each agent observes only a local segment and must collectively produce a globally consistent order via broadcasting, peer-to-peer messaging, or a shared key-value store. Across LLM-based agents, success drops sharply as the number of agents grows, exposing persistent failures in shared state, convention alignment, and consistent termination. To mitigate these breakdowns, we propose CAMOC, a lightweight, drop-in proof-of-concept built on collaboration-aware information sharing, early global metadata exchange, and single-commit verification. CAMOC substantially improves coordination success and efficiency across backends, with the largest gains under shared-state interaction. Overall, MAS-BENCH provides a diagnostic benchmark and CAMOC offers a practical step toward more reliable large-scale LLM collaboration, highlighting a gap between individual reasoning and collective correctness.

1 Introduction

Large language models (LLMs) have become strong general-purpose reasoners and tool users, enabling agentic systems that iteratively perceive state, decide actions, and interact with external environments (Yao et al., 2023b; Shen et al., 2023; Shinn et al., 2023; Yang et al., 2025; Wu et al., 2025). Motivated by empirical scaling laws and the broader "scaling is (often) better" narrative (Kaplan et al., 2020; Hoffmann et al., 2022; Li et al., 2024), recent work increasingly explores multi-agent systems (MAS) as an orthogonal axis of scaling: instead of only scaling a single model, we scale *a society* of LLM agents that communicate to solve tasks.

*Corresponding Author. Email: wenyjiang@ethz.ch

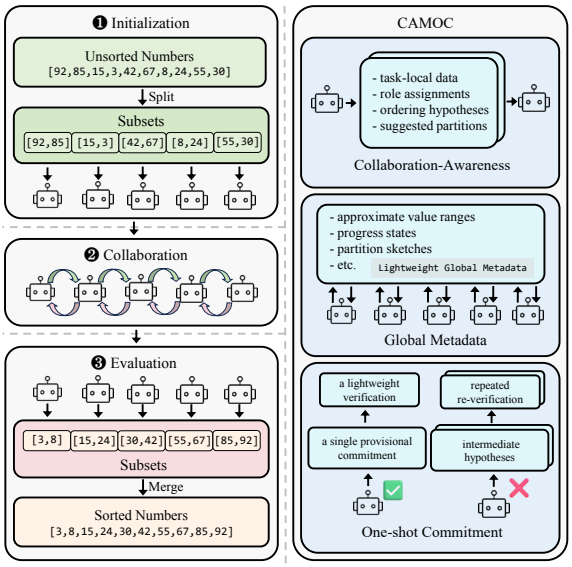

Figure 1: **Overview of MAS-BENCH and CAMOC.** Global information is partitioned across N agents, each observing only local data. Agents must communicate through the provided settings to reconstruct a globally consistent ordering.

However, many reported MAS successes hinge on substantial human scaffolding. Frameworks typically predefine roles (planner/critic/executor), decompose tasks manually or via templates, and constrain interaction with engineered pipelines or "standard operating procedures" (Li et al., 2023; Hong et al., 2024; Qian et al., 2024). While effective, these designs blur the line between genuine autonomous coordination and workflow engineering: improvements may come from the protocol designer rather than the agents' ability to establish shared conventions, reconcile inconsistent beliefs, and converge to a globally correct outcome under partial information.

This paper asks a deliberately sharp question: *Can current LLM-based agents reliably coordinate to solve a simple, structured multi-agent problem when we remove most human scaffold-*

Findings of the Association for Computational Linguistics: ACL 2026, pages 34002–34021

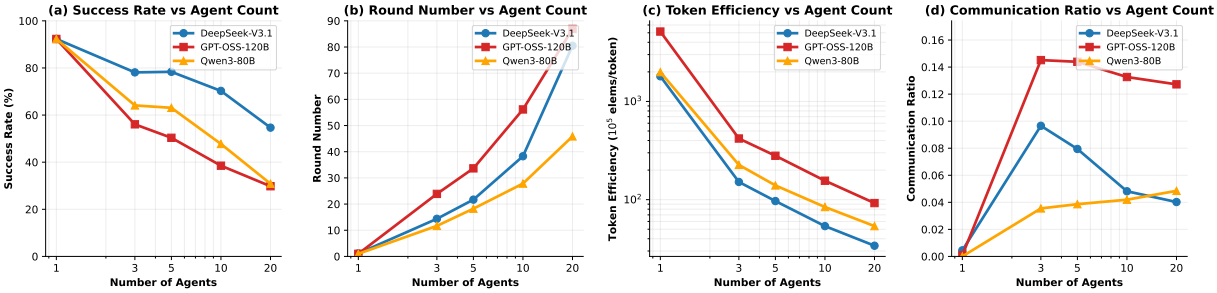

Figure 2: **Scaling trends with increasing agent count.** Each panel reports the mean (across K, input difficulties, and communication modes) as the number of agents N increases: (a) success rate (SR), (b) maximum round number (R), (c) token efficiency (TE), and (d) communication ratio (CR). All three models exhibit a consistent scaling pattern: SR degrades and R grows with larger N, while TE drops sharply, indicating growing coordination overhead under fixed per-agent capacity.

ing and enforce symmetric communication? We focus on a task that is algorithmically elementary but coordination-intensive: *distributed sorting*, inspired by classical distributed data processing (e.g., MapReduce-style partitioning and global ordering) (Dean and Ghemawat, 2008). Each agent sees only a local segment and must communicate through a specified substrate to output a globally consistent sorted partition. Because local sorting is trivial, failures largely reflect breakdowns in inter-agent agreement: boundary inference, shared-state consistency, termination, and "common ground" maintenance.

We introduce MAS-BENCH, a benchmark that isolates multi-agent coordination under explicit communication primitives. MAS-BENCH spans 3 paradigms (Broadcasting, P2P Messaging, and a Shared K-V Store(Wu et al., 2023)) and varies agent scale N, segment size K, and input order structure (§3.2–§3.3). Across state-of-the-art LLMs, we observe a clear coordination scaling effect: increasing N and/or K sharply degrades success while increasing interaction overhead, with the most severe failures arising under shared-state interaction (§5.1–§5.4).

Motivated by recurrent failure patterns, we propose CAMOC, a lightweight drop-in coordination layer. CAMOC structures both what agents communicate and when they commit via collaboration-aware messages, early exchange of coarse global metadata, and a one-shot commitment stage (§4.1). Without retraining, CAMOC substantially improves robustness and token efficiency, with the largest gains under the Shared K-V backend (§5.1–§5.4).

In summary, our contributions are threefold:

- **MAS-BENCH.** A scalable benchmark that isolates multi-agent coordination difficulty in distributed sorting, with deterministic correctness and symmetric communication interfaces.
- **Empirical diagnosis.** A systematic study across models, backends, and scales that reveals persistent coordination bottlenecks and unfavorable scaling trends.
- **CAMOC.** A drop-in coordination layer that improves collaboration without retraining underlying LLMs, yielding success-rate gains up to 40%.

2 Related Work

LLM agents and tool use. LLM agents typically interleave free-form reasoning with environment actions (Yao et al., 2023b; Ji et al., 2025). Tool-augmented controllers extend this loop by delegating subtasks to external models/APIs (Shen et al., 2023; Schick et al., 2023; Patil et al., 2024; Ji et al., 2026). Early "autonomous agent" systems popularized iterative plan-act-refine patterns for long-horizon tasks (Significant-Gravitas, 2026; Nakajima, 2026), while later work improves stability via reflection, self-correction, and memory (Shinn et al., 2023; Madaan et al., 2023). Interactive web/-software agents further stress tool grounding and long-context state tracking (Zhou et al., 2024a; Jimenez et al., 2024; Yao et al., 2023a).

LLM-based multi-agent systems. A dominant paradigm scales agentic behavior through role decomposition and scripted interaction protocols, e.g., role-playing dialogues (Li et al., 2023), pipelines for code and project management (Hong et al., 2024; Qian et al., 2024; Cheng et al., 2025), and programmable multi-agent frameworks (Wu et al., 2023). Deliberation-style protocols (e.g., debate or

structured critique) aim to improve correctness at inference time but often rely on carefully designed turn-taking (Du et al., 2024; Liang et al., 2024) together with fine-tuning (Yang et al., 2026; Wu et al., 2026) for better overall performance. More broadly, the MAS literature highlights how topology, information flow, and agreement mechanisms shape collective outcomes (Wooldridge, 2009; Shoham and Leyton-Brown, 2009; Olfati-Saber et al., 2007), and emergent communication in learning-based MAS offers a complementary lens on coordination under constrained channels (Lazaridou et al., 2017; Foerster et al., 2016).

Benchmarks for agents and multi-agent coordination. Agent benchmarks commonly evaluate tool use and long-horizon interaction (often with subjective scoring or task-specific orchestration) (Liu et al., 2024; Zhou et al., 2024a). Multi-agent benchmarks and simulators emphasize conversational interaction or open-ended social behavior (Chen et al., 2024; Zhou et al., 2024b; Park et al., 2023). In contrast, relatively few benchmarks isolate collective correctness under symmetric and explicit communication primitives, while also measuring coordination cost under scaling.

3 The Multi-Agent Sorting Benchmark

This section introduces MAS-BENCH, a benchmark for measuring how well LLM agents coordinate to solve a simple but coordination-intensive distributed sorting task, and defines metrics that capture both effectiveness and cost.

3.1 Problem Setting

A system contains N agents $\{a_0, \ldots, a_{N-1}\}$. Agent a_i receives a local segment s_i of length K, and the global input is their concatenation. Agents interact only through a communication environment E and must output sorted segments $\hat{s}_i$ such that concatenating $\hat{s}_0, \ldots, \hat{s}_{N-1}$ yields a globally non-decreasing array.

Each instance is a tuple

$$t = (N, K, E, l), \tag{1}$$

where $|l| = N \times K$ and elements follow a total order. Figure 1 illustrates an example with N=5 and K=2.

3.2 Communication Paradigms

We study three symmetric communication paradigms that cover common coordination substrates in distributed systems as is shown in Figure 3. To isolate coordination from network artifacts, all actions are reliable and become visible within one round.

1. **P2P messaging.** Agents send directed messages to any other agent via `send_to(msg, id)`/`recv()`.
2. **Broadcasting.** Messages are visible to all agents via `broadcast(msg)`/`recv()`.
3. **Shared K-V store.** Agents coordinate by reading/writing shared keys via `read(k)`/`write(k, v)`; writes overwrite existing values.

In all settings, each agent runs a tool-calling loop: it observes the current shared state (messages and/or keys), generates an action, and updates the environment. Separate primitives implement each paradigm to compare coordination difficulty under different substrates. The round-level semantics, the full LLM-visible command surface of each paradigm, and the harness-side side effects on submission are specified formally in Appendix F (Algorithm 2 and Table 5).

3.3 Input Order Structure

For fixed (N, K, E), difficulty is mainly controlled by the permutation structure of l. We quantify disorder by inversion count

$$\mathrm{Inv}(l) = \left|\{(p, q) \mid 0 \le p < q < |l|,\ l[p] > l[q]\}\right|, \tag{2}$$

and normalized inversion rate

$$\rho(l) = \frac{\mathrm{Inv}(l)}{\binom{NK}{2}} \in [0, 1]. \tag{3}$$

Higher ρ typically implies more cross-segment constraints and thus more coordination.

We generate five input families: `asc`, `near_asc`, `random`, `near_desc`, `desc`. `asc`/`desc` are globally sorted / reverse-sorted, `random` is fully shuffled, and `near_asc`/`near_desc` freeze 80% of positions and randomly permute the rest to create controlled partial disorder.

3.4 Evaluation Metrics

We measure correctness and coordination cost. Let $\hat{s}_i$ be agent i's submitted segment, y_i the ground-truth segment, and $T_i^{\text{in}}/T_i^{\text{out}}$ the total input/output tokens consumed by agent i during the run.

Success Rate (SR).

$$SR = \frac{1}{N} \sum_{i=1}^{N} \mathbb{1}[\hat{s}_i = y_i]. \tag{4}$$

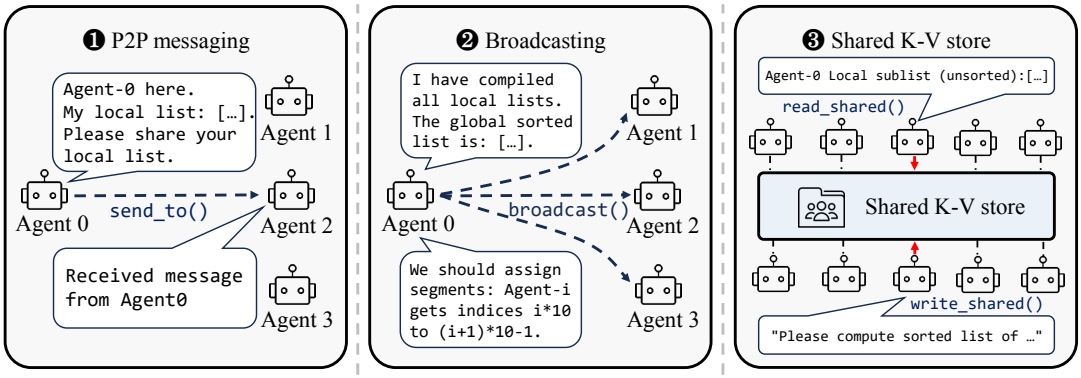

Figure 3: **Communication paradigms studied in MAS-BENCH.** All paradigms provide reliable delivery but differ in how global information is exposed and synchronized.

An instance is successful iff $SR{=}1$.

Round Number (R).

$$R = \max_i r_i, \tag{5}$$

where r_i is the number of tool-calling rounds taken by agent i.

Token Efficiency (TE).

$$TE = \frac{NK}{T_{\text{total}}} \times 10^5, \quad T_{\text{total}} = \sum_{i=1}^{N} (T_i^{\text{in}} + T_i^{\text{out}}). \tag{6}$$

Larger TE indicates fewer tokens per sorted element (scaled by 10^5 for readability).

Communication Ratio (CR).

$$CR = \frac{\sum_{i=1}^{N} T_i^{\text{com}}}{T_{\text{total}}}, \tag{7}$$

where T_i^{com} counts the tokens attributed to communication (messages or K-V updates) for agent i. Higher CR indicates more of the token budget is spent on inter-agent coordination.

Together, SR captures *effectiveness*, R and TE capture *coordination cost*, and CR characterizes *communication intensity*.

4 CAMOC

Coordinating LLM-based agents is difficult not because agents cannot solve their local subproblems, but because global consistency must emerge from fragmented, turn-based interaction. In this setting, mismatched assumptions about roles, partitions, and progress easily accumulate, while long multi-round dialogues often dilute context and delay commitment. CAMOC is a lightweight coordination layer that directly targets these structural issues by shaping *what* agents communicate and *when* they commit.

This section introduces three design principles (§4.1), which are Collaboration-Awareness, Global Metadata, and One-shot Commitment, and illustrates each with concrete interaction snippets (§4.2–§4.4). In the experiments (Table 1), augmenting the same base protocol with CAMOC improves coordination success and efficiency across models and backends.

4.1 Design Principles

CAMOC is guided by three principles operating at two levels. At the **message level**, Collaboration-Awareness and Global Metadata specify *what* agents communicate: beyond local facts, agents share coordination-relevant intent and coarse global signals to reduce early ambiguity. At the **interaction level**, One-shot Commitment specifies *when* agents commit: a single provisional submission followed by lightweight verification, avoiding multi-round re-validation that amplifies context drift. Together, these principles replace ad-hoc negotiation with structured coordination.

1. **Collaboration-Awareness.** Agents communicate not only data but also assumptions, tentative roles, and partition hypotheses, reducing downstream clarification.
2. **Global Metadata.** Agents exchange lightweight global signals (e.g., value ranges or progress states) early to support consistent reasoning under partial views.
3. **One-shot Commitment.** Agents make a single provisional commitment followed by minimal verification; repeated re-validation often degrades accuracy due to context dilution.

Collectively, these principles enable more stable

and efficient coordination.

4.2 Collaboration-Awareness

A common intuition in multi-agent prompting is to have each agent focus solely on its own subtask, avoiding interference with the tasks of other agents. In other words, the approach is to share data rather than instructions. However, our empirical analysis reveals a fundamental conflict between this strategy and the nature of LLM-driven agents.

Therefore, in the CAMOC framework, we encourage agents to not only delegate tasks but also offer suggestions or seek advice from others to quickly reach consensus on certain aspects, thereby facilitating collaborative, awareness-based information sharing.

The following case illustrates the difference between collaboration-aware vs. collaboration-unaware information sharing. Specifically, the former not only shares information but also issues a request, "Please share your local list." In contrast, the latter merely shares information without guiding or concerning itself with the actions of others. Although the former introduces considerable redundancy, it helps implicitly resolve discrepancies and asymmetric information in the collaborative process, ultimately reducing conflicts and making it easier to reach alignment.

Collaboration-Aware

Agent-0 here. My local list: [131, 765, 796, 264, 481, 649, 808, 227, 330, 366]. **Please share your local list.**

Collaboration-Unaware

Agent-1 here. My local list: [971, 878, 160, 490, 619, 852, 527, 309, 823, 438]

4.3 Global Metadata

Previous research typically advocates for on-demand communication (Hu et al., 2024), where agents exchange information only when explicitly required. In contrast, CAMOC employs proactive global signal transmission: agents share coarse-grained global metadata early on, even if it is not immediately needed. This includes approximate value ranges, tentative partitions, and progress indicators. While such information may be redundant or partially inaccurate, it accelerates the flow of information between agents, enabling them to obtain as much relevant data as possible with fewer interaction rounds.

The following case illustrates our approach of attempting to exchange global data, where an agent informs the other agents of its perceived global ordering list. This provides additional information to the agents, potentially benefiting the overall task completion.

Global Metadata

I have compiled **all local lists**. The global sorted list is: [2, 20, 32, 100, ..., 988, 990]. We should assign segments: Agent-i gets indices i*10 to (i+1)*10-1.

In turn-based multi-agent systems, the cost of uncertainty and hallucinations outweighs the cost of redundancy. The principle of minimal information sharing forces agents to repeatedly query missing contextual information, significantly increasing communication overhead. Case below shows that missing part of global metadata leads to extra communications.

Missing Global Metadata

Please share your local list xs[8] urgently. We need it for global sorting.
Please share your local list xs[4] urgently. We need it for global sorting.

4.4 One-shot Commitment

A common approach to improve accuracy is to repeatedly validate intermediate results, inspired by inference-time self-checking. However, our experiments show that in multi-agent settings, multi-round validation often *reduces* accuracy for LLM agents. CAMOC therefore adopts a one-shot commitment strategy.

After coordination, agents make a single provisional submission, followed by one explicit validation step to detect major inconsistencies. Repeated re-validation lengthens context and amplifies discrepancies across rounds, making agents susceptible to context drift and self-contradiction. As a result, additional checks frequently negate their intended benefits.

A single, well-defined commitment balances accuracy and stability, shortens interaction cycles, and avoids prolonged "almost correct" states with

Model & Setting		Broadcasting				P2P Messaging				Shared K-V Store			
		SR%↑	R↓	TE↑	CR%	SR%↑	R↓	TE↑	CR%	SR%↑	R↓	TE↑	CR%
DeepSeek-V3.1	Base	80.8	9.3	2.63	3.8	76.8	38.8	2.50	8.5	38.2	38.2	3.49	3.2
	CAMOC	**89.9**	10.2	6.43	4.3	**82.3**	37.6	6.35	9.9	**84.7**	45.3	5.25	3.7
	Abl. `submit`	89.7	9.1	6.64	3.6	77.4	37.9	6.76	7.7	81.6	43.2	5.02	3.7
	Abl. `message`	84.8	9.4	2.69	4.6	76.7	38.6	2.52	10.3	70.6	49.7	1.88	3.5
Qwen3-Next-80B-A3B	Base	70.9	12.3	2.72	2.4	**57.4**	25.3	2.69	6.1	33.7	27.6	4.74	1.2
	CAMOC	**75.8**	8.0	7.15	2.1	50.6	27.0	6.23	7.2	**67.2**	24.0	6.67	1.0
	Abl. `submit`	66.6	11.6	7.24	2.8	49.6	26.5	7.49	6.2	69.2	24.6	6.83	1.3
	Abl. `message`	74.3	8.3	2.72	2.2	55.8	26.6	2.74	7.6	70.3	22.4	2.84	1.2
GPT-OSS-120B	Base	55.6	16.6	8.09	7.8	53.6	37.1	8.73	22.6	25.6	64.1	10.75	4.3
	CAMOC	**69.8**	15.7	15.92	7.3	61.9	45.9	17.28	22.5	**56.5**	55.1	14.36	4.0
	Abl. `submit`	61.2	16.2	15.92	8.0	**65.4**	36.1	17.89	22.2	54.4	61.6	14.19	4.2
	Abl. `message`	59.1	15.5	8.19	8.2	53.5	45.4	8.79	23.0	52.1	50.8	7.59	4.5

Table 1: **Per-model results on MAS-BENCH under three communication backends.** Numbers are averaged over all task instances. Compared to the Base protocol, CAMOC improves success and token efficiency across models, with especially large gains under the Shared K-V backend, where naive coordination frequently fails. Ablations isolate the contribution of individual components (§5.4). Best results in **bold**.

high execution cost.

Dual-Commitment

I need to check if my previous submission of [0, 1, 4, 12, 23, 29, 32, 34, 39, 47] is correct for the current task. To do this, I must communicate with other agents to reconstruct the global order.

Overall, CAMOC shows that effective collaboration depends less on minimizing communication than on structuring commitment and verification.

Appendix F provides a formal specification of the CAMOC orchestrator (Algorithm 3), the verbatim behavior-list addendum that implements Collaboration-Awareness and Global Metadata, and the rerun addendum that implements One-shot Commitment, so that each design principle corresponds to an identifiable prompt fragment and control-flow branch.

5 Experiments & Analysis

5.1 Experiment Setup

Each agent runs as an independent process interfacing with an LLM, and interacts with the environment through one of three backends: (i) *Broadcasting* (one-to-all announcements), (ii) *P2P Messaging* (explicit directed messages), and (iii) *Shared K-V Store* (agents coordinate by reading/writing shared state).

We evaluate three state-of-the-art LLMs representing diverse architectural paradigms: DeepSeek-V3.1(DeepSeek-AI, 2024), Qwen3-Next-80B-A3B(Team, 2025), and GPT-OSS-120B(OpenAI, 2025). These models span different training recipes and scaling regimes, and exhibit distinct coordination behaviors under identical interfaces, making them suitable stress-tests for multi-agent sorting. To mitigate run-to-run stochasticity from nondeterministic serving and decoding, we repeat each experimental setting multiple times and report results aggregated over the runs. Full system configuration, as well as code and data availability, are reported in the appendix (§B).

5.2 Overview

We structure our analysis around the following five research questions (RQs).

- **RQ1:** How do state-of-the-art LLM agents perform on MAS-BENCH under identical multi-agent constraints?
- **RQ2:** How does the choice of communication backend affect collaboration outcomes?
- **RQ3:** How do collaboration performance and efficiency scale with the number of agents N and local segment length K?
- **RQ4:** Does CAMOC improve collaboration performance across models and communication backends?
- **RQ5:** Which components of CAMOC contribute most to its performance gains, as revealed by ablation studies?

We first summarize per-model performance (Table 1), then isolate scaling trends (Figure 2 and Table 2), and finally attribute gains through ablations (Table 1).

N	Broadcasting			P2P Messaging			Shared K-V Store		
	K=1	5	10	K=1	5	10	K=1	5	10
1	100.0	100.0	100.0	100.0	100.0	100.0	80.0	80.0	80.0
3	82.2	73.9	76.1	76.1	61.7	65.6	58.7	59.1	49.3
5	75.7	72.3	70.0	76.7	61.0	60.3	60.0	56.5	49.3
10	74.8	67.2	52.7	63.3	48.0	38.2	52.4	42.1	37.1
20	60.8	53.5	38.6	46.2	31.5	22.8	41.7	32.8	22.6

Table 2: **Scaling effect on SR (%) with increasing N and K.** Numbers are averaged across all models and difficulties, and shown separately for each communication backend. Increasing N and/or K consistently reduces SR, with the steepest degradation appearing for P2P when both N and K are large, reflecting amplified coordination cost under many-to-many messaging.

5.3 Results & Findings

LLM Performance on MAS-BENCH. Table 1 shows that MAS-BENCH remains challenging even for strong models, especially when coordination is mediated by restrictive backends. Under the Base protocol, DeepSeek-V3.1 achieves the strongest SR in Broadcasting (80.8%), while GPT-OSS-120B is substantially lower (55.6%). Notably, the Shared K-V backend is a major failure mode for Base across all models (e.g., 38.2% for DeepSeek-V3.1 and 25.6% for GPT-OSS-120B), indicating that "implicit" coordination via shared state is not automatically resolved by stronger language models. Notably, even at $N = 1$, the Shared K-V backend exhibits a non-trivial failure rate (20%), which should not arise from coordination difficulty. This suggests the issue is a backend-specific artifact (e.g., read/write or formatting inconsistencies) or a prompting/interface mismatch, rather than a fundamental limitation of single-agent capability; Appendix G.2 (Table 8) isolates this as interface friction via token accounting.

> **Finding (RQ1).** Even when local sorting is trivial, global correctness is limited by cross-agent coordination; naive protocols can collapse under shared-state interaction, revealing a gap between single and multi agent reasoning.

Communication Patterns. Comparing columns in Table 1 reveals systematic backend effects. Broadcasting typically yields higher SR with relatively small R (roughly 8–17 rounds in our setting), as global announcements reduce ambiguity about boundaries and duplicates. P2P Messaging often increases R and CR due to many-to-many negotiation (e.g., DeepSeek-V3.1 has $R \approx 38$ and GPT-OSS-120B has $CR \approx 22$–23%), though the magnitude is model-dependent. Shared K-V Store is the most brittle under Base, since agents must infer a consistent global state from potentially stale or conflicting reads/writes; this is exactly where explicit coordination becomes critical.

> **Finding (RQ2).** Communication topology is not a cosmetic choice: it fundamentally changes the failure surface. Broadcasting favors faster convergence, P2P can incur substantial negotiation overhead, and shared-state coordination is error-prone without additional protocol structure.

Failure-mode Taxonomy. To complement the per-backend aggregates with mechanism-level evidence, we categorize the recurring failure modes observed in the interaction logs. The same five patterns recur across models, each preferentially triggered by a specific backend.

- **Boundary mismatch.** Agents agree on a sorting strategy but disagree on partition boundaries; most frequent under Broadcasting.
- **Premature submission.** An agent calls `submit_result` before sufficient information exchange, producing a locally plausible but globally inconsistent segment.
- **Stale K-V reads.** Under Shared K-V, an agent bases a decision on a key whose value has since been overwritten by another agent, which is the dominant failure mode for this backend.
- **Value hallucination.** An agent emits values absent from the union of local segments, or silently drops elements; prevalence grows with N as context length scales.

• **Convention misalignment.** Agents adopt incompatible coordination conventions (e.g., index-based vs. value-range partitioning), yielding deadlock or mutually inconsistent outputs.

These mechanisms map onto the backend-specific failure surface identified in RQ2: Broadcasting-driven boundary mismatch, K-V-driven stale reads, and the model-agnostic *commit-too-early* and *talk-past-each-other* pair that CAMOC's message and submission primitives (§4.1) are designed to suppress.

Scaling Effects. Scaling degrades performance along multiple axes. First, Table 2 shows that SR generally decreases as N grows, and the drop is more severe for larger K. For example, in Broadcasting SR decreases from 100.0% at $N{=}1$ to 38.6% at $N{=}20$ when $K{=}10$; in P2P the corresponding value is 22.8%. (We observe small non-monotonicities in a few cells, e.g., Shared K-V with $K{=}1$, but the overall trend remains negative with increasing N and/or K.) Second, Figure 2 shows that increasing N drives up R while TE drops sharply, indicating that additional agents primarily introduce coordination overhead rather than parallel speedup. Finally, CR increases with N (often peaking at intermediate N in Figure 2), suggesting that agents increasingly spend tokens on meta-coordination as the system scales. This implies that simply adding more agents is not a viable path to better performance without protocol-level coordination improvements.

Finding (RQ3). Agents on MAS-BENCH exhibits a clear *coordination scaling law*: as N increases, correctness (SR) degrades while both interaction length (R) and coordination overhead (lower TE, higher CR) worsen substantially.

Input family (↑ ρ)	SR%↑	R↓	TE↑	CR
`asc`	87.4	26.9	7.47	7.0
`near_asc`	70.5	29.9	7.11	6.9
`random`	54.1	31.6	7.01	6.8
`near_desc`	52.4	30.1	7.20	6.7
`desc`	52.2	30.7	7.09	7.0

Table 3: **Effect of input order structure.** Higher disorder reduces success rate while only modestly increasing coordination cost.

Performance of CAMOC. CAMOC improves collaboration robustness across models, with especially large gains in the Shared K-V backend. From Table 1, CAMOC raises SR in Shared K-V from $38.2\% \rightarrow 84.7\%$ (DeepSeek-V3.1), $33.7\% \rightarrow 67.2\%$ (Qwen3), and $25.6\% \rightarrow 56.5\%$ (GPT-OSS-120B). In Broadcasting and P2P, CAMOC also improves SR for DeepSeek-V3.1 and GPT-OSS-120B, while typically boosting TE (e.g., DeepSeek-V3.1 TE $2.63 \rightarrow 6.43$ in Broadcasting). The round number R can slightly increase in some settings (e.g., Broadcasting for DeepSeek-V3.1), consistent with CAMOC trading small extra coordination steps for a much higher probability of globally correct termination. Because the raw token overhead is near zero (Appendix G.1, Table 6) while SR improves substantially, the amortised cost per correct submission on Shared K-V drops by roughly one half across all three models (Table 7).

Finding (RQ4). CAMOC provides backend-agnostic gains, and is most impactful when the environment provides weaker coordination primitives (Shared K-V), where it converts unreliable shared-state interaction into consistent global correctness.

5.4 Ablation Study

Table 1 also reports two ablations that isolate key CAMOC components. Across models, removing either component reduces performance, but the failure signature differs by backend. For DeepSeek-V3.1, *Abl.* `message` largely collapses TE (e.g., $6.43 \rightarrow 2.69$ in Broadcasting) and reduces SR sharply under Shared K-V ($84.7\% \rightarrow 70.6\%$), suggesting that message-level coordination is a primary driver of efficiency and shared-state robustness. For GPT-OSS-120B, *Abl.* `message` similarly reverts TE to near-Base levels and decreases SR in Shared K-V ($56.5\% \rightarrow 52.1\%$). Meanwhile, *Abl.* `submit` tends to preserve TE but can reduce SR in several cases, indicating that structured submission/termination criteria chiefly prevent premature or inconsistent finalization. Welch's t-tests across all model×backend cells surface a single statistically significant regression on Qwen3 P2P, which is driven specifically by the One-shot Commitment component and which we dissect in Appendix G.4 (Table 11).

Finding (RQ5). Both components matter: (i) message-level coordination primarily improves *efficiency* and stabilizes shared-state interaction, while (ii) structured submission primarily improves *correctness* by preventing inconsistent termination. Their combination yields the strongest and most consistent gains across backends.

5.5 Discussion & Implications

Coordination as generalized context engineering. Multi-agent collaboration generalizes *context engineering*: the system must manage and align multiple evolving contexts with partial, inconsistent views of the same global state. Communication cost is largely determined by *context overlap*: when subproblems are independent, overlap is low and agents can run almost independently, whereas when overlap is high, coordination becomes the bottleneck. MAS-BENCH is deliberately high-overlap via cross-segment order constraints, and we observe steep SR degradation as N grows (Table 2) together with increasing interaction overhead (Figure 2), which exposes a key limitation of current protocols in maintaining a consistent shared context.

Compute-inefficient collaboration. Even on this simple task, collaboration is *compute-inefficient*: TE collapses with scaling (Figure 2) and P2P can incur large R/CR (Table 1). The bottleneck is not local sorting, but reliably reaching a globally consistent termination with minimal redundant exchange. This points to needs beyond prompting, like coordination-oriented post-training and inference-time protocol support with structured messages and commitment or termination control.

Communication design as first-order. Different backends induce different failure modes: Broadcasting often converges with smaller R, P2P may over-negotiate, and Shared K-V is brittle under naive protocols (Table 1). CAMOC is most beneficial precisely where coordination primitives are weakest (Shared K-V), which suggests that current agents are not backend-invariant and that explicit, auditable coordination layers are necessary in practice.

Transferability of Global Metadata. Global Metadata is highly effective in distributed sorting, but its benefits do not directly generalize to all multi-agent tasks. Sorting is tightly coupled and constraint-heavy: agents must agree on a single global ordering, and coarse signals like value ranges or partitions directly reduce ambiguity and guide convergence. In *loosely coupled* settings, like modular or MapReduce-style tasks, subproblems are largely independent and proactive global sharing may introduce unnecessary overhead with limited benefit. In *semantic or open-ended tasks* like planning or negotiation, there is often no single ground-truth global state, and early global signals can be ambiguous or even harmful, leading to premature convergence or propagation of incorrect assumptions. To empirically test this gradient we extend MAS-Bench with Distributed Maximum (loose coupling) and Distributed Prefix Sum (medium coupling); Base success rates drop monotonically with coupling intensity across all models (Appendix G.3, Table 9), and CAMOC's relative benefit grows accordingly (Table 10) without any task-specific tuning. Nevertheless, the principle still applies when tasks exhibit hidden global constraints or strong information asymmetry, where Global Metadata can act as a coordination prior that helps agents align more efficiently. Overall, Global Metadata is most effective in high-overlap, constraint-driven tasks, but requires adaptation in more open-ended settings with signals that are revisable or uncertainty-aware.

6 Conclusion

We introduce MAS-BENCH, a scalable benchmark that isolates coordination difficulty in distributed multi-agent sorting. It provides a unified task setup across communication paradigms, an empirical study of three frontier LLMs that reveals persistent coordination bottlenecks and unfavorable scaling, and CAMOC as a drop-in layer that consistently improves collaboration, especially under shared-state interaction.

More broadly, our results identify *coordination efficiency under scaling* as a concrete target for future progress. Closing this gap will likely require protocol-aware post-training and inference-time coordination mechanisms, beyond stronger single-agent reasoning alone. Advancing multi-agent systems will depend not only on model capacity, but on principled interfaces that support stable shared state and communication at scale. We hope MAS-BENCH serves as a reproducible objective for developing and measuring backend-invariant multi-agent collaboration.

Limitations

MAS-BENCH centers on distributed sorting, which gives deterministic correctness but leaves out weakly coupled workloads like MapReduce-style computation and open-ended generative tasks. We evaluate only three atomic substrates (Broadcasting, P2P, Shared K-V), not hierarchical, adaptive, or gossip topologies. Agents are homogeneous with fixed prompting, so heterogeneous or role-specialized mixtures may interact with coordination protocols differently. We also assume synchronous, reliable delivery; asynchronous or lossy settings can shift coordination dynamics and failure modes.

Large-scale runs incur non-trivial compute, so we report token and cost statistics for transparency. The benchmark uses synthetic integer sequences and no human data. We position MAS-BENCH and CAMOC as evaluation tools, not deployment-ready artifacts for safety-critical or real-world decision-making.

References

Weize Chen, Yusheng Su, Jingwei Zuo, Cheng Yang, Chenfei Yuan, Chi-Min Chan, Heyang Yu, Yaxi Lu, Yi-Hsin Hung, Chen Qian, Yujia Qin, Xin Cong, Ruobing Xie, Zhiyuan Liu, Maosong Sun, and Jie Zhou. 2024. Agentverse: Facilitating multi-agent collaboration and exploring emergent behaviors. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net.

Xuxin Cheng, Ke Zeng, Zhiquan Cao, Linyi Dai, Wenxuan Gao, Fei Han, Ai Jian, Feng Hong, Wenxing Hu, Zihe Huang, Dejian Kong, Jia Leng, Zhuoyuan Liao, Pei Liu, Jiaye Lin, Xing Ma, Jingqing Ruan, Jiaxing Song, Xiaoyu Tan, Ruixuan Xiao, Wenhui Yu, Wenyu Zhan, Haoxing Zhang, Chao Zhou, Hao Zhou, Shaodong Zheng, Ruinian Chen, Siyuan Chen, Ziyang Chen, Yiwen Dong, Yaoyou Fan, Yangyi Fang, Yang Gan, Shiguang Guo, Qi He, Chaowen Hu, Binghui Li, Dailin Li, Xiangyu Li, Yan Li, Chengjian Liu, Xiangfeng Liu, Jiahui Lv, Qiao Ma, Jiang Pan, Cong Qin, Chenxing Sun, Wen Sun, Zhonghui Wang, Abudukelimu Wuerkaixi, Xin Yang, Fangyi Yuan, Yawen Zhu, Tianyi Zhai, Jie Zhang, Runlai Zhang, Yao Xu, Yiran Zhao, Yifan Wang, Xunliang Cai, Yangen Hu, Cao Liu, Lu Pan, Xiaoli Wang, Bo Xiao, Wenyuan Yao, Qianlin Zhou, and Benchang Zhu. 2025. Higher satisfaction, lower cost: A technical report on how llms revolutionize meituan's intelligent interaction systems. *CoRR*, abs/2510.13291.

Jeffrey Dean and Sanjay Ghemawat. 2008. Mapreduce: simplified data processing on large clusters. *Commun. ACM*, 51(1):107–113.

DeepSeek-AI. 2024. Deepseek-v3 technical report. *CoRR*, abs/2412.19437.

Yilun Du, Shuang Li, Antonio Torralba, Joshua B. Tenenbaum, and Igor Mordatch. 2024. Improving factuality and reasoning in language models through multiagent debate. In *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*, Proceedings of Machine Learning Research, pages 11733–11763. PMLR / OpenReview.net.

Jakob N. Foerster, Yannis M. Assael, Nando de Freitas, and Shimon Whiteson. 2016. Learning to communicate with deep multi-agent reinforcement learning. In *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*, pages 2137–2145.

Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, Tom Hennigan, Eric Noland, Katie Millican, George van den Driessche, Bogdan Damoc, Aurelia Guy, Simon Osindero, Karen Simonyan, Erich Elsen, Jack W. Rae, Oriol Vinyals, and Laurent Sifre. 2022. Training compute-optimal large language models. *CoRR*, abs/2203.15556.

Sirui Hong, Mingchen Zhuge, Jonathan Chen, Xiawu Zheng, Yuheng Cheng, Jinlin Wang, Ceyao Zhang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, Liyang Zhou, Chenyu Ran, Lingfeng Xiao, Chenglin Wu, and Jürgen Schmidhuber. 2024. Metagpt: Meta programming for A multi-agent collaborative framework. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net.

Bin Hu, Chenyang Zhao, Pu Zhang, Zihao Zhou, Yuanhang Yang, Zenglin Xu, and Bin Liu. 2024. Enabling intelligent interactions between an agent and an LLM: A reinforcement learning approach. *RLJ*, 3:1289–1305.

Zimo Ji, Daoyuan Wu, Wenyuan Jiang, Pingchuan Ma, Zongjie Li, Yudong Gao, Shuai Wang, and Yingjiu Li. 2026. Taming various privilege escalation in llm-based agent systems: A mandatory access control framework. *CoRR*, abs/2601.11893.

Zimo Ji, Daoyuan Wu, Wenyuan Jiang, Pingchuan Ma, Zongjie Li, and Shuai Wang. 2025. Measuring and augmenting large language models for solving capture-the-flag challenges. In *Proceedings of the 2025 ACM SIGSAC Conference on Computer and Communications Security, CCS 2025, Taipei, Taiwan, October 13-17, 2025*, pages 603–617. ACM.

Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik R. Narasimhan. 2024. Swe-bench: Can language models resolve real-world github issues? In *The Twelfth*

International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024. OpenReview.net.

Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B. Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. 2020. Scaling laws for neural language models. *CoRR*, abs/2001.08361.

Angeliki Lazaridou, Alexander Peysakhovich, and Marco Baroni. 2017. Multi-agent cooperation and the emergence of (natural) language. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net.

Guohao Li, Hasan Hammoud, Hani Itani, Dmitrii Khizbullin, and Bernard Ghanem. 2023. CAMEL: communicative agents for "mind" exploration of large language model society. In *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*.

Junyou Li, Qin Zhang, Yangbin Yu, Qiang Fu, and Deheng Ye. 2024. More agents is all you need. *Trans. Mach. Learn. Res.*, 2024.

Tian Liang, Zhiwei He, Wenxiang Jiao, Xing Wang, Yan Wang, Rui Wang, Yujiu Yang, Shuming Shi, and Zhaopeng Tu. 2024. Encouraging divergent thinking in large language models through multi-agent debate. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing, EMNLP 2024, Miami, FL, USA, November 12-16, 2024*, pages 17889–17904. Association for Computational Linguistics.

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 Dong, and Jie Tang. 2024. Agentbench: Evaluating llms as agents. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. 2023. Self-refine: Iterative refinement with self-feedback. In *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*.

Yohei Nakajima. 2026. Babyagi. GitHub repository. Accessed: 2026-01-04.

Reza Olfati-Saber, J. Alexander Fax, and Richard M. Murray. 2007. Consensus and cooperation in networked multi-agent systems. *Proc. IEEE*, 95(1):215–233.

OpenAI. 2025. gpt-oss-120b & gpt-oss-20b model card. *CoRR*, abs/2508.10925.

Joon Sung Park, Joseph C. O'Brien, Carrie Jun Cai, Meredith Ringel Morris, Percy Liang, and Michael S. Bernstein. 2023. Generative agents: Interactive simulacra of human behavior. In *Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology, UIST 2023, San Francisco, CA, USA, 29 October 2023- 1 November 2023*, pages 2:1–2:22. ACM.

Shishir G. Patil, Tianjun Zhang, Xin Wang, and Joseph E. Gonzalez. 2024. Gorilla: Large language model connected with massive apis. In *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*.

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. 2024. Chatdev: Communicative agents for software development. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2024, Bangkok, Thailand, August 11-16, 2024*, pages 15174–15186. Association for Computational Linguistics.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. 2023. Toolformer: Language models can teach themselves to use tools. In *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*.

Yongliang Shen, Kaitao Song, Xu Tan, Dongsheng Li, Weiming Lu, and Yueting Zhuang. 2023. Hugginggpt: Solving AI tasks with chatgpt and its friends in hugging face. In *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. 2023. Reflexion: language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*.

Yoav Shoham and Kevin Leyton-Brown. 2009. *Multiagent Systems - Algorithmic, Game-Theoretic, and Logical Foundations*. Cambridge University Press.

Significant-Gravitas. 2026. Autogpt. GitHub repository. Accessed: 2026-01-04.

Qwen Team. 2025. Qwen3 technical report. *CoRR*, abs/2505.09388.

Michael J. Wooldridge. 2009. *An Introduction to Multi-Agent Systems, Second Edition*. Wiley.

Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Shaokun Zhang, Erkang Zhu, Beibin Li, Li Jiang, Xiaoyun Zhang, and Chi Wang. 2023. Autogen: Enabling next-gen LLM applications via multi-agent conversation framework. *CoRR*, abs/2308.08155.

Wangyu Wu, Zhenhong Chen, Xiaowen Ma, Wenqiao Zhang, Xianglin Qiu, Siqi Song, Xiaowei Huang, Fei Ma, and Jimin Xiao. 2026. Contrastive prompt clustering for weakly supervised semantic segmentation. *Expert Syst. Appl.*, 316:131880.

Wangyu Wu, Zhenhong Chen, Xianglin Qiu, Siqi Song, Xiaowei Huang, Fei Ma, and Jimin Xiao. 2025. Llm-enhanced multimodal fusion for cross-domain sequential recommendation. *CoRR*, abs/2506.17966.

Xin Yang, Letian Li, Abudukelimu Wuerkaixi, Xuxin Cheng, Cao Liu, Ke Zeng, Xunliang Cai, and Wenyuan Jiang. 2026. Towards self-robust llms: Intrinsic prompt noise resistance via coipo. *CoRR*, abs/2603.03314.

Xin Yang, Bintao Tang, Yuhao Wang, Zimo Ji, and Wenyuan Jiang. 2025. Can llms write fast system-aware numerical computation code? In *IEEE International Conference on Systems, Man, and Cybernetics, SMC 2025, Vienna, Austria, October 5-8, 2025*, pages 672–675. IEEE.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. 2023a. Tree of thoughts: Deliberate problem solving with large language models. In *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R. Narasimhan, and Yuan Cao. 2023b. React: Synergizing reasoning and acting in language models. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net.

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. 2024a. Webarena: A realistic web environment for building autonomous agents. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net.

Xuhui Zhou, Hao Zhu, Leena Mathur, Ruohong Zhang, Haofei Yu, Zhengyang Qi, Louis-Philippe Morency, Yonatan Bisk, Daniel Fried, Graham Neubig, and Maarten Sap. 2024b. SOTOPIA: interactive evaluation for social intelligence in language agents. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net.

A Artifacts & Open Source Licenses

Our benchmark implementation and experimental code are released under a permissive open-source license. We rely on commonly used open-source libraries for model serving, data processing, and evaluation, and use them in accordance with their respective licenses. Pretrained language models are accessed through their official APIs or repositories and are subject to the licenses and usage terms specified by their providers.

Intended Use and Consistency. All artifacts used in this work are employed in a manner consistent with their stated or implied intended use. Specifically, the proposed benchmark is intended solely for research evaluation of multi-agent coordination; pretrained language models are used for inference-only under their respective research licenses; and all data used in the benchmark are synthetically generated and are not intended to represent or be deployed in real-world sorting or decision-making systems.

B Experimental Details

This appendix provides the full system configuration for our multi-node serving setup, complementing §5.1. Source code will be available upon acceptance and data is available at GitHub.

B.1 Hardware and cluster allocation

All experiments were executed on cluster nodes equipped with NVIDIA GH200 GPUs (120 GB HBM per GPU). A typical multi-node run requests 4 nodes with 4 GPUs per node (16 GPUs total), using one task per node and 288 CPU cores per task: `--nodes=4`, `--ntasks-per-node=1`, `--gpus-per-node=4`, `--cpus-per-task=288`. The job is submitted to the `normal` partition with a 12-hour wall-time limit (`--time=12:00:00`).

B.2 Serving stack and distributed launch

Serving framework. We serve models using SGLang (`lmsysorg/sglang:v0.5.4.post1`) in multi-node mode, launched via `python3 -m sglang.launch_server`. Each allocated node runs exactly one server process under `srun`, and the set of processes collectively forms a single distributed service instance.

Tensor parallelism and node roles. Tensor parallelism is set to the total number of GPUs across nodes, which equals 16 in the default 4×4 setup. The first hostname in `$SLURM_NODELIST` is used as the head node, and all processes share a fixed NCCL initialization address on the head node (`HEAD_NODE:8000`). Each process is assigned its `--node-rank` from `$SLURM_PROCID`, enabling multi-node tensor-parallel execution.

B.3 Model configuration and request formatting

Model weights and caching. Models are loaded from local shared filesystem paths (`--model-path`). SGLang's cache directory is set to a dedicated scratch location (`SGLANG_DG_CACHE_DIR`) to amortize compilation and caching overhead across runs.

Chat template and tool parsing. We use an explicit Jinja chat template (`--chat-template`) matching the model family and enable a model-specific tool-call parser (e.g., `--tool-call-parser deepseekv3`) to robustly interpret command blocks produced by agents. This is required because the benchmark harness relies on text-encoded commands for environment interaction.

B.4 Decoding parameters

Decoding parameters (e.g., temperature/top-p, maximum generation length, and stop sequences) follow the *default* configuration of the serving stack and model backend. Concretely, we do not tune decoding hyperparameters per model or per environment; instead, we use SGLang defaults (as shipped in v0.5.4.post1) together with the model's default generation settings specified by its backend/runtime. Exact request payloads and any framework-level defaults are available in the released configuration files and scripts referenced in §5.1 and §B.

C Token Consumption

This section reports the aggregated token usage of our benchmark runs and provides a cost estimate based on OpenRouter pricing. For each model, we aggregate over all available logs. `total_tokens` counts the sum of all input and output tokens consumed by the harness across all agents and rounds, while `communication_tokens` counts the subset of tokens attributable to communication-related traces (i.e., tool commands and environment observations induced by message/file operations).

Pricing and estimation method. OpenRouter lists separate prices for input and output tokens. We use the listed per-1M token prices for DeepSeek-V3.1 ($0.15/M input, $0.75/M output), Qwen3-Next-80B-A3B-Instruct ($0.06/M input, $0.60/M output), and gpt-oss-120b ($0.02/M input, $0.10/M

Model	#Logs	Total (MTok)	Comm (MTok)	Comm %	Tok/log	Cost ($)
DeepSeek-V3.1	1050	104.373	4.486	4.30	99,403	46.97 [15.66, 78.28]
Qwen3-Next-80B-A3B-Instruct	1050	63.738	2.625	4.12	60,703	21.03 [3.82, 38.24]
gpt-oss-120b	1050	43.660	4.098	9.39	41,580	2.62 [0.87, 4.37]
Total	3150	211.771	11.209	5.29	–	70.62 [20.35, 120.89]

Table 4: Token usage and estimated cost on OpenRouter. "Total (M)" and "Comm (M)" are in millions of tokens. "Cost" reports the midpoint estimate (50/50 input-output) with a conservative range [all-input, all-output].

output). Because our aggregate token counters do not separate input vs. output, we report (i) a midpoint estimate assuming a 50/50 split, and (ii) a conservative range where all tokens are billed as input (lower bound) or as output (upper bound).

Observations. Communication traces account for a modest fraction of overall usage (4–5% for DeepSeek-V3.1 and Qwen3-Next, but 9.39% for gpt-oss-120b), suggesting that differences in coordination behavior can measurably affect token overhead even when the task and harness are fixed. Under the midpoint assumption, communication tokens correspond to roughly $3.13 of the total $70.62 estimated spend across these runs.

D Prompt Design

This appendix summarizes the *shared* prompt structure used across all communication environments in MAS-BENCH. Environment-specific details (e.g., the exact tool list and examples tailored to P2A/P2P/SFS) are defined in the open-source implementation and are not duplicated here.

D.1 Overall structure

Single system prompt as the task contract. Each agent is initialized with a single `system` message that fully specifies (i) the agent identity (`Agent-`i), (ii) the distributed sorting objective, and (iii) the agent's local input segment `xs[i]` along with the global parameters (N, K). The prompt embeds an explicit `check_results` reference function that defines correctness: every agent must submit exactly K integers, and the concatenation of all submissions must equal the globally sorted concatenation of all input segments. This makes the success condition unambiguous and executable.

Tool-mediated interaction via command blocks. Agents are instructed *not* to use the model's native tool-call interface. Instead, they interact with the environment by emitting plain-text commands in fenced code blocks. The prompt enforces a strict grammar: each code block contains *exactly one* command (optionally preceded/followed by natural-language reasoning outside the block). This design cleanly separates "thinking" from "acting" and makes command extraction robust.

Closed-loop observations as textual feedback. After the agent outputs commands, the harness executes them and returns the results back to the agent as `user` messages (e.g., acknowledgements, retrieved messages, read file contents, or error strings). These environment responses become the agent's observations for the next LLM call, enabling iterative coordination without exposing implementation details.

D.2 Common instruction components

Explicit formatting and parsing rules. Across environments, the prompt repeats a small set of hard constraints: (1) one command per fenced code block, (2) no extra commentary inside the code block, (3) natural language is allowed outside code blocks. These constraints ensure deterministic parsing and prevent accidental tool invocation failures.

Coordination guidance (environment-agnostic). The prompt encourages agents to proactively coordinate before submitting, emphasizing that local correctness is insufficient without global consistency. Concretely, it recommends sharing structured information such as: the agent ID, local sublist, intended ownership interval/partition in the global order, and explicit requests/expectations for other agents. The precise mechanism for sharing (messages vs. shared files) is environment-specific, but the content and intent are consistent.

Conservative submission policy. A shared behavioral rule is to avoid early submission: agents should first exchange enough information to agree on a globally consistent ordering/partition. Only when confident that their local output fits the agreed global plan should they call `submit_result`.

Termination and immutability after submission. The prompt treats submission as final. Once an agent issues `submit_result`, it is instructed to stop issuing further commands and not attempt

to modify its answer. This mirrors the benchmark evaluation protocol and prevents "post-hoc" corrections.

Error surfaces and recovery. The harness returns explicit textual error messages when it cannot parse commands, detects invalid arguments, or rejects a command due to state constraints. The prompt implicitly guides recovery by requiring the agent to adapt in the next round (e.g., re-issuing a corrected command), keeping the interaction fully text-based and reproducible.

D.3 Rerun variant for self-correction

Second-run context injection. When the benchmark is configured to rerun a failed case, the system prompt adds a short prefix indicating that this is a second attempt and includes the agent's previously submitted result. The shared instruction is: if the prior submission was correct, resubmit it unchanged; otherwise, use additional coordination to identify the inconsistency and submit a corrected output. This rerun wrapper enables controlled self-correction while preserving the same action interface and evaluation contract.

D.4 Environment-specific prompt additions

Tool documentation blocks. The only major prompt differences across environments are the tool documentation sections (names, argument formats, and examples) and small coordination examples aligned with each environment's abstraction (broadcasting, directed P2P, or shared K-V/files). These differences are intentionally localized so that the task contract, command grammar, and behavioral constraints remain comparable across environments.

E Communication Environments

This appendix summarizes the three communication backends used in MAS-BENCH and how they are exposed to LLM agents. All three environments are implemented as lightweight, in-memory state machines backed by SQLite, wrapped as thread-safe singletons to support multi-agent execution within a single benchmark run.

Agents do *not* call structured function tools; instead, each agent emits plain-text commands in fenced code blocks, which the harness parses and executes.

E.1 Common execution and synchronization model

Round-based stepping. The benchmark runs in synchronous rounds. In each round, the harness first queries the LLM for *all* still-active agents (concurrently), then executes the returned commands (sequentially) to update the environment state. As a result, any message/file update created in round r can only influence an agent's next model call (round $r+1$), even if it is written early during command execution in round r.

Reliability and visibility. To isolate coordination from network artifacts, all environment actions are reliable: writes are committed immediately to the SQLite backend under a global lock, and reads observe committed state. There is no probabilistic loss, delay, or reordering beyond the round boundary described above.

Agent identity and ordering. Agents are referred to by 0-based integer IDs (Agent-i). Internally, SQLite uses 1-based autoincrement row IDs; each environment applies a fixed offset to expose stable 0-based public IDs. When ordering matters, environments return records in ascending insertion order (increasing row ID), which provides a deterministic within-backend ordering of messages/files.

E.2 Broadcasting environment

Implementation overview. Broadcasting is implemented as an in-memory SQLite "chat" database with two tables: `users` and `messages`. Each broadcast inserts one message row per receiver (all users except the sender). All database operations are protected by a single mutex, and the environment is accessed via a process-wide singleton that is reset between benchmark runs.

Message delivery semantics. A broadcast from Agent-i to $N-1$ recipients is materialized as $N-1$ rows, each with `sender_id`, `receiver_id`, `content`, and `is_read`. Receiving uses a pull-based API (`recv`) that returns unread messages for the calling agent ordered by insertion ID; by default, the returned messages are marked read (setting `is_read=1`), so future receives do not return duplicates.

Synchronization with the harness. Agents observe new broadcasts only after they explicitly execute `receive_messages`. Because LLM generation happens before command execution in a round, broadcasts sent in round r become observable to all agents starting in round $r+1$. This yields a

clean, synchronous "one-round latency" pub-sub abstraction.

Tools exposed to LLM agents. Under broadcasting, the harness exposes the following text commands: (i) `receive_messages` (pull unread broadcasts), (ii) `broadcast_message <payload>` (one-to-all message), (iii) `list_agents` (enumerate registered IDs), (iv) `wait` (no-op synchronization marker), (v) `submit_result <list>`. Additionally, when an agent submits, the harness automatically broadcasts a submission announcement as a convenience signal to others (still subject to the round boundary).

E.3 P2P messaging environment

Implementation overview. P2P messaging uses the same SQLite-backed structure as broadcasting (users + messages), but with directed point-to-point sends. A `send` inserts a single message row addressed to a specific receiver. The environment is also a thread-safe singleton reset per run.

Message delivery semantics. Directed messages are addressed by (sender, receiver) IDs; `recv` returns messages for the receiver in insertion order with the same unread/mark-as-read behavior as broadcasting. The environment enforces existence checks for both endpoints (sender and receiver must be registered).

Synchronization and "offline" behavior. As in broadcasting, messages become visible across rounds due to the harness's "generate-then-execute" schedule. In addition, the harness treats agents as offline after `submit_result`: subsequent attempts to message an already-submitted agent are rejected at the harness layer (without inserting into the environment), preventing late coordination with finalized agents.

Tools exposed to LLM agents. Under P2P, the harness exposes: (i) `receive_messages`, (ii) `send_message <to_id> <payload>`, (iii) `wait`, (iv) `submit_result <list>`. Compared to broadcasting, there is no global mailbox listing command and no one-to-all primitive; all global coordination must be realized through many-to-many directed sends.

E.4 Shared K-V store environment

Implementation overview. The shared-state backend is implemented as a minimal shared file system on top of an in-memory SQLite table `files(key, content, created_at, updated_at)`. Each path acts as a key and the stored text content acts as its value; writes overwrite existing values by default, making the abstraction equivalent to a last-writer-wins K-V store. As with the messaging backends, all operations are serialized by a global lock and accessed via a singleton reset between runs.

Read/write and visibility semantics. `write(k, v)` either creates a new record or overwrites the existing record at `k`, updating `updated_at`. `read(k)` returns the full content (erroring if missing), and `list(prefix)` provides discovery by listing all files whose paths share a given prefix, ordered by insertion ID. `delete(k)` removes a key entirely, which is used to mitigate stale state when agents choose to clean up intermediate coordination artifacts.

Synchronization with the harness. Because writes are immediate and reads are consistent, agents can reliably implement shared ledgers, partition plans, and checkpoints. However, as with messaging, any shared-state update written in round r only affects other agents' next-round model calls (round $r{+}1$). This yields a synchronous shared-memory coordination substrate with deterministic overwrite semantics.

Tools exposed to LLM agents. Under Shared K-V store, the harness exposes: (i) `list [prefix]`, (ii) `read <k>`, (iii) `write <k>` followed by a free-form multi-line payload (stored verbatim as content), (iv) `delete <k>`, (v) `wait`, (vi) `submit_result <list>`. Upon submission, the harness additionally writes a small marker file recording that the agent has gone offline and what it submitted; this provides a persistent trace for other agents that rely on discovery via `list`.

F Algorithmic Specification

This appendix gives the formal counterpart to the narrative descriptions in §D and §E: the task generator, the synchronous round, the per-modality command-dispatch contract, and the CAMOC orchestrator together with the exact prompt fragments it injects. The goal is reconstructibility of the method without access to the source.

F.1 Notation

Let $A = \{0, \ldots, N{-}1\}$ index agents. Fix K elements per agent, $\text{mode} \in \{\texttt{random}, \texttt{asc}, \texttt{desc}, \texttt{near\_asc}, \texttt{near\_desc}\}$, and a round budget $R_{\max}{=}100$ per phase. Each agent i holds a chat history $M[i]$, seeded with one system message built by BUILDPROMPT (§F.6),

Algorithm 1 INITSORTTASK

Require: $N, K \in \mathbb{Z}_{>0}$; mode $\in$ {`random`, `asc`, `desc`, `near_asc`, `near_desc`}.
Ensure: (xs, expected): per-agent inputs and ground-truth sorted slices; $|\text{xs}[i]|=|\text{expected}[i]|=K$.

$L \leftarrow \text{sample}(\text{range}(10NK),\ NK)$ ▷ distinct integers
ans $\leftarrow \text{sorted}(L)$ ▷ ground-truth ascending order
if mode $\in$ {`desc`, `near_desc`} **then**
  $B \leftarrow \text{sorted}(L, \text{rev=True})$
else
  $B \leftarrow \text{sorted}(L)$
end if
$\rho \leftarrow 1$ if mode=`random`; 0.2 if "near" $\in$ mode; 0 otherwise
$B' \leftarrow$ PARTIALSHUFFLE(B, ρ) ▷ permute $\lfloor \rho NK \rfloor$ random positions
xs $\leftarrow \left(B'[iK{:}(i{+}1)K]\right)_{i=0}^{N-1}$
expected $\leftarrow \left(\text{ans}[iK{:}(i{+}1)K]\right)_{i=0}^{N-1}$
return (xs, expected)

Algorithm 2 STEPBENCH

Require: chat histories M, submission slots S, environment singleton $\mathcal{E}$.
Ensure: one synchronous round applied in place: M extended with the new assistant/user turns and $S/\mathcal{E}$ updated via the command dispatcher of Table 5.

for all $i \in A$ with $S[i] = \perp$ **in parallel do**
  $(t_i,\ e_i) \leftarrow \text{LLM}(M[i])$ ▷ thread pool, one call per active agent
end for
for all $i \in A$ with $S[i] = \perp$ **sequentially do**
  append (assistant, t_i) to $M[i]$
  if $e_i \neq \perp$ **then**
    append (user, μ_{err}) to $M[i]$; **continue**
  end if
  cmds $\leftarrow$ PARSEFENCED(t_i)
  if cmds $= \varnothing$ **then**
    append (user, μ_{noCmd}) to $M[i]$; **continue**
  end if
  for all $b \in$ cmds **in textual order do**
    $r \leftarrow$ DISPATCH$(i,\ b,\ \mathcal{E},\ S)$ ▷ Table 5
    append (user, r) to $M[i]$
  end for
end for

and a submission slot $S[i] \in \mathbb{Z}^K \cup \{\perp\}$. All agents share an environment singleton $\mathcal{E}$ (one of Broadcasting, P2P, Shared K-V; §E) whose state is reset at the start of every phase.

F.2 Task generator

The success predicate CHECK(xs, ys) requires $|\text{ys}[i]|=K$ for all i and $\text{sorted}(\bigcup_i \text{xs}[i]) = \text{concat}(\text{ys})$. The predicate is permutation-equivalent within the multiset: any concatenation matching the sorted union passes. Per-agent strict correctness (ys$[i]$=expected$[i]$) is recorded separately (metric SR, §3.4).

F.3 Synchronous round

Every round executes *all* active agents in one parallel LLM batch, then drains each reply's fenced command blocks sequentially against $\mathcal{E}$. No command written in round r is observable before round $r{+}1$.

Above, μ_{err} and μ_{noCmd} denote the literal harness-injected reply strings "Environment could not process that step" and "No commands detected in last reply." PARSEFENCED returns every match of the non-greedy DOTALL regex `` ```(?:[^\n`]*\n)?(.*?)``` ``, discarding any language tag on the opening line and stripping trailing newlines. All blocks in one reply are executed.

F.4 Command-dispatch contract

Table 5 summarises the LLM-visible command surface of each modality: the environment effect and the reply string the harness injects back into $M[i]$ on the next round. Invalid arguments leave $S[i] = \perp$ and return a typed error reply; the agent may retry on its next turn. Any uncaught exception becomes `<cmd> -> error: <exc>`; an unknown verb becomes `Unknown command: <cmd>`. In the Shared K-V row, "path" denotes the key and "content" the value; the verbs retain file-system names purely for historical reasons.

F.5 RUNBENCH and the CAMOC orchestrator

A run is parameterised by two orthogonal knobs: the behavior-list variant beh $\in$ {short, CAMOC} injected into BUILDPROMPT, and a boolean twoPhase that enables a single-commit verification cycle. CAMOC is (beh=CAMOC, twoPhase=True); the remaining three settings yield Base and the two one-factor ablations (Table 1). For brevity let $\mathcal{P}_r(i, \pi) :=$ BUILDPROMPT$(i, \text{xs}[i];$ beh, rerun$=r$, prev$=\pi)$; phase 1 uses $\mathcal{P}_{\text{F}}(i, \perp)$ and phase 2 uses $\mathcal{P}_{\text{T}}(i, \text{subs}[i])$.

The four conditions used in Table 1 correspond to

Condition	beh	twoPhase
Base	short	False
Abl. `message`	short	True
Abl. `submit`	CAMOC	False
CAMOC	CAMOC	True

F.6 Prompt fragments injected by BUILDPROMPT

BUILDPROMPT concatenates three fragments into the initial system message: (i) a common header (agent identity, N, K, the verbatim `xs[i]` list, and

Modality	Command	Environment effect	Reply to agent i
Broadcasting	`receive_messages`	fetch unread rows to i; mark read	`broadcast from Agent-`j`: . . .` joined by `||`
	`broadcast_message` *text*	insert $(i, j,$ *text*$)$ for all $j \neq i$	`delivered (Agent-`j`,. . .)`
	`list_agents`	enumerate `users`	`Agent-0:. . . , Agent-1:. . .`
	`wait`	no-op	`acknowledged`
	`submit_result` *list*	$S[i] \leftarrow$ *list*; auto-broadcast "Agent-i submitted result *list*"	`recorded . . . | broadcasted to . . .`
P2P	`receive_messages`	fetch unread rows to i; mark read	`from Agent-`j`: . . .` joined by `|`
	`send_message` *to body*	if $S[to] \neq \perp$ reject; else insert $(i, to, body)$	`delivered to Agent-to` / offline-refusal
	`wait`	no-op	`acknowledged`
	`submit_result` *list*	$S[i] \leftarrow$ *list*; no auto side effect	`recorded list`
Shared K-V	`list_files` [*prefix*]	rows with `path` $\sim$ *prefix*%	`id=. . . path=. . . len=. . . updated=. . .`
	`read_file` *key*	return full value at *key*, or error	`content=<verbatim>`
	`write_file` *key*\n*value*$_{\textit{multi-line}}$	upsert $(key, value)$; overwrite	`wrote . . . preview=<120 chars>`
	`delete_file` *key*	remove row at *key*	`removed . . .`
	`wait`	no-op	`acknowledged`
	`submit_result` *list*	$S[i] \leftarrow$ *list*; auto-write `Agent-`i`_submission.txt`	`recorded . . . | wrote_file=. . .`

Table 5: Command-dispatch contract per modality. The three auto side effects on submission (broadcast announcement for Broadcasting, offline rejection for P2P, marker file for Shared K-V) are the sole asymmetries between modalities at the commitment step.

Algorithm 3 RUNBENCH

Require: $N, K,$ mode; behavior variant beh $\in \{\text{short}, \text{CAMOC}\}$; flag twoPhase $\in \{\text{False}, \text{True}\}$; round budget $R_{\max}$.
Ensure: (`global_check`, $(\text{correct}[i])_{i \in A}$): global multiset checker outcome (§F.5) and per-agent strict equality to $\text{expected}[i]$, taken on the final submission S (of phase 2 when twoPhase, of phase 1 otherwise).

$(\text{xs}, \text{expected}) \leftarrow$ INITSORTTASK(N, K, mode)
reset $\mathcal{E}$; register $\{\text{Agent-}i\}_{i=0}^{N-1}$
for $i \in A$ **do**
  $M[i] \leftarrow [\mathcal{P}_{\text{F}}(i, \perp)]$; $S[i] \leftarrow \perp$
end for
$r \leftarrow 0$
while $r < R_{\max} \ \wedge \ \exists\, i\colon S[i] = \perp$ **do**
  STEPBENCH$(M, S, \mathcal{E})$; $r \leftarrow r + 1$
end while
$\text{subs} \leftarrow (S[0], \ldots, S[N-1])$
if twoPhase **then** ▷ CAMOC and Abl. `message`
  reset $\mathcal{E}$; register $\{\text{Agent-}i\}_{i=0}^{N-1}$
  for $i \in A$ **do**
    $M[i] \leftarrow [\mathcal{P}_{\text{T}}(i, \text{subs}[i])]$; $S[i] \leftarrow \perp$
  end for
  $r \leftarrow 0$
  while $r < R_{\max} \ \wedge \ \exists\, i\colon S[i] = \perp$ **do**
    STEPBENCH$(M, S, \mathcal{E})$; $r \leftarrow r + 1$
  end while
end if
return (CHECK(xs, S), $(S[i]{=}\text{expected}[i])_{i \in A}$)

the literal `check_results` reference function); (ii) the modality-specific command reference matching Table 5; and (iii) the behavior list. The fragments in (i) and (ii) are identical across beh $\in \{\text{short}, \text{CAMOC}\}$. The sole behavioral difference is the three additional clauses inserted into the behavior list when beh$=$CAMOC, which we reproduce verbatim below.

Agent Role and Focus: *You must concentrate solely on the actions and data required for your specific role. While you should proactively share your own outputs when relevant and clearly request necessary information from other agents, you are not to speculate about or manage the needs of others. Your focus remains exclusively on determining what you need to obtain from others to fulfill your function and what to submit, not on what they should submit.*

Communication Priority and Efficiency: *You must optimize communication by strictly limiting the number of messages you send in each round. It is acceptable to gather required data incrementally across multiple rounds if needed. However, the highest priority is always given to responding to requests from other agents. In every round, you must first address and fully resolve any incoming requests before you initiate any new requests or communication of your own.*

Deliberate Action Cycle: *Before taking any action in a new round, you must begin by thoroughly reviewing all received messages and available information. Carefully interpret the current state, including any shared data or pending requests. Only after this consideration should you plan and execute your response.*

These three clauses are inserted between the "avoid submitting too early" rule and the submit-lock rule of the short prompt, and the bullet "What you expect other agents to do" is removed from the structured-message guidance. They instantiate, respectively, the role-scope discipline, the receive-first bounded communication, and the per-round read-then-act cycle of §4.2–4.4.

Phase-2 rerun addendum. When rerun$=$True, BUILDPROMPT prefixes two lines to the header and appends one line after the behavior list. No

phase-1 conversation, environment state, or peer submission is carried over; the only link between the two phases is each agent's own previous output, injected by value as `<prev>`.

Rerun addendum (verbatim prompt text)

(prepended after "You are Agent-i in a multi-agent system solving a distributed sorting task.")

This is the second run of the task, you should consider the previous results and check if your submission is correct. If not, you should submit the correct result.

The previous results you submitted are: `<prev>`

(appended after the behavior list)

You should consider the previous results you submitted: `<prev>` and check if your submission is correct by using tools list above. If not, you should communicate with other agents by tools and submit the correct result. If you think your previous submission is correct, you should submit the same result again. You must communicate with other agents to get enough information to check if you are correct before you submit!

Because $M[i]$ is reset at the start of phase 2, this addendum and the injected `<prev>` value are the *only* signals the model receives about phase 1. This realises the One-shot Commitment principle (§4.4) without accumulating phase-1 context in the LLM window.

G Additional Experiments

This appendix reports additional evidence collected during the response period: CAMOC's token-cost profile (§G.1), an isolation of the $N{=}1$ Shared K-V anomaly (§G.2), generalisation beyond sorting to loose- and medium-coupling tasks (§G.3), and a per-component regression diagnostic (§G.4). All numbers are aggregated over the same $\sim$2,200-run grid used in the main body unless stated otherwise.

G.1 CAMOC Token Cost

Per-backend overhead. Table 6 reports total tokens per backend for Base/ablation runners versus full CAMOC. Differences are within $\pm 6\%$ across all backends: CAMOC adds 1–2% on the noisier Shared K-V backend and is slightly *cheaper* on P2P and Broadcasting, reflecting fewer clarification loops. Topology dominates token consumption by 2–5$\times$, while variant choice within a single backend accounts for $<10\%$.

Backend	Base/Abl.	CAMOC	Δ
Shared K-V	$\sim$620K	$\sim$628K	$+1-2\%$
P2P	$\sim$258K	$\sim$254K	$-1-2\%$
Broadcasting	$\sim$102K	$\sim$ 97K	$-4-6\%$

Table 6: Total tokens per backend ($\sim$2,200 runs in aggregate). CAMOC's redundancy cost is negligible; backend topology is the dominant driver of token consumption.

Tokens per correct instance on Shared K-V. Because CAMOC's overhead is flat while its success rate is much higher (Table 1), the amortised cost *per correct submission* drops by roughly one half across all three models (Table 7).

Model	Base	CAMOC	Reduction
DeepSeek-V3.1	$\sim$2.8M	$\sim$1.3M	-54%
Qwen3-Next-80B-A3B	$\sim$1.6M	$\sim$0.82M	-49%
GPT-OSS-120B	$\sim$0.75M	$\sim$0.35M	-53%

Table 7: Tokens per correct instance on Shared K-V. CAMOC roughly halves the effective cost of a successful run despite adding only 1–2% raw tokens.

G.2 Investigating the $N{=}1$ Shared K-V Artifact

The 20% failure rate at $N{=}1$ under Shared K-V (Table 1) cannot be attributed to coordination, since a lone agent has no one to coordinate with. Token accounting at $N{=}1$ (Table 8) shows Shared K-V consumes $\sim$28% more tokens than Broadcasting/P2P for the same workload (two-sided Welch's t-test, $p{<}0.001$). The overhead is spent parsing list/read/write/delete semantics. We therefore interpret the $N{=}1$ gap as *interface friction* rather than coordination breakdown; part of CAMOC's gain under Shared K-V stems from channelling interactions into a narrower, more regular command sequence and thereby reducing this friction.

G.3 Generalisation Beyond Sorting

To test whether the coordination scaling law and CAMOC's principles are sorting-specific, we extend MAS-Bench with two tasks covering different coupling regimes: *Distributed Maximum* (loose coupling, where agents share only their local maxima) and *Distributed Prefix Sum* (medium coupling,

Backend at $N{=}1$	Tokens / run
Shared K-V	$\sim$1,360
P2P	$\sim$1,070
Broadcasting (P2A)	$\sim$1,060

Table 8: Per-run token consumption at $N{=}1$ by backend. The $\sim$28% Shared K-V overhead is significant ($p{<}0.001$) and isolates interface friction from coordination difficulty.

where agents share cumulative sums). Base success rate decreases monotonically with coupling intensity across all three models (Table 9), confirming that the benchmark's difficulty axis is *coordination demand*, not task realism.

Task (coupling)	DS-V3.1	GPT-OSS	Qwen3
Aggregation (loose)	$\sim$61%	$\sim$28%	$\sim$21%
Prefix Sum (medium)	$\sim$34%	$\sim$15%	$\sim$ 3%
Sorting (tight)	$\sim$12%	$\sim$ 9%	$\sim$ 1%

Table 9: Base success rate (averaged over (N, K, mode) settings) for three tasks spanning loose, medium, and tight coordination coupling. SR drops with coupling intensity across all models.

Applying CAMOC without any task-specific tuning yields consistent gains; the relative benefit grows with coupling intensity (Table 10). Ablation structure also transfers: structured messaging (*Abl.* `submit`) captures most of the gain, matching the sorting ablation in Table 1.

Condition	Dist. Max	Prefix Sum
Base	$\sim$58%	$\sim$32%
Abl. `message`	$\sim$63%	$\sim$35%
Abl. `submit`	$\sim$66%	$\sim$38%
CAMOC	$\sim$**71**%	$\sim$**43**%

Table 10: CAMOC and ablation SR on Distributed Maximum (loose) and Distributed Prefix Sum (medium) with DeepSeek-V3.1, averaged over (N, K, mode) settings. Gains scale with coupling intensity; ablation structure mirrors sorting (Table 1).

G.4 Per-component Regression Diagnostic on Qwen3 P2P

A Welch's t-test sweep across all model$\times$backend cells identifies a single statistically significant CAMOC *regression*: Qwen3 on P2P drops from Base 56.9% to full CAMOC 51.2% ($p{<}0.05$). The ablation isolates the cause (Table 11): removing One-shot Commitment (*Abl.* `submit`) recovers Base-level performance, whereas removing message structuring leaves the regression intact. Qwen3 on P2P thus appears to benefit from the iterative refinement loops that One-shot Commitment intentionally suppresses; the same model still gains $+31$ to $+34$pp on Shared K-V. Pairwise differences between backends remain highly significant ($p{<}0.001$); within a single backend, CAMOC's *token*-level differences relative to Base are small and typically non-significant, matching Table 6.

Qwen3 P2P condition	SR
Base	56.9%
Abl. `message`	55.1%
Abl. `submit`	49.3%
CAMOC	51.2%

Table 11: Per-component SR for Qwen3 on P2P. Welch's t-test Base vs. CAMOC: $p{<}0.05$. Removing *Abl.* `submit` shows the regression is driven by the One-shot Commitment component, which suppresses iterative refinement that Qwen3 relies on in this backend.