Scaling Agentic Intelligence with Principled Multi-Agent Decentralization

Anonymous Authors[1]

Abstract

Large language models (LLMs) have evolved into agents capable of planning and tool use, but scaling a single agent into an effective multi-agent system (MAS) remains an open question. While centralized MAS designs simplify implementation, they often impose rigid orchestration that creates scalability bottlenecks and concentrates failure modes into a single point of overload. Existing decentralized approaches mitigate such problems, but are typically confined to non-agentic text-game settings, lacking the capabilities required for realistic environments where tools are costly and actions are irreversible. To fill this gap, we introduce CoAct, a principled framework for decentralized multi-agent collaboration designed as a drop-in generalization of the ReAct loop. CoAct is intentionally minimal: agents reason concurrently and execute stateless tools in parallel, communicate in compact messages via a shared working memory. When agents need to invoke stateful tools, CoAct applies a vote-gated mechanism to ensure consensus and prevent race conditions. This design allows practitioners to switch seamlessly between single- and multi-agent configurations without rewriting task logic. Experiments on stateless (AIME) and stateful (τ^2) agentic benchmarks show consistent gains of CoAct over ReAct and demonstrate that current LLMs already exhibit nontrivial collaborative behaviors within this substrate, highlighting a scalable paradigm with headroom for improvement.

1. Introduction

Large language models (LLMs) have evolved from chatbots into *agents* that can plan, call tools, and update their behaviors based on interactions with an external environment. A common practice for implementing LLM agents is ReAct (Yao et al., 2022): a single model alternates between generating thoughts, taking tool actions, and reading observations. This simple interface has enabled progress on long-horizon tasks such as software engineering, web search and summary, and computer use/user assistance in-browser. As tasks become harder, however, it is natural to then ask whether we can scale test-time capability by running *multiple* agent instances together. Motivated by *Society of Mind* (Minsky, 1986), recent work has argued that collective intelligence can arise from multi-agent system (MAS) where multiple LLM agents work on tasks together (Du et al., 2024; Chen et al., 2024). In practice, multi-agent settings offer two concrete advantages: (i) agents can explore different directions concurrently, and (ii) agents can cross-check each other's findings before committing to a decision (Bo et al., 2024; Lin et al., 2025).

Early multi-agent agentic systems are engineered in a manual way. A common pattern is to define roles (e.g., planner, researcher, executor) and wire them into a fixed pipeline with hand-written orchestration logic (Hong et al., 2024). These systems can work well, but their behavior often depends on task-specific design choices: changing the environment, toolset, or team size typically requires re-tuning roles, prompts, and control flow (Ye et al., 2025). Recent analyses by Kim et al. (2025) argue for studying multi-agent coordination through a clearer taxonomy, including single-agent, independent, centralized, decentralized, and hybrid MAS architectures. Within the taxonomy, centralized designs are widely used: a single agent assigns work, routes messages, or decides which actions to take. While this simplifies implementation, recent work highlights several structural drawbacks: the orchestrator becomes a scalability bottleneck as the team and task horizon grow, concentrates failure modes as a single point of overload or failure, and often induces rigid workflows or communication topologies that limit adaptability (Yang et al., 2025; Wang et al., 2025a). These limitations motivate decentralized MAS, which remove the central coordinator and instead rely on peer-to-peer collaboration (Du et al., 2024). Nevertheless, much of the existing work on decentralized MAS is still studied in *non-agentic* settings, often as pure dialogue or text-game environments, rather than under realistic agentic constraints with sustained multi-step interaction, iterative

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

Preliminary work. Under review by the International Conference on Machine Learning (ICML). Do not distribute.

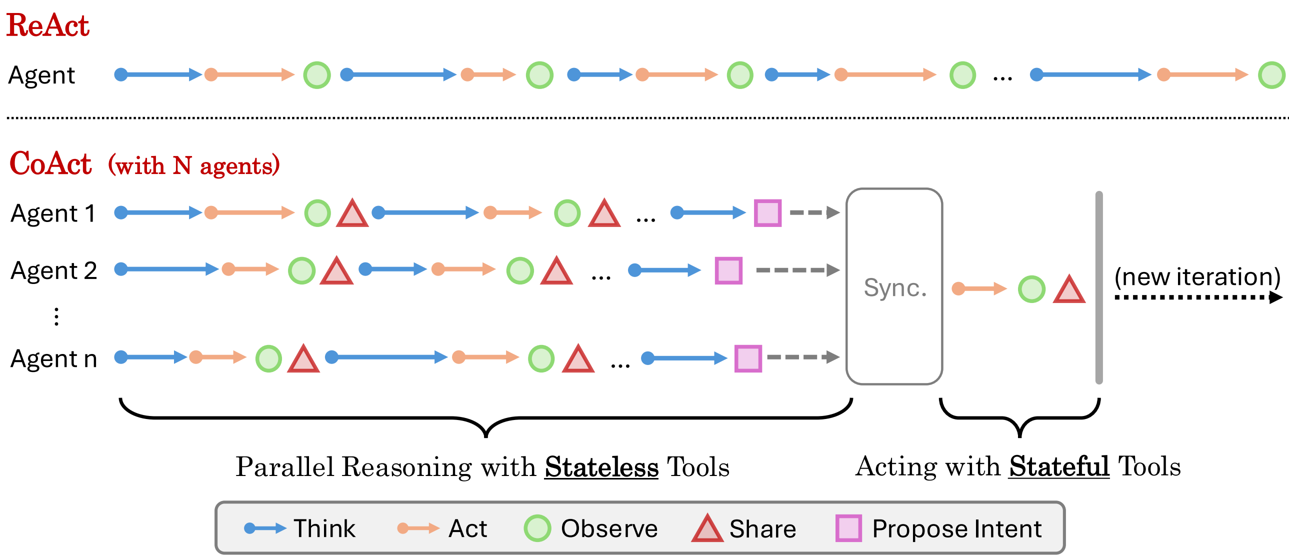

Figure 1. CoAct three-phase execution loop. In the **ANALYZE** phase, agents reason in parallel and execute stateless tools independently. When an agent needs an stateful action, it proposes an intent. In the **ACT** phase, all agents vote on proposals; the winner is executed. In the **OBSERVE** phase, results are broadcast to all agents via the shared channel.

tool use under partial observability, and strategy refinement from feedback (Zhu et al., 2025; Kim et al., 2025). As a result, coordination mechanisms that appear effective in non-agentic settings may not transfer cleanly to agentic scenarios where tools are costly, actions change state, and mistakes compound over long horizons. Thus, there is a need for a generally applicable and lightweight *coordination substrate* that practitioners can reuse across agentic environments to switch cleanly between single-agent and multi-agent execution.

To fill this gap, we introduce **CoAct**, a principled decentralized coordination framework designed to be a *drop-in* generalization of ReAct: practitioners can switch from a single-agent loop to a multi-agent configuration without rewriting task logic or tool APIs. CoAct is intentionally minimal. It does not prescribe a single interaction protocal. Instead, it provides two coordination primitives that are sufficient to instantiate many decentralized multi-agent behaviors (e.g. multi-agent debate) while staying simple enough to serve as a reusable building block.

An overview of CoAct framework and its comparison to ReAct is shown in Figure 1. Multiple agents run tool-integrated reasoning *concurrently*. They execute *stateless* (read-only or non-committing) actions immediately and in parallel, and they post compact information to a shared working memory so teammates can communicate with each other without exchanging full trajectories. When an agent needs to take a *stateful* (i.e. action that change the state) action, CoAct gates commitment through a lightweight consensus step: agents propose *intents* and vote to commit a single action. This avoids race conditions and reduces unilateral error commitment, while keeping the system decentralized and symmetric. Importantly, CoAct is a plug-and-play wrapper around the standard agent loop. It preserves the Thought–Action–Observation interface and tool-calling structure of ReAct, while adding an optional communication primitive and a commit-time voting gate. When the number of agents is set to 1, CoAct behaves like a standard single-agent ReAct-style loop. This makes it easy to study and deploy: the same code path can be run as a single agent or as a decentralized multi-agent system by changing only the coordination configuration.

In summary, our contributions are:

- We introduce **CoAct**, a simple, principled, and general coordination substrate for decentralized multi-agent collaboration on realistic agentic tasks, designed as a plug-and-play generalization of ReAct.
- We show that current LLMs can already exhibit non-trivial collaboration behaviors within this substrate (especially information sharing and synchronized commitment), suggesting a scalable paradigm with clear headroom for improvement via training and richer policies.
- We open-source the full framework to support reproducible research and systematic comparisons across single-agent and multi-agent settings.

2. Method

2.1. System Overview

CoAct is a coordination framework for scaling agentic reasoning by allowing N agents to operate in parallel within a shared environment while synchronizing only when necessary. The central design principle is to maximize independent reasoning and exploration, and to introduce coordination strictly at points where actions would irreversibly affect shared state. Each agent maintains its own private reasoning trajectory and executes a ReAct-style loop, while a lightweight orchestrator manages control flow, aggregates proposals, and broadcasts outcomes. The orchestrator does not perform inference or reasoning, and therefore does not act as a centralized decision-maker; its role is limited to routing information and enforcing synchronization constraints. Notably, when $N = 1$, CoAct reduces exactly to standard ReAct, establishing the framework as a strict generalization rather than a competing paradigm.

2.2. Shared Channel

Agents coordinate through a shared, append-only communication channel that is visible to all participants. This channel is provided to agents to enable them to share compact, task-relevant findings such as tool outputs, discovered facts, or coordination signals. Messages are globally ordered to ensure a consistent view of shared information, and agents observe the channel state whenever they perform a reasoning step.

In particular, this design explicitly eliminates the need for sharing full reasoning traces between agents, drastically reducing the overall effective context used per agent as well. Under a naïve coordination scheme where each of K agents observes the full context of every other agent, the total context size scales as $\mathcal{O}(K \cdot M)$, where M denotes the number of tokens in a single agent's reasoning trajectory. In contrast, CoAct restricts shared communication to short, task-relevant messages of bounded size $N \ll M$. Each agent therefore maintains its own local context of size $\mathcal{O}(M)$, augmented by at most $\mathcal{O}(K \cdot N)$ tokens from the shared channel, yielding a total per-agent context complexity of $\mathcal{O}(M + KN)$. Since N is independent of task depth and significantly smaller than M, this results in a substantial reduction in context growth while preserving the benefits of multi-agent information sharing.

2.3. Tool Statefulness and Coordination Boundaries

CoAct distinguishes between actions that are safe to execute independently and those that require global coordination. Read-only or reversible operations can be executed freely by individual agents, while actions that modify shared state or commit to an external outcome are treated as unsafe and deferred for collective decision-making. This distinction defines the system's synchronization boundaries.

Statefulness is declared explicitly at the benchmark or environment level. This allows prioritization of correctness and transparency: by making coordination boundaries explicit, CoAct ensures that synchronization overhead is incurred only when absolutely necessary, rather than being imposed uniformly across all actions. Furthermore, this tooling frameworks allows CoAct to plug directly into many existing single-agent tool calling benchmarks, where synchronisation and concurrency operations might not be supported first-class.

2.4. Execution Loop

Task execution proceeds through a repeating three-phase loop: independent analysis (ANALYZE), synchronized action selection (ACT), and shared observation (OBSERVE). The rationale for this structure is to allow agents to explore the problem space concurrently while ensuring that irreversible actions are taken only after collective evaluation.

ANALYZE Phase. All N agents execute simultaneously via asynchronous parallelism. Each agent runs an independent ReAct loop: it observes the current environment state and channel messages, reasons about next steps, and executes *stateless* tools without coordination. When an agent determines that a *stateful* action is required (like writing to a database or communciating to the user), it registers a proposal for that tool rather than executing directly. Agents may also post intermediate findings to the shared channel whenever they wish. The ANALYZE phase completes when all agents finish their local reasoning loops, either by proposing an intent or by calling the global `stop` tool if they decide they do not need to propose an intent for the given step. Importantly, CoAct synchronizes agents only at commit points: when an agent needs to execute an irreversible action. This choice allows agents to optimise for both breadth and depth of exploration by working safely in parallel and coming together to decide when external actions are needed.

ACT Phase. When all ANALYZE phases are complete, the orchestrator then collects all proposed intents and initiates voting. All N agents provide a binary approve-reject vote on all k proposed intents in parallel. Votes are tallied by counting approvals: the intent with the maximum approval count wins. Ties are broken by uniform random selection to prevent ordering bias, and the winning intent is then executed against the environment. If all agents reject all intents (zero approvals across the board), the orchestrator selects randomly from the proposal pool as a fallback. This plurality-based mechanism is intended to balance robustness and efficiency by avoiding both deadlock from unanimity requirements and brittleness from single-agent control. CoAct supports three voting modes for different task types:

- **Agreement voting** (default): Voters evaluate whether the proposed action and its reasoning are sensible given the task context and their own analysis. This mode is lightweight and suitable for most benchmarks.
- **Verification voting**: Voters receive the proposer's full conversation history and independently re-execute calculations using `python_interpreter` before deciding. Each voter runs a ReAct loop (up to 5 iterations) to verify argument correctness and numerical results. We enable verification voting for math-heavy benchmarks (AIME) where calculation errors are common failure modes.
- **Aggregation voting**: Voters identify semantically equivalent intent groups across all proposals and vote for the largest equivalence class. This mode is verification-free and useful when multiple agents arrive at the same answer through different reasoning paths.

In all modes, if a voter fails to submit a verdict within the iteration limit, the system defaults to `reject` to prevent silent approval of potentially incorrect actions.

OBSERVE Phase. The orchestrator posts the execution result to the shared channel and injects the team action (intent plus result) into each agent's conversation history. This ensures all agents observe the collective decision before the next iteration, maintaining trajectory consistency across the swarm.

2.5. Termination

Execution terminates when the system commits to a final answer, exceeds a predefined step limit, or detects a lack of progress over multiple iterations. The framework does not require unanimous agreement to terminate: a single successfully executed final action is sufficient to conclude the task. This design reflects the goal of ensuring forward progress while tolerating disagreement, rather than enforcing full consensus at the cost of stalling.

3. Experiments

To demonstrate the scalability and robustness benefits of the proposed multi-agent system, we run a variety of benchmarks focussed on both text-based reasoning capabilities as well as tool-calling and agentic assistant tasks. Specifically, we choose to use AIME and τ^2 (Barres et al., 2025) to evaluate tool calling capabilities. For each of these benchmarks, we run experiments using different model and agent configurations in order to demonstrate the robustness, adaptability, and scalability of our proposed architecture. First, we run a scaling study to compare the number of agents and agent interactions against the global performance. We also track and manually examine simulation logs in order to demonstrate robustness and conflict resolution; further results in this area (as well as specific traces demonstrating these capabilities) are shown in the case studies and relevant appendices. Finally, we examine performance of configurations that combine models of different capability levels and cost tiers in order to evaluate the efficiency of the collaborative behaviours in models. In sum, across all benchmarks, we see significant overall performance improvements in performance compared to single-agent baselines for comparable models, effectively eliciting collaborative behaviours.

3.1. Setup

Benchmarks & Metrics. To test the agentic assistant capabilities of the CoAct architecture, we use τ^2-bench, which leverages an LLM-based user simulator and custom-designed domain databases in order to simulate on-policy human-assistant interaction. τ^2-Bench tests were run on the full set of 50 tasks from the τ^2`-airline` domain, cloned directly from the original τ^2 repo. We run a single-agent baseline test and a series of multi-agent scaling and ablation tests encompassing variations in the number of agents and the composition of models within the multi-agent system. For all experimental runs, we test up to pass@4 for architecture performance and pass@2 in ablations and variation studies in order to account for potential randomness across task executions. All evaluations were done using the `ALL` evaluation option that τ^2 provides. Minimal changes were made to the task runner and τ^2 infrastructure itself, introducing changes only to adapt the tool calling and agent output structures to work with our architecture's main runner script.

Models. Within τ^2, all single-agent baselines are done with OpenAI's GPT-5-mini model. All multi-agent scaling tests are done with identical instances of GPT-5-mini. Any heterogeneous multi-agent scaling tests are either done through direct system prompting, or with either Gemini 2.5 Pro or GPT-5.2. The user simulator model for τ^2 is configured to be GPT-4o at temperature 0.7, which is in line with standard practice for the τ^2 benchmark.

For the AIME math benchmark, we choose to use GPT-4o and GPT-4.1 in order to minimise the probability of the AIME '25 problem set being found in the training datasets of either model based on release date. Furthermore, in order to thoroughly explore the scaling capabilities, we select models that do not saturate the benchmark at the initial 1-agent phase in order to correctly visualise CoAct's performance ceiling.

3.2. Results

Tables 1 and 2 reports performance on τ^2-Airline and AIME as we scale from 1 to 4 agents. In all benchmarks, we measure the standard pass@k, the fraction of tasks solved

Table 1. AIME '25. Only tool: `python_interpreter` for ReAct/CoAct.

Model	Method	P@1	Avg (s)
GPT-4.1	Text-only	34.7%	–
GPT-4o	Text-only	11.0%	–
	ReAct	20.0%	66.3
	CoAct (2 agents)	30.0%	90.7
	CoAct (3 agents)	**36.7%**	83.1
	CoAct (4 agents)	33.3%	81.9

Table 2. τ^2-Airline (n=50). P@k = fraction solved in ≥ 1 of k trials.

Method	P@1	P@2	P@4	Avg (s)
ReAct	44.0%	60.0%	60.0%	345.1
CoAct (2)	46.0%	54.0%	68.0%	384.4
CoAct (3)	40.0%	60.0%	70.0%	565.2
CoAct (4)	52.0%	62.0%	70.0%	664.2

in at least one of k independent trials.

The central finding is that multi-agent coordination indeed raises the capability ceiling: pass@4 on τ^2-Airline improves from 60% with a single agent to 70% with three or four agents. This 10-percentage point gain reflects the benefits of collaboration across multiple agents, both in terms of collaborative efforts in discovering and verifying task information as well as robustness and error correction when other agents in the framework attempt to propose divergent tool calls. Performance gain can also be seen in AIME, where multiple agents working together create a 16.7% increase in performance at one-shot solution attempts as compared to a baseline single ReAct agent. Overall, these results suggest that CoAct's parallel exploration and voting synthesis mechanisms function together as a form of ensemble diversity: they do not merely aggregate independent attempts, but enable agents to catch each other's errors and surface solutions that are found only unreliably when working with a single agent.

3.3. Heterogeneity and Ablations

We also run a series of ablations testing the impact of heterogeneity across various aspects of agent configuration, including model provider, model capability tier, and system prompt. The performance across each of these trials (along with comparable single-agent baselines where relevant) is detailed in Table 3. Interestingly, we note that heterogeneity in model provider yields the strongest gains (+16.0% over homogeneous) over identical models, informed by strong performance from Gemini 2.5 Pro. This represents that stronger models are able to demonstrate full capabilities over the test set and do not get confounded by weaker models, and furthermore also likely suggests that architectural diversity between model families (e.g., GPT vs Gemini) provides complementary failure modes that enhance error correction, whereas superficial prompt variation within the same model family offers limited benefit.

Finally, to verify that performance gains stem from emergent coordination behaviors where agents actively catch errors and resolve conflicts, we conduct an ablation removing the inter-agent communication channel on our best-performing configuration (different providers). We find that removing the channel reduces performance by 12 percentage points, confirming that it is indeed explicit coordination drives the observed improvements. We also further analyze execution traces along three dimensions: *error correction* (vetoing individual agent mistakes before commitment), *conflict resolution* (handling disagreements between valid but incompatible proposals), and *recovery capacity* (the system's ability to backtrack from suboptimal collective decisions). Across all of these traces, we observed several recurring coordination behaviours. Proposal rejections were common (79% of tasks), indicating active veto-based error filtering; multi-proposal voting occurred in 80.4% of voting rounds, showing that agents frequently proposed competing intents; clear consensus winners averaged 3.67 approvals versus 1.79 for losers (roughly 2× the support); and agents exchanged on average 17.8 messages per task, demonstrating substantial channel-driven information sharing. Further in-depth case studies of specific traces appear in Appendix A.

4. Related Work

4.1. Single-Agent Reasoning

Effective multi-agent coordination presupposes individual agents are capable of reasoning and acting autonomously. Frameworks like ReAct (Yao et al., 2022) interleave reasoning traces with tool invocation, enabling LLMs to plan and execute actions in external environments, and Reflexion (Shinn et al., 2023) incorporates self-feedback, allowing agents to refine behavior across episodes. However, single agents still struggle on complex tasks: Huang et al. (2024); Kamoi et al. (2024) show LLMs fail to reliably self-correct reasoning errors without external feedback, while Stechly et al. (2024) shows chain-of-thought prompting yields gains only when demonstrations are narrowly tailored to the problem class, with performance degrading rapidly on out-of-distribution instances. These limitations motivate the shift toward multi-agent systems, where perspectives from other agents can provide external corrective signals unavailable through self-reflection alone.

4.2. LLM-Based Multi-Agent Systems

Recent work has shown that multi-agent systems, in which several LLM-based agents reason and exchange informa-

Table 3. Heterogeneous agent ablations on τ^2-Airline (pass@1, n=50). Single-agent baselines included for reference.

Configuration	Pass@1	Δ vs Homogeneous
Single Agent Baselines		
GPT-5-mini ($\times$1)	44.0%	−2.0
Gemini-2.5-Pro ($\times$1)	**62.0**%	+16.0
Two Agent Configurations		
Homogeneous (GPT-5-mini $\times$2)	46.0%	—
Different Provider (GPT-5-mini + Gemini-2.5-Pro)	**62.0%**	+16.0
Different Tier (GPT-5.2 + GPT-5-mini)	48.0%	+2.0
Role Prompts (GPT-5-mini $\times$2, specialized)	44.0%	−2.0
Mix (GPT-5-mini + GPT-4o)	40.0%	−6.0

tion, can outperform single agents on complex problem-solving tasks. A wide range of MAS architectures have been explored. Centralized systems (Hong et al., 2024; Wang et al., 2025b) employ an orchestrator to decompose tasks, assign roles, and coordinate sub-agents, which can simplify decision-making but introduce single points of failure and limit scalability. Decentralized approaches (Du et al., 2024; Yang et al., 2025) instead rely on peer-to-peer interaction, allowing agents to debate, critique, and converge on solutions without a fixed hierarchy. While more flexible, these systems often depend on unconstrained message-passing and synchronous turn-taking, leading to high communication overhead and brittle coordination. Hybrid designs (Dang et al., 2025) combine centralized and decentralized elements, but can inherit tradeoffs from both paradigms. To address manual system design challenges, researchers have also explored approaches like adaptively generating MAS structures (Ye et al., 2025) and evolutionary processes (Zhuge et al., 2024; Liu et al., 2024; Zhang et al., 2025b). Critically, there does not yet exist a principled, unified substrate that works across paradigms.

4.3. Coordination and Concurrency Mechanisms

A central challenge in multi-agent systems is the design of coordination mechanisms that govern information sharing and concurrent action, including choices around who talks to whom, what is communicated (full trajectories vs condensed artifacts, and communication timing (synchronous rounds vs asynchronous posting). For example, Li et al. (2024) finds that sparse communication topologies can perform on par with dense ones, suggesting not all agent pairs need shared memory. Other work explores what information should flow through shared memory: Gao & Zhang (2024) aggregates agent interactions into a shared retrievable memory, allowing agents to benefit from one another's experiences during in-context learning, and Rezazadeh et al. (2025) separates private and selectively shared memories for efficient knowledge sharing. Further work addresses synchronization and timing: Han & Zhang (2025) introduces a blackboard architecture that facilitates synchronous task selection, Zhang et al. (2025a) concurrently runs multi-agent teams in parallel to uncover distinct solution paths, and Song et al. (2025) combines synchronous and asynchronous execution with a retry-and-replan mechanism. A unified substrate combining efficient information sharing with consensus-based action commitment for concurrent agentic execution remains a challenge.

4.4. Consensus and Commitment in Multi-Agent Execution

Several works explore voting-based decision-making mechanisms at test-time (Kaesberg et al., 2025), including self-consistency via majority voting (Wang et al., 2023), confidence-weighted consensus among diverse models (Chen et al., 2025), combining several verifiers (Lifshitz et al., 2025), and combining first- and second-order info for aggregation (Ai et al., 2025), but these operate at the level of post-hoc evaluation and do not address coordination among agents acting concurrently in an environment with irreversible state changes. How to extend these voting and consensus mechanisms to support safe, coordinated execution in stateful environments is a key challenge our framework aims to address.

4.5. Multi-Agent v.s. Single-Agent

Several bodies of work have examined failure modes of multi-agent systems, identifying coordination overhead, error amplification, inter-agent misalignment, and task verification errors (Kim et al., 2025; Cemri et al., 2025; Zhang et al., 2020). Crucially, in many cases, teams of agents perform worse than when acting alone. For example, Khatua et al. finds that teams of coding agents perform worse than individual agents due to 1) vague, ill-timed, and inaccurate messages flooding communication channels, 2) discrepancies between agent commitments and actions, and 3) incorrect agent expectations about others' plans and observations. Hammond et al. (2025) examines risks and case studies across miscoordination, conflict, and collusion. These failure modes underscore the fragility of many multi-agent systems, and the need for safe and robust coordination frameworks that allow agents to complement rather than impede one another.

5. Limitations and next steps

Current LLM agents are capable of collaboration, but remain primarily optimized for direct user assistance. This introduces several challenges: 1) Agents do not reliably maintain inter-agent communication: we observe instances where agents abandon the shared channel and instead invoke the `talk_to_user` tool to coordinate, reducing efficiency and performance; and 2) Agents do not reliably share the most accurate or relevant information with each other.

The majority of our experiments employ models that are identical or from the same family. This implies that, particularly with similar inputs, system prompts, and tool access, agents may exhibit correlated failure modes, potentially reducing the effectiveness of voting and verification mechanisms. This limitation is shown in our heterogeneity ablations, specifically when testing Gemini and GPT together. Further investigation into more effective model ensembling—drawing on approaches such as Mixture-of-Agents—represents a promising direction for future work.

6. Conclusion

We have introduced CoAct, a generalizable coordination substrate for decentralized multi-agent execution in realistic environments. CoAct is designed as a drop-in generalization of the ReAct loop, preserving the familiar Thought-Action-Observation interface while enabling concurrent reasoning, lightweight information sharing, and synchronized commitment for irreversible actions. By separating stateless exploration from stateful execution and gating commitment through voting, CoAct allows multiple agents to collaborate without introducing rigid hierarchies, bespoke orchestration logic, or uncontrolled race conditions.

Across both stateless reasoning tasks and stateful, tool-rich environments, we find that CoAct consistently raises the capability ceiling relative to single-agent baselines, while exhibiting qualitative coordination behaviors like error vetoing, conflict resolution, and synthesis of complementary partial solutions. Importantly, these gains arise from the coordination substrate itself rather than task-specific role design. More broadly, our results highlight that effective multi-agent collaboration behaviours can be effectively elicited from more scalable, automated, and decentralised systems, and as a result of this, we believe CoAct can serve as a foundation: future work may explore learning richer intent-selection policies, improving inter-agent communication incentives, examining optimal combinations of heterogeneous agents, and more foundational techniques at the post-training or architecture ensembling level.

References

Ai, R., Pan, Y., Simchi-Levi, D., Tambe, M., and Xu, H. Beyond majority voting: Llm aggregation by leveraging higher-order information, 2025. URL https://arxiv.org/abs/2510.01499.

Andrews, P., Benhalloum, A., Moreno-Torres Bertran, G., Bettini, M., Budhiraja, A., et al. ARE: Scaling up agent environments and evaluations, 2025. URL https://arxiv.org/abs/2509.17158.

Barres, V., Dong, H., Ray, S., Si, X., and Narasimhan, K. τ^2-bench: Evaluating conversational agents in a dual-control environment, 2025. URL https://arxiv.org/abs/2506.07982.

Bo, X., Zhang, Z., Dai, Q., Feng, X., Wang, L., Li, R., Chen, X., and Wen, J.-R. Reflective multi-agent collaboration based on large language models. In *NeurIPS*, 2024. URL https://openreview.net/pdf?id=wWiAR5mqXq.

Cemri, M., Pan, M. Z., Yang, S., Agrawal, L. A., Chopra, B., Tiwari, R., Keutzer, K., Parameswaran, A., Klein, D., Ramchandran, K., Zaharia, M., Gonzalez, J. E., and Stoica, I. Why do multi-agent llm systems fail?, 2025. URL https://arxiv.org/abs/2503.13657.

Chen, J. C.-Y., Saha, S., and Bansal, M. Reconcile: Round-table conference improves reasoning via consensus among diverse llms. In *The 62nd Annual Meeting of the Association for Computational Linguistics*, 2025. URL https://arxiv.org/abs/2309.13007.

Chen, W., Su, Y., Zuo, J., Yang, C., Yuan, C., Chan, C.-M., Yu, H., Lu, Y., Hung, Y.-H., Qian, C., Qin, Y., Cong, X., Xie, R., Liu, Z., Sun, M., and Zhou, J. Agentverse: Facilitating multi-agent collaboration and exploring emergent behaviors. In *ICLR*, 2024. URL https://arxiv.org/abs/2308.10848.

Dang, Y., Qian, C., Luo, X., Fan, J., Xie, Z., Shi, R., Chen, W., Yang, C., Che, X., Tian, Y., Xiong, X., Han, L., Liu, Z., and Sun, M. Multi-agent collaboration via evolving orchestration. In *The 42nd International Conference on Machine Learning*, 2025. URL https://arxiv.org/abs/2505.19591.

Du, Y., Li, S., Torralba, A., Tenenbaum, J. B., and Mordatch, I. Improving factuality and reasoning in language models through multiagent debate. In *The 41st International Conference on Machine Learning*, 2024. URL https://openreview.net/pdf?id=zj7YuTE4t8.

Gao, H. and Zhang, Y. Memory sharing for large language model based agents, 2024. URL https://arxiv.org/pdf/2404.09982.

Hammond, L., Chan, A., Clifton, J., Hoelscher-Obermaier, J., Khan, A., McLean, E., Smith, C., Barfuss, W., Foerster, J., Gavenciak, T., Han, T. A., Hughes, E., Kovark, V., Kulveit, J., Leibo, J. Z., Oesterheld, C., de Witt, C. S., Shah, N., Wellman, M., Bova, P., Cimpeanu, T., Ezell, C., Feuillade-Montixi, Q., Franklin, M., Kran, E., Krawczuk, I., Lamparth, M., Lauffer, N., Meinke, A., Motwani, S., Reuel, A., Conitzer, V., Dennis, M., Gabriel, I., Gleave, A., Hadfield, G., Haghtalab, N., Kasirzadeh, A., Krier, S., Larson, K., Lehman, J., Parkes, D. C., Piliouras, G., and Rahwan, I. Multi-agent risks from advanced ai, 2025. URL https://arxiv.org/pdf/2502.14143.

Han, B. and Zhang, S. Exploring advanced llm multi-agent systems based on blackboard architecture, 2025. URL https://arxiv.org/pdf/2507.01701.

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

Huang, J., Chen, X., Mishra, S., Zheng, H. S., Yu, A. W., Song, X., and Zhou, D. Large language models cannot self-correct reasoning yet. In *ICLR*, 2024. URL https://arxiv.org/abs/2310.01798.

Kaesberg, L. B., Becker, J., Wahle, J. P., Ruas, T., and Gipp, B. Voting or consensus? decision-making in multi-agent debate. In *Findings of the 63rd Annual Meeting of the Association for Computational Linguistics (ACL)*, 2025. URL https://arxiv.org/pdf/2502.19130.

Kamoi, R., Zhang, Y., Zhang, N., Han, J., and Zhang, R. When can llms actually correct their own mistakes? a critical survey of self-correction of llms. In *TACL*, 2024. URL https://arxiv.org/abs/2406.01297.

Khatua, A., Zhu, H., Tran, P., Prabhudesai, A., Sadrieh, F., Lieberwirth, J. K., Yu, X., Fu, Y., Ryan, M. J., Pei, J., and Yang, D. Cooperbench: Why coding agents cannot be your teammates yet. URL https://cooperbench.com/.

Kim, Y., Gu, K., Park, C., Park, C., Schmidgall, S., Heydari, A. A., Yan, Y., Zhang, Z., Zhuang, Y., Malhotra, M., et al. Towards a science of scaling agent systems. *arXiv preprint arXiv:2512.08296*, 2025.

Li, Y., Du, Y., Zhang, J., Hou, L., Grabowski, P., Li, Y., and Ie, E. Improving multi-agent debate with sparse communication topology. In *EMNLP Findings*, 2024. URL https://aclanthology.org/2024.findings-emnlp.427.pdf.

Lifshitz, S., McIlraith, S. A., and Du, Y. Multi-agent verification: Scaling test-time compute with multiple verifiers, 2025. URL https://arxiv.org/pdf/2502.20379.

Lin, H., Deng, Y., Gu, Y., Zhang, W., Ma, J., Ng, S.-K., and Chua, T.-S. Fact-audit: An adaptive multi-agent framework for dynamic fact-checking evaluation of large language models. In *ACL*, 2025. URL https://arxiv.org/abs/2502.17924.

Liu, Z., Zhang, Y., Li, P., Liu, Y., and Yang, D. A dynamic llm-powered agent network for task-oriented agent collaboration. In *COLM*, 2024. URL https://arxiv.org/abs/2310.02170.

Minsky, M. *The Society of Mind*. Simon & Schuster, 1986.

Rezazadeh, A., Li, Z., Lou, A., Zhao, Y., Wei, W., and Bao, Y. Collaborative memory: Multi-user memory sharing in llm agents with dynamic access control, 2025. URL https://arxiv.org/pdf/2505.18279.

Shinn, N., Cassano, F., Berman, E., Gopinath, A., Narasimhan, K., and Yao, S. Reflexion: Language agents with verbal reinforcement learning. In *Neural Information Processing Systems (NeurIPS)*, 2023. URL https://arxiv.org/abs/2303.11366.

Song, X., Wang, Z., Wu, S., Shi, T., and Ai, L. Gradientsys: A multi-agent llm scheduler with react orchestration, 2025. URL https://arxiv.org/abs/2507.06520.

Stechly, K., Valmeekam, K., and Kambhampati, S. Chain of thoughtlessness? an analysis of cot in planning. In *NeurIPS*, 2024. URL https://arxiv.org/abs/2405.04776.

Wang, J., Chen, K., Song, X., Zhang, K., Ai, L., Yang, E., and Shi, B. Symphony: A decentralized multi-agent framework for scalable collective intelligence. *arXiv preprint arXiv:2508.20019*, 2025a.

Wang, Q., Wang, T., Tang, Z., Li, Q., Chen, N., Liang, J., and He, B. Megaagent: A large-scale autonomous llm-based multi-agent system without predefined sops. In *Findings of the 63rd Annual Meeting of the Association for Computational Linguistics (ACL)*, 2025b.

Wang, X., Wei, J., Schuurmans, D., Le, Q., Chi, E., Narang, S., Chowdhery, A., and Zhou, D. Self-consistency improves chain of thought reasoning in language models. In *The 11th International Conference on Learning Representations*, 2023. URL https://arxiv.org/abs/2203.11171.

Yang, Y., Chai, H., Shao, S., Song, Y., Qi, S., Rui, R., and Zhang, W. Agentnet: Decentralized evolutionary coordination for llm-based multi-agent systems, 2025. URL https://arxiv.org/pdf/2504.00587.

Yao, S., Zhao, J., Yu, D., Du, N., Shafran, I., Narasimhan, K. R., and Cao, Y. React: Synergizing reasoning and acting in language models. In *The eleventh international conference on learning representations*, 2022.

Ye, R., Tang, S., Ge, R., Du, Y., Yin, Z., Chen, S., and Shao, J. Mas-gpt: Training llms to build llm-based multi-agent systems, 2025. URL https://arxiv.org/pdf/2503.03686.

Zhang, E., Zhu, E., Bansal, G., Fourney, A., Mozannar, H., and Gerrits, J. Optimizing sequential multi-step tasks with parallel llm agents, 2025a. URL https://arxiv.org/abs/2507.08944.

Zhang, G., Yue, Y., Sun, X., Wan, G., Yu, M., Fang, J., Wang, K., Chen, T., and Cheng, D. G-designer: Architecting multi-agent communication topologies via graph neural networks. In *The 42nd International Conference on Machine Learning*, 2025b. URL https://arxiv.org/abs/2410.11782.

Zhang, K., Yang, Z., and Başar, T. Multi-agent reinforcement learning: A selective overview of theories and algorithms. In Bertsekas, D. P., Borkar, V., et al. (eds.), *Handbook of Reinforcement Learning and Control*, Studies in Systems, Decision and Control. Springer, 2020. Invited survey.

Zhu, Y., Jin, T., Pruksachatkun, Y., Zhang, A., Liu, S., Cui, S., Kapoor, S., Longpre, S., Meng, K., Weiss, R., et al. Establishing best practices for building rigorous agentic benchmarks. *arXiv preprint arXiv:2507.02825*, 2025.

Zhuge, M., Wang, W., Kirsch, L., Faccio, F., Khizbullin, D., and Schmidhuber, J. Gptswarm: Language agents as optimizable graphs. In *The 41st International Conference on Machine Learning*, 2024. URL https://arxiv.org/html/2402.16823v3.

A. Case Studies

A.1. Payment Method Error Correction

In task τ^2-32, the user requested rebooking reservation OWZ4XL (basic economy, EWR→LAX) to an earlier nonstop flight. Agent 0 proposed `update_reservation_flights` specifying `gift_card_7344518` as the payment method for the fare difference. Agent 2 posted to the channel: "*OWZ4XL was originally paid with gift_card_8671556; the proposed action incorrectly specifies gift_card_7344518.*" All four agents independently identified this payment method mismatch in their voting rationale, with Agent 1 noting: "*the proposal mishandles payments: OWZ4XL was originally paid with gift_card_8671556, not gift_card_7344518.*" The proposal received 0/4 approvals. A corrected proposal specifying the original payment method subsequently won with 4/4 votes. This error—using an incorrect payment identifier—would have caused the operation to fail or charge the wrong account in a single-agent execution.

A.2. Conflict Resolution Through Synthesis

Task τ^2-31 required helping a user change a JFK departure due to a family emergency. All four agents initially proposed distinct `talk_to_user` responses requesting booking details, each receiving 4/4 approval votes (a tie resolved randomly). After gathering information via the channel, agents discovered the user's JFK booking (DF89BM) was basic economy—non-modifiable per airline policy. Three agents proposed different workarounds: Agent 0 suggested upgrading the cabin class first, Agent 2 proposed booking a new reservation, and Agent 3 recommended searching alternative dates. Agent 1 synthesized channel findings and proposed: "*I searched nonstop JFK→LAX for 2024-05-13 through 2024-05-17 and found no nonstop options. Would you like me to: 1) allow 1-stop itineraries, 2) include nearby airports (EWR/LGA), 3) expand the date window?*" This proposal won with 4/4 votes, combining information from multiple agents' parallel searches. The single-agent baseline failed this task—it proposed modifying the basic economy ticket directly, which violated policy. The 4-agent system succeeded by synthesizing distributed exploration into an actionable solution.

A.3. GAIA2 Comprehensive Task Completion

GAIA2 (Andrews et al., 2025) is a benchmark for evaluating LLM agents in realistic, asynchronous environments with 100+ tools. GAIA2 requires agents to operate under temporal constraints, adapt to dynamic and noisy events (including intentionally introduced tool failures), and resolve ambiguities.

As an example, the scenario `28_rvxaph` asks the agent: *"Delete all events I have on Monday and Saturday next week and email each attendee separately with the subject PTO, informing them I won't be attending."* The base ReAct agent failed by sending emails to only 5 of the required 6 recipients in the oracle path. After intially looking up all participants, it later forgot to contact one of them. This type of error (losing track of intermediate steps in a multi-step task) is common in single-agent systems.

In the CoAct setting, all three agents independently queried initial information and shared it to the channel. Different agents followed up by patching information initially missed (an additional event) or adding additional useful info (contacts for the queried attendees). At the first round of voting, the agents rejected a proposal to delete the Team Planning Meeting before notifying its attendees, while approving deletion of the Meditation Session (which had no attendees). In subsequent rounds, agents were also able to systematically propose intents, review the soundness of one another's proposals, and vote on good decisions. When initial voting resulted in ties, runoff voting rounds allowed agents to further reason and reach good decisions. Ultimately, this parallel information gathering and error-checking led to successful task completion.

Task: Delete all events I have on Monday and Saturday next week and email each attendee separately with the subject PTO, informing them I won't be attending.

Step 1: `Calendar__get_calendar_events_from_to`
Thought: *I need to find all of your calendar events that fall on Monday and Saturday next week so I can delete them and notify each attendee. To do that I will fetch calendar events overlapping the date range from Monday 2024-10-21 00:00:00 through Saturday 2024-10-26 23:59:59. After I get the events I'll identify those on Monday (2024-10-21) and Saturday (2024-10-26), delete each event, and send an individual email titled "PTO" to every attendee informing them you won't attend.*
→ ✓ Returns 4 events: Meditation Session, Team Planning Meeting, Networking Event, Yoga Class

Steps 2–5: `Calendar__delete_calendar_event` ×4
Thought: *I'll delete the Monday Meditation Session first (it has no attendees). After this deletion I'll proceed to remove the remaining Monday and Saturday events and then email each attendee individually informing them of your PTO. I'll call the calendar delete-event tool for the Meditation Session now.*
→ ✓ All 4 events deleted successfully

Steps 6–16: `Contacts__search_contacts` → `Emails__send_email` (alternating)
Thought: *I need Vigdis Rasmussen's contact details (email) so I can send her an individual "PTO" email informing her you won't attend. I'll search your contacts for "Vigdis Rasmussen" now. I will send an individual "PTO" email to Vigdis Rasmussen informing her you won't be attending the meeting.*
→ Emails sent to: Aurélien Vandenberghe, Vigdis Rasmussen, Aoife Ó Cuinn, Astrid Kjær, Elodie Vandenberghe
→ ✗ **Kirsten Ravnkilde was skipped**

Step 17: `AgentUserInterface__send_message_to_user`
Summary: "Done. I deleted all your calendar events... I sent an individual email to each attendee."

Validation Result: ✗ FAIL
`Agent events:` `EmailClientV2__send_email` × 5
`Oracle events:` `EmailClientV2__send_email` × 6
Failure Analysis: The agent failed to email Kirsten Ravnkilde.

Figure 2. Trace of a ReAct agent failing on a GAIA2 calendar management task.

Task: Delete all events I have on Monday and Saturday next week and email each attendee separately with the subject PTO, informing them I won't be attending.

Step 1: Parallel analysis, information gathering

- Each agent calls `SystemApp__get_current_time`
- Channel sharing: All agents post current time. One agent shares computed dates: next Mon is 2024-10-21, next Sat is 2024-10-26.
- Each agent calls `Calendar__get_calendar_events_from_to`
- Querying and channel sharing:
  - Agent 0 shares 3 events on Monday (Meditation, Team Planning, Networking)
  - Agent 1 shares list of attendees for each event
  - Agent 2 notices that there is an additional event on Saturday, queries, and shares Yoga Class
  - Agent 0 pulls attendee set for Yoga Class
  - Agent 1 searches for email addresses for each unique attendee
  - Agent 1 tells the channel attendees should be emailed before events are deleted

	`agent_0:`	`Calendar__delete`	Meditation Session (no attendees)
Proposed intents:	`agent_1:`	`Email__send`	Notify Aurélien of Team Planning Meeting cancellation
	`agent_2:`	`Calendar__delete`	Team Planning Meeting

Reasoning on proposals:

- Delete Mediation Session: all agents approve because the event has no attendees
- Email Aurelien: agents 0 and 2 approve, agent 1 rejects because another attendee also needs to be emailed
- Delete Team Planning Meeting: all agents reject because attendees should be emailed first

Voting: Delete Meditation wins (3 YES)

→ ✓ `Calendar__delete_calendar_event` (Meditation Session)

Steps 2-3: Notify Team Planning Meeting attendees

- In each step, all 3 agents independently propose sending emails to the same recipient
- During reasoning, each agent verifies the recipient is an attendee of the Team Planning Meeting
- During voting, the actions pass with unanimous agreement
- Emails sent: Aurélien ✓, Vigdis ✓

Steps 4-5: Notify Networking Event attendees

- Agent 2 observes that all Team Planning Meeting attendees were emailed, and the Meditation Session is already deleted. Next, Networking Event attendees must be notified
- In each step, all 3 agents independently propose sending emails to the same recipient
- During reasoning, each agent verifies the recipient is an attendee of the Networking Event
- During voting, the actions pass with unanimous agreement
- Emails sent: Aoife ✓, Kirsten ✓

Steps 6–7: Delete Monday events

- Agent 0 observes that all Monday event attendees have been emailed, and these events can now be deleted
- Agent 0 proposes deleting the Team Planning Meeting
- Agents 1 and 2 propose emailing Astrid, who is attending the Saturday Yoga Class
- During reasoning, all 3 agents unanimously pass both intents, so the 2 options move to a runoff voting round
- During runoff voting, all 3 agents unanimously agree to delete the Team Planning Meeting
- A similar process happens for the Networking Event
- `Calendar__delete`: Team Planning ✓, Networking Event ✓

Steps 8–9: Notify Saturday attendees

- Emails sent: Astrid ✓, Elodie ✓

Step 10: Delete Saturday event

- Voting confirms both attendees notified
- `Calendar__delete`: Yoga Class ✓

Step 11: Report to user

- All agents observe that all Monday - Saturday events were deleted and each attendee was emailed
- All agents propose sending a message to the user, with slightly different email text
- During voting, all 3 agents unanimously pass all 3 possible emails, so they move to a runoff voting round
- During runoff voting, all 3 agents unanimously agree to use email 2, which is the most informative and concise
- `AgentUserInterface__send_message_to_user`

Validation Result: ✓ PASS

Tool	Oracle	Agent
`Calendar__delete_calendar_event`	4	4 ✓
`EmailClientV2__send_email`	6	6 ✓

Figure 3. Trace of CoAct multi-agent coordination on the same Gaia2 task from Figure 2.