SCALING MULTI-AGENT INTELLIGENCE WITH SUBSTRATE-CENTRIC COORDINATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Existing large language model (LLM) based multi-agent systems have been shown to scale poorly, and we identify three problems: reading burden that grows with the group, useful information diluted by shared work of unknown quality, and fragility to failed agents. We argue that the problems are largely due to agent-centric designs: the coordination structure is wired into fixed workflows, central coordinators, or peer-to-peer messaging, and recent blackboard-style methods that move the coordination into a shared medium relieve only the last problem. To close the gaps, we build SCALEMAS, where asynchronous agents interact with a shared datastore of linked messages via three operations: publishing messages with optional references to earlier ones that they build on, searching for messages by content, and visiting messages by address. Search returns at most k messages, so an agent's read stays bounded at any group size; results are ranked by relevance and by how often they are cited, which we find to be an effective proxy for quality; and messages persist after their producers leave, so the group carries on when individual agents fail. Across four agentic tasks, SCALEMAS outperforms agent-centric and blackboard-style baselines at lower inference compute, continues to scale up to 128 agents, and tolerates losing agents midway.

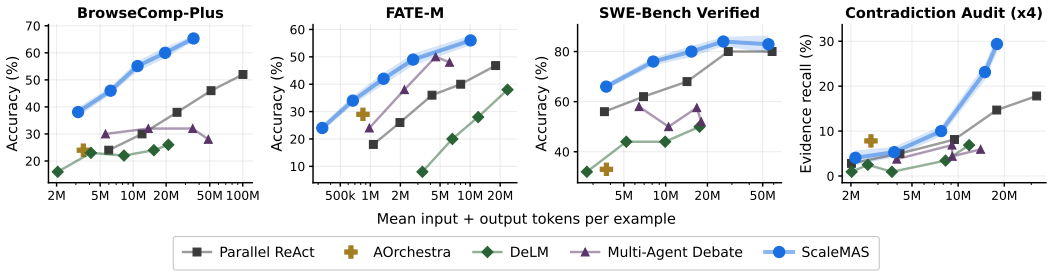

Figure 1: **SCALEMAS shows stronger scalability.** Task performance v.s. inference compute per example (log scale) with Qwen3.5-9B. Each point is one group size, scaled from 1 to 2, 4, 8, and 16 agents; AOrchestra, whose coordinator decides how many sub-agents to launch, is a single point. At matched compute, SCALEMAS reaches higher performance than Multi-Agent Debate (all-to-all message-passing), AOrchestra (coordinator-based), DeLM (blackboard-style shared context), and parallel ReAct agents without communication, and it improves with group size up to eight agents on every benchmark.

1 INTRODUCTION

Scaling a group of cooperative agents should increase its collective capability, but this benefit is not automatic in current large language model (LLM) based multi-agent systems (MAS). Empirical studies report superlinearly increased inference compute, diminishing or negative returns from additional agents, and settings in which a group underperforms a single agent given the same inference budget (Smit et al., 2023; Zhang et al., 2026; Kim et al., 2026b; Zheng et al., 2026).

We identify three major problems that grow with the number of agents and that many of the current designs leave unsolved. First, what each agent must read increases with the group: message-passing systems deliver all senders' outputs to a receiver agent (Du et al., 2023; Liu et al., 2026), and coordinator-based systems route every sub-agent's result into a central agent's context (Su et al.,

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

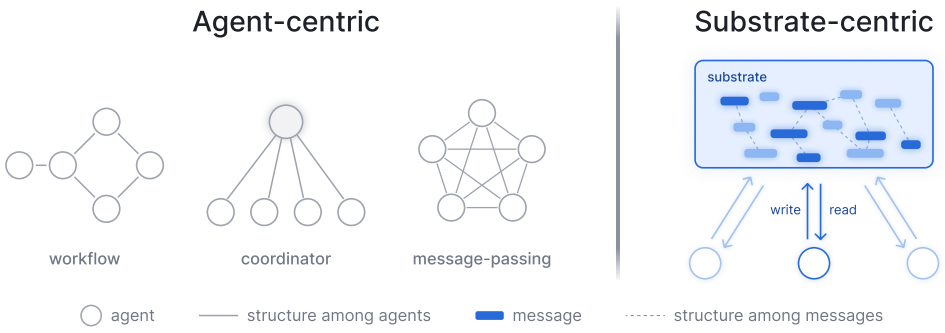

Figure 2: **Two design principles for multi-agent coordination. Left:** agent-centric designs wire coordination structure into the agents. A workflow-based system fixes agents' successors, a coordinator-based system routes results through one main agent, and a message-passing system lets agents address others. **Right:** substrate-centric designs place the coordination in a shared medium. Messages are pushed to a datastore, organized with certain structures; agents interact only with the medium by writing to and reading from it.

2025; Ruan et al., 2026), thus a larger group floods its readers with input that could exceed context limits. Second, the useful information dilutes as the group grows: even with bounded reading, an agent must find the few relevant results in an ever larger body of partial work shared by others, and it is hard to tell the quality of the shared information. Third, failure becomes the routine rather than the exception: the more agents run, the more likely some of them stall or crash, yet workflow-based systems fix each agent's role and successor at design time (Hong et al., 2024; Qian et al., 2024) and coordinator-based systems route everything through one agent (Cemri et al., 2026), so a lost agent could easily cause the system to crash.

What these methods share is that they adopt *agent-centric* designs by wiring coordination structures into agents' instructions and loops. Another design principle has been adopted for decades since early blackboard systems (Newell, 1962; Nii, 1986; Erman et al., 1980): rather than engineering agents for better coordination, one can design the *substrate*[1] where agents only interact with a shared environment and self-coordinate (Figure 2). Recent work in LLM-based multi-agent systems has been trying to adopt this principle by giving LLM agents either a shared context (Mao & Mirhoseini, 2026; Park et al., 2026; Qi et al., 2026) or a shared filesystem (Qu et al., 2026; Kim et al., 2026a). This could alleviate the third problem: no agent waits on a particular peer, so a stalled agent blocks no one; and results persist after their producers leave, so a failed agent loses only its unfinished work. However, it leaves the first two problems unsolved. A shared context grows linearly with the group size, which still could cause context overflow; a shared filesystem bounds nothing, and leaves each agent to search a growing pile of files with no signal of their quality. A scalable substrate for LLM agents therefore needs more than a shared medium: it must bound each agent's reading as the group's output grows, while maintaining effective information discovery for every agent.

Both requirements have a working precedent at the scale of billions of readers and writers: the Web. Its scalability comes in part from avoiding readers having to read everything: search bounds attention by returning a small set of relevant results, while hypertextual links make information navigable on demand and also provide signals of its importance (Brin & Page, 1998; Page et al., 1999). Drawing on these principles, we build ScaleMAS, a task-agnostic substrate on which LLM agents self-coordinate at scale. The agent substrate maintains a datastore of persistent messages interconnected through references, and a search engine indexed over them. Agents run asynchronously, and each interacts only with the substrate through three operations: it *publishes* messages to the datastore, optionally referencing earlier ones; it *searches* for messages by content and gets at most k returned results; and it *visits* messages for details by address. When a new message references earlier ones, the referenced messages gain importance weight, which combines with lexical relevance to rank subsequent search results. Every message is displayed with its references and with the later messages that reference it, but never with its author or its weight.

Several properties follow from the design. First, an agent takes in bounded messages per read operation, regardless of how many agents are writing or how large the datastore has grown. Second,

[1]Throughout this paper, we define an agent substrate as a combination of: a datastore that organizes agents' messages, operations for reads and writes, and deterministic rules that connect the datastore and the operations.

the importance of shared information is earned through reuse: weight accumulates only when others build on a message, so search surfaces what the group has actually relied on, and the signal is enhanced as the datastore grows. Third, membership is flexible to change. Because messages outlive their producers, a newcomer reads the datastore and can catch up by searching for and visiting messages, and an agent that leaves takes nothing with it. Moreover, lineage of shared results is inspectable, as each message carries references and its citers, and a reader can check what a claim rests on before building on it.

We evaluate ScaleMAS on four agentic tasks spanning deep research (Chen et al., 2025b), theorem proving (Jiang et al., 2026), software engineering (Jimenez et al., 2024), and contradiction audit, a benchmark we construct in which agents identify contradictions in large document corpora. We use two LLMs as backbones and compare against multi-agent debate (all-to-all message-passing) (Du et al., 2023), DeLM (blackboard-style shared context) (Mao & Mirhoseini, 2026), AOrchestra (coordinator-based delegation) (Ruan et al., 2026), and parallel ReAct agents without communication (Yao et al., 2022). Our experiments examine how collective performance and inference compute change with group size, with additional scaling studies extending to 128 agents. We also evaluate resilience to unreliable membership and ablate citation-based ranking to assess its contribution. A group of N agents with one failed agent performs comparably to $N - 1$ healthy agents, indicating that the remaining agents can continue cooperating despite the loss of individual participants. These results suggest that multi-agent systems scale further when coordination is designed into the substrate the agents share rather than wired into the agents themselves.

2 Method

ScaleMAS is an agent substrate with three parts: 1) a datastore of messages, 2) three operations that agents use to write to it and read from it, and 3) deterministic rules that decide what is written and what a read returns. N agents run the same harness under the same prompt. Each is an ordinary tool-using LLM agent that keeps the tools of its task and gains the three extra operations, which are its only access to the datastore. No agent holds a role, sends messages to another agent, or assigns work to other agents, and the agents run asynchronously. Figure 3 shows an example that traces two agents through the substrate.

2.1 The datastore

The datastore D is an append-only sequence of messages. A message $m = (u, a, r, R)$ records one piece of completed work: the sub-task u the agent carried out, stated in the words someone about to do the same work would search for; the artifact a that the work produced; a short rationale r; and a set R of references to what the work built on. A reference names either an earlier message or a resource exposed by the task's own tools, such as a corpus snippet or a repository file. Each message receives the next address when it is written and is never edited or deleted, so a reference always resolves to the text it was written against, and the references form a directed acyclic graph over D that grows only at its newest end. Each message also carries a citation weight $s_m \in [0, 1)$, which the rules maintain and which search uses for ranking. The sub-task and artifact together are limited to 256 or 1,024 tokens depending on the benchmark (Table 4), and a message carries at most 32 references.

2.2 The operations

`publish`(u, a, r, R) appends a message and makes it readable by every agent from that moment on. The two read operations match the two ways a reader can know what it wants. `search`(q) takes a free-text description of what the agent needs and returns the k highest-ranked messages. `visit`$(m_1, \ldots, m_j)$ returns the named messages, for an agent that a search result or a message's references has already pointed somewhere specific. Both return each message as its four fields, the messages it references, and the most recent of the messages that reference it, up to eight. The author and the citation weight are never shown, so a reader judges a message by its content and its links. An agent ends by submitting its answer together with references and a rationale in the same form as a message, except on SWE-bench Verified, where the answer is a patch. The messages an

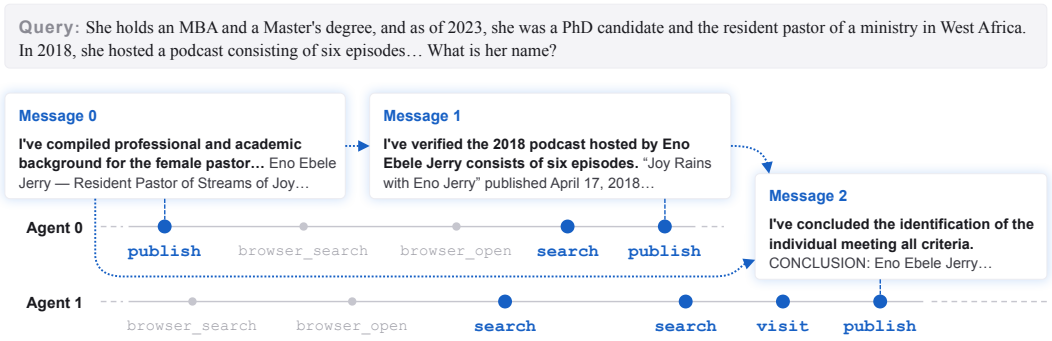

Figure 3: **Example of two agents working through the substrate** on a deep research query. Each row is one agent's trajectory, with the three operations in blue and the task's own tools in gray. A dashed stem marks the step that published a message, and a dotted arrow points from a referenced message to the message that cites it.

answer cites, followed transitively through their references, form the answer's *evidence chain*, which Section 3.4 uses to trace where correct answers come from.

A walk-through. Figure 3 shows how the operations combine on a deep research query. Agent 0 publishes Message 0, a candidate answer with the background it has gathered, then browses further, searches the datastore, and publishes Message 1, which verifies one clue of the query and cites Message 0. Agent 1 has meanwhile browsed on its own; it searches the datastore, finds Message 1, follows its reference to Message 0 with `visit`, and publishes Message 2, which combines the two into a conclusion and cites both. Neither agent addresses the other or waits for it: Agent 1 learns of Agent 0's work only by reading the datastore, and its final message rests on evidence it did not gather itself.

2.3 The rules

What is written. `publish` accepts a message only if every reference names something that exists and the rationale names every reference and states what it contributed. A message that breaks either rule is rejected with the offending reference named, and nothing is written. The first rule keeps references pointing backward in time. The second turns a reference into a stated dependency that a reader can check against the cited text, rather than an incidental mention. These are the only constraints ScaleMAS places on what an agent writes.

How weight is assigned. A new message starts at weight $s_0 = 0$. When a message references m, the weight of m moves a fraction α of the way toward one, $s_m \leftarrow s_m + \alpha(1 - s_m)$, once per citing message, and at every step of the group all weights decay, $s_m \leftarrow \gamma s_m$. A message that later work builds on therefore gains weight, and one that the group stops building on loses it. References to task resources carry no weight.

What a read returns. The search index ranks messages by

$$\text{score}(q, m) = s_m + \frac{\text{Rel}(q, m)}{\max_{m' \in D} \text{Rel}(q, m')}. \tag{1}$$

The citation weight plus a retrieval score normalized to $[0, 1]$; any retrieval function can serve as Rel, and we use BM25 (Robertson & Zaragoza, 2009). Only messages with a positive retrieval score are ranked, ties go to the earlier message, and the top k are returned, so the weight reorders relevant messages but cannot surface an unrelated one. The citation weight follows the idea of PageRank (Brin & Page, 1998; Page et al., 1999), with links written by the agents as they work. `visit` likewise accepts at most k addresses per call. Every read therefore adds at most k messages of bounded length to an agent's context, however many agents are writing and however large D has grown: an agent's intake grows with how often it reads, which the agent decides, and not with the size of the group.

Table 1: **Example of a cross-file contradiction from the SymPy documentation corpus.** The injected text is in red, and the clashing claims are underlined.

`modules/solvers/solveset.rst`	`modules/sets.rst` (edited)
... Set objects conform to Pythonic conventions when possible, i.e., `x in A` and `for i in A` both work when they can be computed.	... By contrast, `x in y` always gives `TypeError` for SymPy sets, even when the membership could easily be computed.

3 EXPERIMENTS

3.1 CONTRADICTION AUDIT

Few existing tasks allow multi-agent systems to be studied at different scales under control. Also, on many existing agentic benchmarks a single agent already performs well, which leaves little room to measure how performance changes with group size. We therefore construct a new task, Contradiction Audit, which is designed to reward combining information spread across agents: The agents receive a large software documentation corpus with injected factual contradictions, and are tasked with reporting as many pairs of contradicting passages as they can find. The corpus can grow while the injected contradictions stay fixed, so difficulty can be varied without changing the task. More details are given in Appendix B.

Benchmark construction. We source the Markdown and reStructuredText documentation files from 16 open-source software projects, including CPython, the Linux kernel, and PyTorch, and exclude files that disagree with the rest by design (changelogs, release notes, deprecation notices). We use GPT-5.6 to select pairs of document snippets on a common subject and rewrite the prose of one to contradict the other. An edit is kept only if a `diff` against the original snippet shows that it changes a few sentences and leaves the rest intact, and a second round of LLM judge (also with GPT-5.6) confirms that the edited snippet reads naturally, is not trivial (so that LLMs will not find it by memorization), is self-consistent, and truly conflicts with its partner. The LLM judge will run four times, and a snippet will be accepted iff. all four judges are positive. Each corpus receives twenty injected contradictions over forty distinct snippets, ten within one file and ten across files.

Difficulty and scoring. Level $\times n$ pads the forty contradiction snippets with seeded random distractors to $40n$ snippets, so the corpus grows while the answers stay fixed. For example, a $\times 8$ split means besides the 20 pairs of snippets that a system should really focus on, there are $40 \times 8 - 40 = 280$ other irrelevant snippets as distractions. We use a relaxed metric for evaluation: A contradiction counts as found when a reported pair includes its edited snippet, and extra reports are not penalized. The score is the fraction of the twenty found, averaged over 16 corpora. In the example of Table 1, each passage reads naturally on its own and the two sit in different documents, so detecting the contradiction requires reading one snippet and locating the other.

3.2 SETUP

Benchmarks. Beyond Contradiction Audit, we select three more benchmarks that stress a group of LLM agents in different ways. **BrowseComp-Plus** (Chen et al., 2025b) is search-intensive: the evidence for each question is scattered across a large corpus of web documents, reached through a search index. **FATE** (Jiang et al., 2026) is reasoning-intensive: each theorem proving problem asks for a complete Lean 4 proof that the compiler checks. **SWE-bench Verified** (Jimenez et al., 2024) measures the resolution of real GitHub issues, and requires multiple steps of verification and revision. SCALEMAS is harness-agnostic: we run it on a standard ReAct loop (Yao et al., 2022), implemented with the Hugging Face smolagents library[2], for BrowseComp-Plus, FATE, and Contradiction Audit, and on mini-SWE-agent (Yang et al., 2024) for SWE-bench Verified. More benchmark details are presented in Appendix C.3.

Baselines and parity. We compare against the following methods: Parallel **ReAct** (Yao et al., 2022) runs N agents without communication between them. **Multi-Agent Debate (MAD)** (Du et al., 2023) is agent-centric with all-to-all message passing: agents exchange their intermediate work, each reading what every other produced. **AOrchestra** (Ruan et al., 2026) is agent-centric

[2] https://github.com/huggingface/smolagents

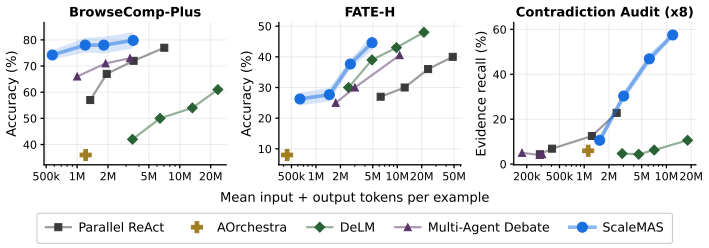

Figure 4: **Scaling study with Claude-5-Sonnet.** Task performance v.s. inference compute per example (log scale) on BrowseComp-Plus, FATE, and Contradiction Audit. Each point is one group size, scaled from 1 to 2, 4, and 8 agents. At matched compute, ScaleMAS outperforms every baseline, and its lead on Contradiction Audit widens as the group grows.

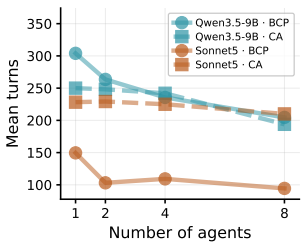

Figure 5: **Number of turns**, averaged over agents and examples, on BrowseComp-Plus (BCP) and Contradiction Audit x8 (CA) with both LLMs.

with a central coordinator: a main agent decomposes the task and delegates to sub-agents, each of which reports back once. **DeLM** (Mao & Mirhoseini, 2026) is blackboard-style: agents write to a shared context that all of them read, without addressing one another. For parallel ReAct, MAD, DeLM, and ScaleMAS, an agent stops once it outputs a final answer, and the system stops when all agents stop. For AOrchestra, the system stops once the coordinator agent stops. Every method runs with **Qwen3.5-9B** on all four benchmarks and with **Claude-5-Sonnet** on all but SWE-bench Verified. The stronger model gets the harder benchmark splits (FATE-H and Contradiction Audit x8; Qwen3.5-9B gets FATE-M and x4). All methods share identical benchmark task prompts, per-benchmark limit on action steps, and the benchmark's tools and evaluation harness.

3.3 Main Results

Performance grows with group size. Figures 1 and 4 plot performance against the inference compute per example, one point per group size. With Qwen3.5-9B, scaling ScaleMAS from one agent to sixteen raises performance on every benchmark, from 38.1% to 65.3% on BrowseComp-Plus, from 24.0% to 56.0% on FATE-M, and from 4.1% to 29.4% on Contradiction Audit x4. On SWE-bench Verified, it levels off after eight agents, where parallel ReAct saturates as well, which points to the limit of the LLM's capability. Parallel ReAct also improves with group size but stays below ScaleMAS at every group size on all four benchmarks. Part of this gap is present with a single agent, where the system degrades to a single agent with a private substrate; Table 2 separates this effect from the benefit of reading one another's messages.

The baselines behave differently: on BrowseComp-Plus and Contradiction Audit, adding agents to MAD or DeLM helps little and sometimes hurts, and neither exceeds 7% on Contradiction Audit at any group size. Both methods deliver the group's outputs into each agent's context, so adding agents enlarges the reading burden of every member. Experiments with Claude-5-Sonnet show similar patterns (Figure 4), most visibly on Contradiction Audit x8, where ScaleMAS rises from 10.6% with one agent to 57.5% with eight.

Scaling is more compute-efficient. At matched inference compute, ScaleMAS is ahead of baselines on every benchmark (Figures 1 and 4). For example, with Qwen3.5-9B (Figure 1), at a similar compute of about 20M tokens per example on BrowseComp-Plus, 8 ScaleMAS agents reach 60.0% while 16 DeLM agents reach 26.0%; and on SWE-bench Verified, 4 ScaleMAS agents reach the same 80.0% as 8 ReAct agents with about half the tokens (15.3M against 28.3M). The difference grows with the group: independent ReAct agents repeat each other's searching and reading, so their tokens per example roughly double with every doubling of the group, whereas on BrowseComp-Plus and Contradiction Audit the tokens used by ScaleMAS grow more slowly than its size, because agents reuse what others have already published instead of rediscovering it. The pattern holds with Claude-5-Sonnet: at about 3.5M tokens per example on BrowseComp-Plus, eight ScaleMAS agents reach 79.8%, against 73.0% for eight MAD agents and 72.0% for four ReAct agents. Adding agents to MAD, in contrast, increases tokens without improving accuracy: on BrowseComp-Plus with Qwen3.5-9B, its tokens per example grow from 5.6M with two agents to 48.5M with sixteen, while its accuracy stays between 28% and 32%.

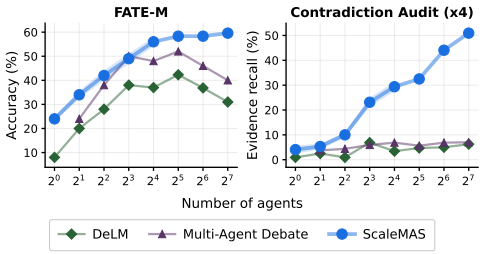

Figure 6: **Scaling to 128 agents** with Qwen3.5-9B on FATE-M and Contradiction Audit x4, comparing ScaleMAS to other multi-agent systems: DeLM and MAD.

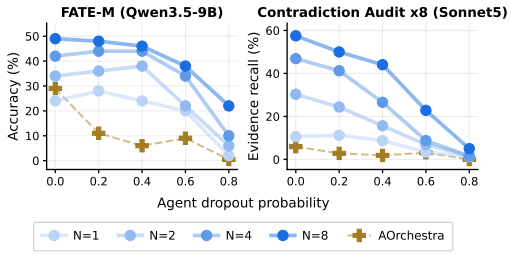

Figure 7: **Resilience to unreliable membership.** Each agent drops out of each step independently with probability p. FATE-M with Qwen3.5-9B and Contradiction Audit x8 with Claude-5-Sonnet, 1 to 8 agents, AOrchestra for reference.

Agents take fewer turns in larger groups. Figure 5 reports the mean number of turns an agent takes before it submits. It falls as the group grows, from 304 with one agent to 205 with eight on BrowseComp-Plus with Qwen3.5-9B and from 150 to 94 with Claude-5-Sonnet, and it also falls on Contradiction Audit with both backbones. The total work of the group therefore grows more slowly than its size: eight Qwen3.5-9B agents on BrowseComp-Plus take about 5.4 times as many turns in total as a single agent. One reason is that an ScaleMAS agent that finds the result it needs already published can build on it instead of deriving it again; Section 3.4 further tests whether correct answers indeed draw on other agents' work. The turns an agent saves are mostly calls to the task's own tools: with Claude-5-Sonnet on BrowseComp-Plus, an agent calls them 80 times per example in a group of eight against 134 times alone. On Contradiction Audit x8, these calls fall from 131 to 98 while the agent's searches of the datastore rise from 12 to 65, so agents in larger groups explore the task less on their own and read more of what the others have found.

3.4 Analysis

Scaling continues to 128 agents. Figure 6 extends the sweep to $N = 128$ agents with Qwen3.5-9B. On FATE-M, ScaleMAS makes most of its improvement by $N = 16$ and then levels off just below 60%, whereas MAD peaks at $N = 32$ and declines afterwards, and DeLM ends lower at $N = 128$ than at $N = 8$. On Contradiction Audit x4, ScaleMAS improves at every doubling and reaches 50.9% at $N = 128$, while DeLM and MAD never exceed 7%. The baselines thus show the diminishing or negative returns that prior studies report when multi-agent systems grow (Kim et al., 2026b), which we attribute to their unbounded reads. In turn, ScaleMAS keeps agents' reads lightweight and turns additional agents into higher performance.

Peer citations automatically curate the datastore. The substrate never judges the quality of a message; the only quality-related signal is which messages the agents choose to cite. We test whether this signal is informative on all BrowseComp-Plus runs of ScaleMAS, using two complementary criteria. The first is *use*: whether a message lies on the evidence chain of a correct final answer, that is, whether the answer cites it directly or through a chain of references. Ranking the messages within each run by citation count, the fraction on such a chain increases monotonically from 29% in the least-cited quintile to 85% in the most-cited (Figure 8a). Because the evidence chain itself follows citations, this measure reflects connectivity as well as relevance.

The second criterion is independent of any answer: whether a message names a gold evidence document annotated by BrowseComp-Plus. Among messages that name a source, those naming gold evidence receive about twice as many citations from other agents as those naming other documents (Figure 8b). These results show that, without any central ranking, peer citations can effectively single out the messages that matter: those that final answers build on, and those that hold genuine evidence.

Correct answers are assembled across agents. Using BrowseComp-Plus runs with Qwen3.5-9B, we ask whether a correct answer's evidence chain contains a message, written by another agent, that names a gold document the submitting agent never opened itself. The share of correct answers built this way grows with the group, from 9% with two agents to 24% with eight, and the trend holds under a stricter criterion that also excludes documents the submitter saw in its own search

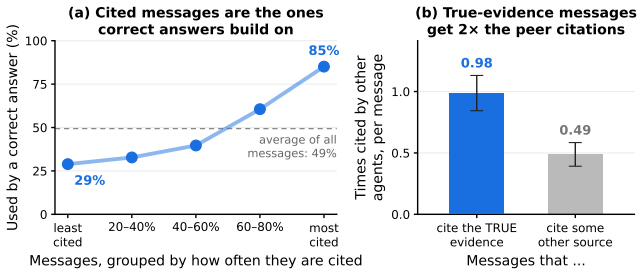

Figure 8: **Peer citations curate the datastore** (BrowseComp-Plus, all ScaleMAS runs). **(a)** Share of messages on the evidence chain of a correct answer, by fifth of within-run citation count, over the 413 examples whose correct answer cites a message; dashed: all messages. **(b)** Mean citations from other agents for source-citing messages that name a gold evidence document or another document, weighting examples equally. Error bars: 95% bootstrap CIs.

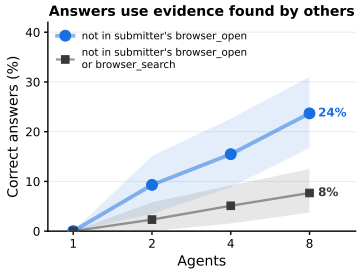

Figure 9: **Correct answers use evidence found by peers** (BrowseComp-Plus, Qwen3.5-9B). Share of correct answers whose citation chain holds other's message naming a gold document the submitter never opened, or never saw in its searches either.

results (Figure 9). This indicates that a division of labor emerges although no agent is ever assigned a sub-task: agents increasingly answer by combining evidence that others found, and more correct answers rest on it as the group grows.

Individual failures do not break the group. Figure 11 adds one agent that fails on every step (i.e. it always returns error messages), holding the number of healthy agents fixed. For ScaleMAS, a group with one failing agent scores about as well as its healthy agents alone; with seven healthy agents, for instance, both reach 19.4%. This is because ScaleMAS does not assign jobs to agents, and does not require an agent to finish anything: a failing agent simply publishes nothing, while the healthy agents keep finding and building on each other's messages, so the group loses only the failing agent's accumulated context. DeLM, in contrast, falls below 1% when one of its members fails. This may be partially due to its task decomposition design, which could be susceptible to individual failures when the errors are injected in the decomposition step.

Agents can leave and rejoin at any time. In Figure 7, each agent drops out of each step independently with probability p, so agents keep leaving and returning over the course of a run; at $p = 0.8$, an agent is absent from four of every five steps on average and the group's membership changes constantly. ScaleMAS keeps working throughout: even at $p = 0.8$, eight agents still solve 22.0% of FATE-M problems, and on Contradiction Audit x8, eight agents that drop out of two of every five steps still score higher than every baseline without dropout. This is because nothing in ScaleMAS waits on a particular agent, and what an agent has published stays in the datastore while it is away, so a returning agent picks up the group's progress and continues. Larger groups absorb dropout better (at $p = 0.8$ on FATE-M, 22.0% with eight agents against 2.0% with one), whereas AOrchestra, which routes every result through a single coordinator, loses more than half of its score already at $p = 0.2$ because it cannot tolerate the loss of the coordinator.

3.5 Ablation Studies

Does the citation weight improve what search returns? Figure 10 compares the default ranking, which adds the citation weight to lexical relevance, with BM25 alone and with a random order over matching messages. Removing the citation weight lowers accuracy at every group size on BrowseComp-Plus and Contradiction Audit x4 without reducing tokens read: at $N = 8$, 60.0% against 40.8% and 23.1% against 9.7%. On FATE-M the two stay within four points up to $N = 4$ and separate at $N = 8$ (49.0% against 42.0%), where BM25 alone also reads twice the tokens (5.7M against 2.7M). A random ranking does as well as BM25 alone at $N = 8$ on all three benchmarks, so lexical relevance alone recovers little of what the citation weight adds.

Is the improvement from communication, or from running more agents? Table 2 holds the group at four agents and removes communication in two steps. A private substrate, partitioned by author so that each agent reads only its own messages, lowers accuracy on every benchmark, by 8.3 points on BrowseComp-Plus and 8.4 on SWE-bench Verified. Removing the datastore, which leaves four independent ReAct agents, lowers it further on three of the four benchmarks (on Contradiction

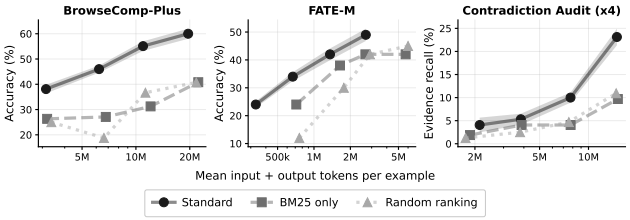

Figure 10: **Ablating citation weight in search ranking.** Task performance versus mean tokens per example with Qwen3.5-9B, ranking by lexical relevance plus the citation weight (standard), by BM25 alone, or at random. Each point is a group size from 1 to 8.

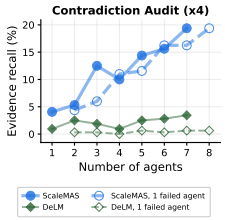

Figure 11: **One permanently failing agent** on Contradiction Audit `x4` with Qwen3.5-9B. Dashed: one agent always fails; solid: all agents work.

Table 2: **Ablating cross-agent communication.** Task performance (%) with four Qwen3.5-9B agents. *Private substrate* partitions the datastore by author, so each agent reads only its own messages; *w/o substrate* runs independent ReAct agents.

	BrowseComp-Plus	FATE-M	SWE-bench Verified	Contradiction Audit (x4)
ScaleMAS	**55.1**	**42.0**	**80.0**	**10.0**
↳ private substrate	46.8	38.0	71.6	6.6
↳ w/o substrate (ReAct)	38.0	36.0	68.0	8.1

Audit, 8.1% against 6.6%). Reading one another's messages accounts for about half or more of the improvement over independent sampling.

4 Related Work

LLM-based multi-agent systems revisit three coordination structures from classic distributed problem solving: agents that address one another directly, a designated agent that mediates, and a shared blackboard. Agents address one another directly in (Du et al., 2023) and along the edges of a topology in (Qian et al., 2025); a designated agent mediates in (Fourney et al., 2024) and (Ruan et al., 2026); and the blackboard has been revived repeatedly, with (Han & Zhang, 2025) removing direct contact between agents entirely and (Mao & Mirhoseini, 2026) placing typed notes in a context every agent reads. A further line asks which agents should be connected: Zhuge et al. (2024) optimizes graph connectivity together with node prompts, and Zhang et al. (2024) generates a task-specific topology. These methods change who is connected to whom. What arrives in a reader's context is still chosen by whoever sends along those edges, so the question of who selects is left where these structures put it. We discuss earlier work and further related lines in Appendix A.

5 Conclusion

We identified three problems that grow with the number of agents: the reading burden on each agent, the dilution of useful shared work of unknown quality, and fragility to failing agents. ScaleMAS addresses them with an agent substrate: agents interact only through a shared datastore of linked messages, every read returns at most k messages ranked by relevance and by how often they are cited, and messages persist after their producers leave. Across four benchmarks and two backbones, ScaleMAS improves with group size up to eight agents on every benchmark, and with Qwen3.5-9B up to sixteen on three of the four, while reaching higher performance than message-passing, coordinator-based, and blackboard-style baselines at matched inference compute. Our analysis shows the mechanisms at work: peer citations concentrate attention on messages that hold genuine evidence, correct answers increasingly draw on evidence found by other agents, and the group tolerates both a permanently failing agent and agents that drop out at any step.

Limitations and future work. Due to limited compute, our experiments scale groups up to 128 agents; larger groups are left to future work. ScaleMAS is also used only at inference time, and training agents to coordinate through the substrate is another direction for future work.

AI USE STATEMENT

In this work, we used generative AI tools to assist with grammar corrections and generate synthetic data sets. We have not used generative AI tools to help design or provide feedback on research methodology or experiments, implement methods, assist with translation, or interpret results. Developing theoretical models or conceptual frameworks, formulating mathematical claims, providing critical ingredients for proving mathematical claims, assisting in the writing of proofs, proposing or refining hypotheses, and cleaning or reformatting data sets are not applicable to this work. We have reviewed all AI-assisted work. AI-assisted modifications were reviewed and edited by the authors to ensure they faithfully convey the intended meaning. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REFERENCES

Nicholas J Belkin and W Bruce Croft. Information filtering and information retrieval: Two sides of the same coin? *Communications of the ACM*, 35(12):29–38, 1992. 15

Tim Berners-Lee. Information management: A proposal. *http://www. w3. org/History/1989/proposal. html*, 1989. 15

Sergey Brin and Lawrence Page. The anatomy of a large-scale hypertextual web search engine. *Computer networks and ISDN systems*, 30(1-7):107–117, 1998. 2, 4, 15

Vannevar Bush et al. As we may think. *The atlantic monthly*, 176(1):101–108, 1945. 15

Mert Cemri, Melissa Z Pan, Shuyi Yang, Lakshya A Agrawal, Bhavya Chopra, Rishabh Tiwari, Kurt Keutzer, Aditya Parameswaran, Dan Klein, Kannan Ramchandran, et al. Why do multi-agent llm systems fail? *Advances in Neural Information Processing Systems*, 38, 2026. 2

Weize Chen, Jiarui Yuan, Chen Qian, Cheng Yang, Zhiyuan Liu, and Maosong Sun. Optima: Optimizing effectiveness and efficiency for llm-based multi-agent system. In *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 11534–11557, 2025a. 15

Zijian Chen, Xueguang Ma, Shengyao Zhuang, Ping Nie, Kai Zou, Andrew Liu, Joshua Green, Kshama Patel, Ruoxi Meng, Mingyi Su, et al. Browsecomp-plus: A more fair and transparent evaluation benchmark of deep-research agent. *arXiv preprint arXiv:2508.06600*, 2025b. 3, 5, 18

Danny Dolev and Tomás Feder. Multiparty communication complexity. In *30th Annual Symposium on Foundations of Computer Science*, pp. 428–433. IEEE Computer Society, 1989. 15

Yilun Du, Shuang Li, Antonio Torralba, Joshua B Tenenbaum, and Igor Mordatch. Improving factuality and reasoning in language models through multiagent debate. *arXiv preprint arXiv:2305.14325*, 2023. 1, 3, 5, 9

E. H. Durfee, V. R. Lesser, and D. D. Corkill. Trends in cooperative distributed problem solving. *IEEE Trans. on Knowl. and Data Eng.*, 1(1):63–83, March 1989. ISSN 1041-4347. doi: 10.1109/69.43404. URL https://doi.org/10.1109/69.43404. 15

Lee D. Erman, Frederick Hayes-Roth, Victor R. Lesser, and D. Raj Reddy. The hearsay-ii speech-understanding system: Integrating knowledge to resolve uncertainty. *ACM Comput. Surv.*, 12(2): 213–253, June 1980. ISSN 0360-0300. doi: 10.1145/356810.356816. URL https://doi.org/10.1145/356810.356816. 2, 15

Adam Fourney, Gagan Bansal, Hussein Mozannar, Cheng Tan, Eduardo Salinas, Friederike Niedtner, Grace Proebsting, Griffin Bassman, Jack Gerrits, Jacob Alber, et al. Magentic-one: A generalist multi-agent system for solving complex tasks. *arXiv preprint arXiv:2411.04468*, 2024. 9

David Gelernter. Generative communication in linda. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 7(1):80–112, 1985. 15

Bochen Han and Songmao Zhang. Exploring advanced llm multi-agent systems based on blackboard architecture. *arXiv preprint arXiv:2507.01701*, 2025. 9

Sirui Hong, Mingchen Zhuge, Jonathan Chen, Xiawu Zheng, Yuheng Cheng, Jinlin Wang, Ceyao Zhang, Steven Yau, Zijuan Lin, Liyang Zhou, et al. Metagpt: Meta programming for a multi-agent collaborative framework. In *International Conference on Learning Representations*, volume 2024, pp. 23247–23275, 2024. 2

Jiedong Jiang, Wanyi He, Wang Yuefeng, Guoxiong Gao, Yongle Hu, Jingting Wang, Nailin Guan, Peihao Wu, Bryan Dai, Liang Xiao, et al. Fate: A formal benchmark series for frontier algebra of multiple difficulty levels. In *International Conference on Learning Representations*, volume 2026, pp. 10543–10580, 2026. 3, 5, 18

Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. Swe-bench: Can language models resolve real-world github issues? In *International Conference on Learning Representations*, volume 2024, pp. 54107–54157, 2024. 3, 5, 18

To Eun Kim, Xuhong He, Dishank Jain, Ambuj Agrawal, Negar Arabzadeh, and Fernando Diaz. Multi-agent transactive memory. *arXiv preprint arXiv:2606.19911*, 2026a. 2

Yubin Kim, Ken Gu, Chanwoo Park, Chunjong Park, Samuel Schmidgall, A Ali Heydari, Yao Yan, Zhihan Zhang, Yuchen Zhuang, Yun Liu, et al. Capable language models can outgrow the benefits of collaboration. *Nature Machine Intelligence*, 8(7):1157–1172, 2026b. 1, 7

Xuecheng Liu, Daman Arora, Gokul Swamy, and Andrea Zanette. Message passing enables efficient reasoning. *arXiv preprint arXiv:2607.01077*, 2026. 1

Yuzhen Mao and Azalia Mirhoseini. Decentralized multi-agent systems with shared context. *arXiv preprint arXiv:2606.10662*, 2026. 2, 3, 6, 9, 18

Allen Newell. Some problems of basic organization in problem-solving programs. 1962. URL https://api.semanticscholar.org/CorpusID:60835107. 2, 15

H. Penny Nii. Part one: The blackboard model of problem solving and the evolution of blackboard architectures. *AI Mag.*, 7(2):38–53, June 1986. ISSN 0738-4602. doi: 10.1609/aimag.v7i2.537. URL https://doi.org/10.1609/aimag.v7i2.537. 2

Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The pagerank citation ranking : Bringing order to the web. In *The Web Conference*, 1999. URL https://api.semanticscholar.org/CorpusID:1508503. 2, 4

Jongho Park, Vasilis Kontonis, Shivam Garg, Akshay Krishnamurthy, and Dimitris Papailiopoulos. Scaling discovery through test-time communication, 2026. URL https://arxiv.org/abs/2609.21032. 2, 18

Joon Sung Park, Joseph O'Brien, Carrie Jun Cai, Meredith Ringel Morris, Percy Liang, and Michael S Bernstein. Generative agents: Interactive simulacra of human behavior. In *Proceedings of the 36th annual acm symposium on user interface software and technology*, pp. 1–22, 2023. 15

Zhenting Qi, Jack Fan, Karly Hou, Daisy Xinlei Lin, Satyaki Chakraborty, Mete Kemertas, Yu Yao, Sophia Simeng Han, Han Zheng, Zihan Wang, Xiangjun Wang, Himabindu Lakkaraju, and Yilun Du. Coact: A minimal coordination substrate for decentralized LLM agents. In *Workshop on Failure Modes of Agentic AI at ICML 2026*, 2026. URL https://openreview.net/forum?id=AvxJqQW1X5. 2

Chen Qian, Wei Liu, Hongzhang Liu, Nuo Chen, Yufan Dang, Jiahao Li, Cheng Yang, Weize Chen, Yusheng Su, Xin Cong, et al. Chatdev: Communicative agents for software development. In *Proceedings of the 62nd annual meeting of the association for computational linguistics (volume 1: Long papers)*, pp. 15174–15186, 2024. 2

Chen Qian, Zihao Xie, YiFei Wang, Wei Liu, Kunlun Zhu, Hanchen Xia, Yufan Dang, Zhuoyun Du, Weize Chen, Cheng Yang, et al. Scaling large language model-based multi-agent collaboration. In *International Conference on Learning Representations*, volume 2025, pp. 41488–41505, 2025. 9

Ao Qu, Han Zheng, Zijian Zhou, Yihao Yan, Yihong Tang, Shao Yong Ong, Fenglu Hong, Kaichen Zhou, Chonghe Jiang, Minwei Kong, et al. Coral: Towards autonomous multi-agent evolution for open-ended discovery. *arXiv preprint arXiv:2604.01658*, 2026. 2

Stephen Robertson and Hugo Zaragoza. The probabilistic relevance framework: Bm25 and beyond. *Found. Trends Inf. Retr.*, 3(4):333–389, April 2009. ISSN 1554-0669. doi: 10.1561/1500000019. URL https://doi.org/10.1561/1500000019. 4

Jianhao Ruan, Zhihao Xu, Yiran Peng, Fashen Ren, Zhaoyang Yu, Xinbing Liang, Jinyu Xiang, Yongru Chen, Bang Liu, Chenglin Wu, et al. Aorchestra: Automating sub-agent creation for agentic orchestration. *arXiv preprint arXiv:2602.03786*, 2026. 2, 3, 5, 9

Andrew Semenov and Svyatoslav Dorofeev. Beyond compaction: Structured context eviction for long-horizon agents. *arXiv preprint arXiv:2606.11213*, 2026. 15

Aditi Singh, Abul Ehtesham, Saket Kumar, Tala Talaei Khoei, and Athanasios V Vasilakos. Agentic retrieval-augmented generation: A survey on agentic rag. *arXiv preprint arXiv:2501.09136*, 2025. 15

Andries Smit, Paul Duckworth, Nathan Grinsztajn, Thomas D Barrett, and Arnu Pretorius. Should we be going mad? a look at multi-agent debate strategies for llms. *arXiv preprint arXiv:2311.17371*, 2023. 1

R. G. Smith. *The contract net protocol: high-level communication and control in a distributed problem solver*, pp. 357–366. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1988. ISBN 093461363X. 15

Hongjin Su, Shizhe Diao, Ximing Lu, Mingjie Liu, Jiacheng Xu, Xin Dong, Yonggan Fu, Peter Belcak, Hanrong Ye, Hongxu Yin, et al. Toolorchestra: Elevating intelligence via efficient model and tool orchestration. *arXiv preprint arXiv:2511.21689*, 2025. 1

Yong Wu, YanZhao Zheng, TianZe Xu, ZhenTao Zhang, YuanQiang Yu, JiHuai Zhu, Chao Ma, BinBin Lin, BaoHua Dong, HangCheng Zhu, et al. Contextbudget: Budget-aware context management for long-horizon search agents. *arXiv preprint arXiv:2604.01664*, 2026. 15

Wujiang Xu, Zujie Liang, Kai Mei, Hang Gao, Juntao Tan, and Yongfeng Zhang. A-mem: Agentic memory for llm agents. *Advances in Neural Information Processing Systems*, 38:17577–17604, 2026. 15

John Yang, Carlos Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. Swe-agent: Agent-computer interfaces enable automated software engineering. *Advances in Neural Information Processing Systems*, 37:50528–50652, 2024. 5, 18

Andrew Chi-Chih Yao. Some complexity questions related to distributive computing(preliminary report). In *Proceedings of the Eleventh Annual ACM Symposium on Theory of Computing*, STOC '79, pp. 209–213, New York, NY, USA, 1979. Association for Computing Machinery. ISBN 9781450374385. doi: 10.1145/800135.804414. URL https://doi.org/10.1145/800135.804414. 15

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629*, 2022. 3, 5

Yuting Zeng, Weizhe Huang, Lei Jiang, Tongxuan Liu, Xitai Jin, Chen Tianying Tiana, Jing Li, and Xiaohua Xu. S2-mad: Breaking the token barrier to enhance multi-agent debate efficiency. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pp. 9393–9408, 2025. 15

Guibin Zhang, Yanwei Yue, Xiangguo Sun, Guancheng Wan, Miao Yu, Junfeng Fang, Kun Wang, Tianlong Chen, and Dawei Cheng. G-designer: Architecting multi-agent communication topologies via graph neural networks. *arXiv preprint arXiv:2410.11782*, 2024. 9

Yuzhe Zhang, Feiran Liu, Yi Shan, Xinyi Huang, Xin Yang, Yueqi Zhu, Xuxin Cheng, Cao Liu, Ke Zeng, Terry Jingchen Zhang, et al. Silo-bench: A scalable environment for evaluating distributed coordination in multi-agent llm systems. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 29379–29398, 2026. 1

Yujia Zheng, Zhuokai Zhao, Zijian Li, Yaqi Xie, Mingze Gao, Lizhu Zhang, and Kun Zhang. Thought communication in multiagent collaboration. *Advances in Neural Information Processing Systems*, 38:123389–123418, 2026. 1

Wanjun Zhong, Lianghong Guo, Qiqi Gao, He Ye, and Yanlin Wang. Memorybank: Enhancing large language models with long-term memory. In *Proceedings of the AAAI conference on artificial intelligence*, volume 38, pp. 19724–19731, 2024. 15

Mingchen Zhuge, Wenyi Wang, Louis Kirsch, Francesco Faccio, Dmitrii Khizbullin, and Jürgen Schmidhuber. Language agents as optimizable graphs. *arXiv preprint arXiv:2402.16823*, 2024. 9

APPENDIX

A Extended Related Work **15**

B Extended Details on Contradiction Audit **15**

B.1 Corpora 15

B.2 Splitting into snippets 15

B.3 Injecting contradictions 16

B.4 Difficulty levels 16

B.5 Tools and reporting 17

B.6 Scoring 17

C Implementation Details **17**

C.1 Hyperparameters 17

C.2 Agent harness 17

C.3 Evaluation details 18

C.4 Prompts 18

A Extended Related Work

Cooperative problem solving and coordination. How a group of problem solvers should divide work and share partial results is an old question, surveyed for the pre-LLM era in Durfee et al. (1989). The three structural answers this paper builds on were fixed in that period. The blackboard metaphor appears in Newell (1962), which already describes workers who each read everything on a shared surface, and was built in Erman et al. (1980), whose knowledge sources are activated by what appears on the blackboard rather than by calls from one another. Direct addressing was put to work for task allocation in Smith (1988), whose nodes exchange messages and share no memory, while Gelernter (1985) gives a shared medium in which writers do not name readers. The communication complexity of such structures was studied in parallel, from Yao (1979) to Dolev & Feder (1989), in terms of bits on a channel. The readers that literature assumes can scan a growing shared surface with little effort, so the amount of reading was never the binding constraint. It becomes one when the reader is a language model.

Reducing what agents read. A second line accepts the delivery structure and makes each delivery smaller. Zeng et al. (2025) propagates only exchanges it judges novel, and Chen et al. (2025a) trains agents to communicate more compactly; a parallel effort operates inside a single agent, compressing or evicting accumulated history to keep a long run within the context window (Wu et al., 2026; Semenov & Dorofeev, 2026),. Both reduce the size of what has already been routed to an agent. The bound studied here applies one step earlier, to how much is routed at all, and is set by the reader rather than by the sender. The two are complementary: every method in our experiments, ScaleMAS included, manages its own history in the same way (Appendix C.2).

Retrieval, memory, and information access. Shared stores for language agents are usually built to persist: Zhong et al. (2024) and Park et al. (2023) maintain per-agent stores across sessions, and Xu et al. (2026) links notes and rewrites older ones. The datastore studied here is not of this kind: it is created for one task and discarded with it, and exists so that agents working at the same time can reach each other's results. Retrieval by an agent has also been studied where the corpus is fixed in advance (Singh et al., 2025); here the corpus is the group's own output, so what an agent can find at any moment is what the others have published by that moment. The distinction our substrate rests on is older than either literature: Belkin & Croft (1992) separates systems in which profiles decide what reaches a reader from systems in which a reader expresses a need and initiates retrieval. The three families of Section 1 are of the first kind; the design of Section 2 is of the second, and takes its two access modes from the tradition of Bush et al. (1945) and Berners-Lee (1989), with ranking informed by reference structure as in Brin & Page (1998).

B Extended Details on Contradiction Audit

This section describes details about how Contradiction Audit (Section 3.1) is built and scored. The pipeline has four stages: collecting documentation, splitting it into snippets, injecting contradictions, and deriving difficulty levels.

B.1 Corpora

The benchmark covers sixteen open-source projects: CPython, Docker, Godot, JAX, Kubernetes, the Linux kernel, LLVM, NumPy, pandas, PyTorch, Ray, SciPy, scikit-learn, SQLAlchemy, SymPy, and Hugging Face Transformers. From the published documentation of each project we keep only Markdown and reStructuredText files. Files that disagree with the rest of the documentation by design are excluded, since they would contain contradictions we did not inject: release notes, changelogs and "what's new" pages; deprecation notices, roadmaps and to-do pages, which describe changing state; fragments that are transcluded verbatim into other pages; documentation templates; and contributor, governance and license pages. For a few projects we also exclude version-specific API dumps and migration guides. Table 3 lists the size of each corpus.

B.2 Splitting into snippets

Each file is split into paragraphs, that is, runs of non-blank lines, and the paragraphs are packed greedily into snippets of about 1024 tokens (`tiktoken o200k_base`). A paragraph is never split,

Table 3: The sixteen documentation corpora of Contradiction Audit, with the number of files kept after filtering and the number of snippets in the full corpus.

Project	Files	Snippets	Project	Files	Snippets
CPython	507	2,806	pandas	78	392
Docker	930	2,074	PyTorch	245	597
Godot	1,586	4,512	Ray	534	1,347
JAX	233	920	SciPy	339	635
Kubernetes	1,630	3,115	scikit-learn	102	475
Linux kernel	3,433	8,319	SQLAlchemy	152	583
LLVM	1,229	2,663	SymPy	317	732
NumPy	199	573	Transformers	726	1,507
Total: 12,240 files and 31,250 snippets.					

so every snippet ends at a paragraph boundary and may run slightly over the target to complete one; a file containing a single paragraph longer than twice the target is skipped. Snippets keep the order of their files, and each snippet begins with a line naming the file it comes from, which the agents see.

B.3 Injecting contradictions

Contradictions are injected into pairs of snippets by GPT-5.6, one pair at a time, in five steps.

Candidate pairs. Random pairs of snippets almost never discuss the same subject, so candidates are drawn from lexical overlap. A term is a lowercased word of at least four characters or one containing a digit; a pair is a candidate if the two snippets share at least two terms that each occur in at most eight snippets of the corpus. Candidates are visited in decreasing order of shared terms, with ties broken by a fixed random seed.

Shared subject. The model is shown the two snippets inside their full files, so that names and references can be resolved, and asked for up to three subjects that both snippets discuss and that are central to what their files document, rather than passing mentions. Pairs without such a subject are skipped.

Edit. The model rewrites the prose of one of the two snippets so that it can no longer be true together with the other, and explains why the two now conflict. It may only edit natural-language prose, never code, tables, list entries or headings, and must copy the rest of the snippet unchanged.

Mechanical checks. The edited snippet is compared with the original. The edit is rejected if it changes more than 30% of the snippet, if its changes are spread over more than eight separate places, or if it changes only formatting. Each sentence containing an edit must also occur exactly once in the corpus, so that the injected claim can be located without ambiguity.

Judges. Four independent GPT-5.6 judges then check that the edited snippet reads naturally on its own, that the contradiction is not trivial, so that a model cannot find it from memorized knowledge of the project, that the edited snippet does not contradict itself, and that the two snippets truly cannot both be true. A contradiction is kept only if all four judges accept it.

The walk over candidates stops once each corpus holds twenty contradictions: ten whose two snippets come from the same file and ten whose snippets come from different files. Every snippet takes part in at most one contradiction, so the twenty contradictions occupy forty distinct snippets.

B.4 Difficulty levels

Level $\mathrm{x}n$ keeps the forty contradiction snippets and adds $40(n-1)$ distractor snippets drawn at random with a fixed seed, for $40n$ snippets in total. Distractors are drawn first from snippets that form a whole file on their own, so that the reduced corpus consists of complete files as far as possible. The kept snippets stay in their original corpus order and are renumbered from zero. The contradictions are identical at every level, so a higher level differs only in how much unrelated text surrounds them.

The main experiments use `x4` (160 snippets per corpus) with Qwen3.5-9B and `x8` (320 snippets) with Claude-5-Sonnet.

B.5 Tools and reporting

Every method receives the same two corpus tools. `fetch_snippet` returns the full text of a snippet given its index. `search_snippet` is an exact, case-insensitive text search over the corpus, which returns up to ten matching snippets with the surrounding context of each match. Each agent is told the number of snippets and a starting position; the starting positions of a group are randomly set. A contradiction is reported as a pair of snippet indices with a short reason, and an agent's final answer is the list of pairs it is confident of. In ScaleMAS, a reported contradiction is also published as a message whose rationale quotes the conflicting text of both snippets; a quote that is not verbatim text of the cited snippets is rejected.

B.6 Scoring

A contradiction counts as found when some reported pair includes its edited snippet. Reported pairs that match no injected contradiction are not penalized. The score of a run is the fraction of the twenty contradictions found by the pairs its agents report, and the score of a method is this fraction averaged over the sixteen corpora.

C Implementation Details

C.1 Hyperparameters

Table 4 lists the settings used in all experiments unless stated otherwise. The citation weight follows Section 2.3: a new message starts at weight s_0, each message that cites it moves its weight a fraction α of the way toward one, and all weights decay by a factor γ at every step. Each agent may take a limited number of actions, where every tool call counts as one action, including a call that is rejected; a run also ends after a maximum number of steps, at which point every agent still running submits the best answer it can from the work already done.

Table 4: Settings of ScaleMAS in all experiments unless stated otherwise. The same temperature, generation limit, and random seeds are shared by every other baseline.

Setting	BrowseComp-Plus	Contradiction Audit	FATE	SWE-bench Verified
Messages returned per `search` (k)		8		
Messages per `visit` call		8		
Retrieval function, BM25 k_1 and b		BM25, 1.5 and 0.75		
Citation weight α, γ, s_0		0.3, 0.98, 0		
References per message (maximum)		32		
Tokens per message (sub-task and artifact)	256	256	1,024	1,024
Actions per agent (maximum)	300	250	100	300
Steps per run (maximum)	400	250	120	400
Context summarization	yes	yes	yes	no
Temperature		1.0		
Generation limit per call (tokens)		16,384		
Tool-call format		native tool calling		
Random seed		$0, 1, 2$		

C.2 Agent harness

ScaleMAS shares the identical agent harness with all baselines: ReAct, DeLM, AOrchestra, and MAD. For Contradiction Audit, FATE, and BrowseComp-Plus, the agent summarizes its own history periodically: once the history reaches ten turns, all but the three most recent turns are replaced

by a summary of at most 1,024 tokens written by the same model, and an earlier summary is rewritten together with the newly summarized turns. The mini-SWE-agent harness used on SWE-bench Verified keeps the full history instead.

C.3 Evaluation details

For the three third-party benchmarks, we report Pass@N accuracy (where N is the number of agents), following other multi-agent systems work (Park et al., 2026; Mao & Mirhoseini, 2026). This evaluation protocol is shared across all methods we tested. Pass@N measures what the group finds rather than which answer it would commit to; we treat answer selection as orthogonal, and single out the benefit of coordination by comparing with parallel ReAct agents.

Every method submits a final answer when its action limit is reached: ReAct and MAD through the forced final step of their agent loop, DeLM and AOrchestra by submitting the answer recorded so far. Each baseline follows its released design, adapted to the shared benchmark adapter.

BrowseComp-Plus. We use the questions and the document corpus of BrowseComp-Plus (Chen et al., 2025b), served through the search service and browser tools of OpenResearcher. Agents have three tools: `browser_search` returns the top 20 documents of a BM25 search over the corpus; `browser_open` opens a document or a link on the current page and shows a window of about 1,024 tokens, which the agent can move; and `browser_find` searches for a text pattern in an open page. Each agent's final answer is graded by an LLM judge (Qwen3.5-flash) with OpenResearcher's grading prompt, which extracts the final answer from the response and compares it with the gold answer. Experiments use the first 100 questions.

FATE. FATE (Jiang et al., 2026) poses problems in abstract algebra as formal statements in Lean 4 with Mathlib, at increasing levels of difficulty; we use FATE-M (150 problems) and FATE-H (100 problems). Agents have two tools: `search_mathlib` returns the 10 best-matching declarations from an index of Mathlib, and `check_lean` compiles a candidate proof and returns the compiler's messages, at most 10 times per agent with a limit of 900 seconds per compilation. A final proof is compiled with Lean 4 (v4.28.0) and counts as correct only if it proves the stated theorem, compiles without errors, and contains no `sorry`. Experiments use the first 100 questions of FATE-M, and use all questions of FATE-H.

SWE-bench Verified. We run SWE-bench Verified (Jimenez et al., 2024) on the mini-SWE-agent scaffold (Yang et al., 2024). Each agent works in its own container holding the repository at the base commit of the issue, runs shell commands with a limit of 60 seconds per command, and submits the diff of its changes as a patch. A patch is graded with the benchmark's tests: it is applied to a fresh copy of the repository, and the instance counts as resolved only if all tests that the fix must make pass (`FAIL_TO_PASS`) and all tests that must keep passing (`PASS_TO_PASS`) pass. Experiments use the first 100 instances of the test split.

C.4 Prompts

The part of the system prompt that describes the datastore and its operations is shared by every benchmark. In it, `publish`, `search`, and `visit` appear as the tools `publish_message`, `search_datastore`, and `visit_messages`:

```
You are working on a multi-step task across many sessions.

## The datastore: your record of the work

The datastore is a pile of messages holding everything already done on the input task.
Every message in it is an earlier work. You resume from them rather than start over.

- `publish_message(sub_task, artifact, rationale, refs)` is the ONLY way anything
  you do leaves a trace. Work you finish but never write down is lost.
- Every message records work ALREADY DONE, in three parts:
  - **sub_task** -- the piece of work you did, in the PRESENT PERFECT
    (e.g. "I've processed Resource8", "I've ruled out the second candidate"),
    stated in the words you would later search for to find it again;
  - **artifact** -- what the work produced: the result that can be picked up and
    built on without redoing it;
  - **rationale** -- BRIEFLY, how you got there, NAMING each of your refs and
    what it contributed (e.g. "Message12 gave the bound").
```

```
  Plus refs (pointers to other messages or to the resources your tools expose),
  and an id. Messages are never edited or deleted -- you only add new ones.
- Messages carry sequential names -- e.g. Message0, Message23 -- usable in refs.
- Any unit of work can be a message -- a digest of something you read, a verification of
  an earlier message, a consolidation of several, a piece of the deliverable. So
  can *raising* something: the `sub_task` is "Asked an open question", and the
  artifact is the question itself.
- MOST of what you write should be INTERMEDIATE work: a partial
  result, a lead worth taking, a dead end, something you ruled out, a step that
  turned out to be wrong, etc.
  Write these as messages AS YOU GO rather than saving up a finished answer.

## How to work

1. READ BEFORE YOU WRITE. BEFORE starting a piece of work, search for it -- the
   wording of the sub-task you are about to do, and the name of anything you
   would work on. If a message already declares that work, it is DONE: build on
   it and move to uncovered ground instead of repeating it.
   TWO READ PATHS:
   1) `search_datastore(<query>)` finds messages by their WORDING -- use it when you do not
     yet know which messages matter.
   2) `visit_messages(<message ids>)` fetches messages whose names
   you ALREADY know: from a message's refs, from a search hit that cited
   something, from anywhere an id was put in front of you.
2. GROUND EVERYTHING, AND SAY HOW. Refs declare what a message builds on -- e.g.
   "refs": ["Message23"] is a message building on message 23. A Message ref merges your
   work into an existing trail instead of starting a new one; a resource ref
   records what the message draws on so the source can be checked. Only reference names you
     have actually seen.
   IF YOU USED A MESSAGE, REFERENCE IT. If an earlier message's result saved you a step,
     sent you down
   a route, or you checked, corrected, extended, or contradicted it, put it in
   your refs. Two rules make a citation checkable, and BREAKING EITHER ONE REJECTS THE
   WHOLE MESSAGE -- it is not written, and you are told exactly which ref was at
   fault:
   - **IT MUST EXIST.** A ref names something you have ACTUALLY SEEN.
   - **YOUR RATIONALE MUST NAME IT, with what it contributed** -- e.g. "Message23 ruled
     out the direct route, so this takes the inductive one", "Input7 supplied
     the figure this checks".
3. WRITE TO BE FOUND. Your message is retrieved by its THREE FIELDS, nothing
   else, and messages containing a query's **exact wording**
   score far above messages merely sharing some of its words: use the exact,
   specific words (and names) that would be searched for later. This matters most
   in `sub_task`: it is what stops the same work being done twice. Keep
   `rationale` short -- it is the audit trail, not the payload.
```