MAScope: Diagnosing Collaboration Loss in Multi-Agent Systems

Anonymous authors
Paper under double-blind review

Abstract

Multi-agent systems assemble agents with different expertise, yet performance depends on how that expertise is organized, not on how much of it is present. Existing benchmarks only evaluate a method's outcome or a prescribed team's process, and cannot tell what a method's own collaboration contributes. We introduce `MAScope`, an environment of 51 expert agents holding private knowledge from Stack Exchange and 2,766 collaborative tasks with verifiable evidence and work dependencies. `MAScope` follows each required contribution from the agent that produces it into the final answer, splitting collaboration into four measurable capabilities: *expert selection*, *task orchestration*, *information transfer*, and *result integration*. Comprehensive experiments on twelve representative methods show that having the required experts does not deliver the task. Once work turns interdependent, success drops by 48%, over 3× that of a no-collaboration single agent. Trajectory analysis attributes this loss to three gaps: (1) **planning rigidity** in task orchestration, (2) **semantic compression** in information transfer, and (3) **adjudicative aggregation** in result integration. Even exhaustive collaboration and added memory recover little of it, since neither changes how a method organizes the work. `MAScope` gives a common basis for measuring and improving multi-agent collaboration, available at `https://anonymous.4open.science/r/MAScope-7B76`.

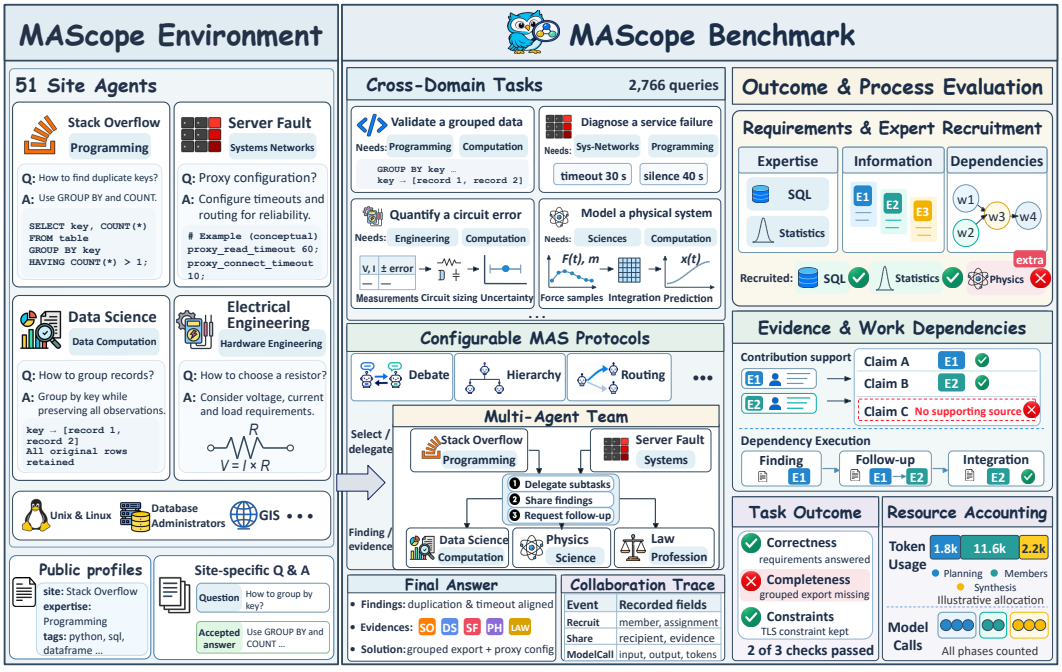


Figure 1: Overview of `MAScope`. The environment (**left**) provides 51 expert agents with public profiles and private Q&A corpora. The benchmark (**right**) lets MAS methods recruit agents and collaborate on 2,766 tasks, and verifies expert recruitment, evidence support, work dependencies, task outcome, and resource use.

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.

1 INTRODUCTION

Agent tasks are moving from closed-form question answering toward interactive work in real environments such as software development, data analysis, and tool use (Trivedi et al., 2024). These tasks require cross-domain knowledge, long-horizon state tracking, and adaptation to feedback (Yao et al., 2024; Ma et al., 2024). Multi-agent systems (MAS) meet these demands by distributing a task among agents with different expertise and solving it through communication and mutual verification, and are regarded as a promising organizational form for complex tasks (Li et al., 2023a).

However, gathering agents does not by itself create collaborative value. A larger team brings wider expertise, yet multiplies coordination overhead and opens more paths for errors to propagate (Kim et al., 2025). Working together, agents may misjudge their peers' intentions (Li et al., 2023b), leave commitments unfulfilled and features conflicting in coding tasks (Khatua et al., 2026), and omit context, repeat reasoning, or aggregate prematurely in information-seeking tasks (Cemri et al., 2025). The same agents under different organizational structures yield markedly different results (Zhuge et al., 2024; Qian et al., 2024). A team's value lies in the collaboration itself, and the central challenge is to *identify what kind of collaboration is valuable and where it breaks down*.

Existing domain benchmarks (Jimenez et al., 2023; Mialon et al., 2023; Phan et al., 2026) evaluate a method only through its final outcome (Kapoor et al., 2025), leaving an internal failure difficult to interpret. Recent multi-agent benchmarks measure the collaboration process through communication structure, coordination behavior, and framework-level capabilities (Zhu et al., 2025; Grötschla et al., 2025; Zhang et al., 2026; Orogat et al., 2026), and each examines one facet of a team it prescribes. Under both designs, a required contribution goes untraced through a method's own workflow, from where it is demanded to where the outcome adopts it. Our preliminary study (§ 2) shows that this blind spot is harmful: **(1)** a MAS answers correctly without a required agent's evidence, and **(2)** correct findings are discarded before reaching the answer. Attributing a MAS outcome to its collaboration therefore requires end-to-end evaluation across the whole workflow.

We introduce `MAScope` (§ 3) to make that evaluation possible. Our contributions are threefold:

- **An expert environment with disjoint private knowledge.** We build one expert agent per Stack Exchange community (Stack Exchange, Inc., 2014), 51 in all, each holding a private corpus no other agent can reach. Every passage returns with its source identifier, so a contribution stays traceable from the agent that retrieves it to the answer that carries it.
- **Collaborative tasks with certified work dependencies.** `MAScope` has 2,766 queries in 499 families, with 11,018 expertise units across three dependency structures (PARALLEL, SEQUENTIAL, and expert DISCOVERY) and three expertise scales. We design two deterministic rules to certify each dependency, making its predecessor indispensable to any accepted answer.
- **End-to-end evaluation of the collaboration workflow.** We decompose the workflow into four measurable capabilities, *expert selection*, *task orchestration*, *information transfer*, and *result integration*, and decide each from the logged trajectory against a frozen reference, locating every failure at the step where it occurs in each method's own workflow.

With `MAScope`, we evaluate twelve representative MAS methods across the three coordination classes (§ 4) and find that having the required experts does not deliver the task. The methods average only 41.3% success, and four of them fall below a no-collaboration single agent holding the same corpora. Once work turns interdependent, success drops by 48%, over 3× that of the no-collaboration agent. Trajectory analysis traces this loss to three gaps between agents: ① **planning rigidity** in task orchestration, ② **semantic compression** in information transfer, and ③ **adjudicative aggregation** in result integration. Even exhaustive collaboration and added memory recover little of it, since neither changes how a MAS organizes the work. `MAScope` thus locates the bottleneck in how work is organized and makes each gap measurable and comparable across methods.

2 MOTIVATIONS AND OBSERVATIONS

We first examine whether the final outcome of a MAS reflects the collaboration behavior behind it. We run three representative methods, AutoGen (Wu et al., 2024), CrewAI (Moura, 2023), and Anemoi (Ren et al., 2025), on multi-hop question answering (MuSiQue, Trivedi et al., 2022) and

general assistant tasks (GAIA, Mialon et al., 2023), with a shared backbone, four agents, and a budget of 150k tokens per task, averaged over five runs. For every run we freeze all agent records before aggregation and re-aggregate twice under the same rule, **(I)** from the complete record and **(II)** after masking the evidence contributed by one required agent (Appendix E). The same trajectories yield two observations about the MAS collaboration process:

Observation 1: A MAS answers correctly without a required agent's evidence. Masking one required agent lowers accuracy by only 12.5% on MuSiQue but by 48.7% on GAIA (Figure 2, left). The gap arises because MuSiQue evidence is textual and the aggregating agent can reason it out from the remaining context and its own knowledge, whereas GAIA evidence comes from file processing and web retrieval and cannot be reproduced by reasoning alone. Such compensation lets a correct answer hide a missing contribution, by an amount that shifts severalfold with the environment.

Observation 2: Correct findings are discarded before reaching the answer. Among failed runs, 29.8% on MuSiQue and 45.5% on GAIA already contain the decisive finding, produced by some agent but never carried into the answer (Figure 2, right). The failure lies downstream of the finding, in the handoff and the aggregation, where a stronger agent has nothing to add, so a wrong answer can hide a contribution the team had already made.

Together the two observations show that evaluation on the outcome cannot separate obtaining a contribution from using it. An evaluation of collaboration thus has to make every required contribution indispensable, ruling out compensation, and traceable from production to adoption, exposing where it is lost.

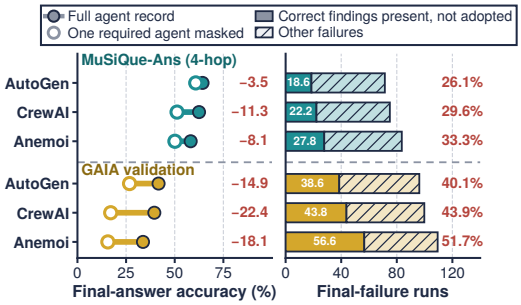


Figure 2: MAS collaboration behavior on MuSiQue and GAIA. We report final-answer accuracy with the full agent record and with one required agent masked (**left**), and failed runs split by whether a correct finding was already present (**right**).

3 MASCOPE: TRACING CONTRIBUTIONS TO THEIR ADOPTION

To evaluate multi-agent collaboration end to end, we develop `MAScope`, an environment where every required contribution is indispensable and traceable from the agent that produces it to the answer that adopts it. First, we build an expert environment from 51 Stack Exchange communities and compose collaborative tasks from it (§ 3.1). Second, we organize these tasks along two independent axes, collaboration structure and required expertise (§ 3.2). Third, we design an evaluation framework that follows each contribution through the workflow and measures four capabilities (§ 3.3).

3.1 BENCHMARK CONSTRUCTION

Expert environment. We instantiate one expert agent per Stack Exchange community, 51 in total. Each agent holds a public profile and a private corpus. MAS methods see only the profile, which lists the site name, expertise scope, and topic tags, when deciding whom to recruit. Execution draws on the corpus, which holds the community's questions, accepted answers, and source identifiers for retrieval and citation. Disjoint corpora prevent one agent from supplying evidence that belongs to another, while identifiers minted inside the benchmark keep a memorized copy of Stack Exchange from passing as evidence, so a task with one required corpus hidden is all but unsolvable (Appendix A.3). All agents share the same backbone and differ only in their corpora, so task completion depends on how their information is organized.

Task definition. A task is a tuple $T = (q, U, G, V)$. The method sees only q, a user's goal and observations, naming no site. Behind it, U enumerates the information units a solution requires, each bound to a source identifier, G records which units can be scoped only once a predecessor finding exists, and V states the acceptance criteria. Tasks admit multiple valid paths, with equivalent evidence and alternative routes recorded in the reference, so evaluation fixes no order of agent calls.

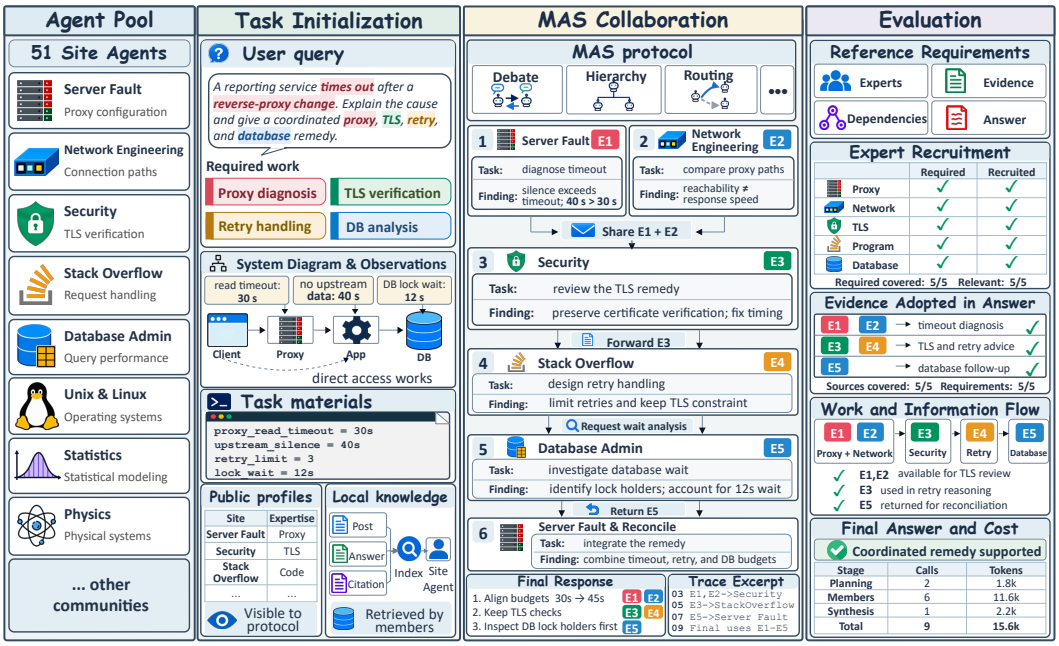


Figure 3: A MAScope run on a service-diagnosis task. The task needs five information units (E1–E5), the last three of which can be scoped only once the diagnosis in the first two is in hand. The evaluator checks recruited experts, whether each dependency was carried forward with its constraints, which evidence reached the answer, and the tokens each stage consumed.

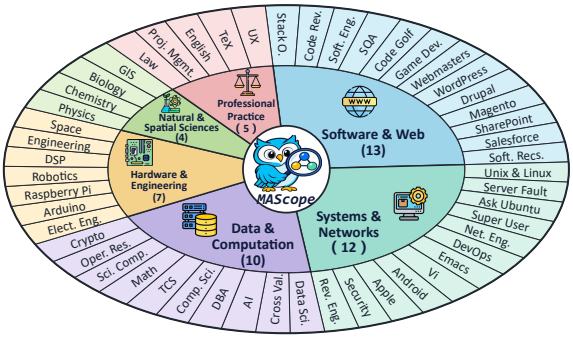
Construction pipeline. On top of this environment, each task is built in four steps.

- **Evidence alignment.** From real community questions and answers, we select units that belong to different sites yet serve one goal, verify each against its corpus, and bind it to a source identifier.
- **Task composition.** The demands of these units are rewritten into one user-facing question that gives the goal and the observed symptoms, leaving the required expertise to be inferred.
- **Dependency certification.** A unit is dependent when its inputs or objectives can be fixed only by a predecessor finding. We introduce two deterministic rules certifying an edge when the predecessor alone states a term the unit must carry and the unit's passage is reachable only with that finding, and mark expert discovery when the finding reveals the expertise needed next.
- **Validation and freezing.** To confirm that a task is well posed, we solve it end to end, checking that every corpus is reachable and the expertise no finding hides is inferable from q. Units whose evidence cannot be recovered from their own corpus are revised or dropped, tasks are frozen by family, and only q, the profiles, and the environment ship.

3.2 Benchmark Overview

Dataset statistics. MAScope contains 2,766 queries from 499 task families with 11,018 required expertise units. The 51 agents span six resource groups, from software engineering to professional practice, and a single task usually draws on more than one group, producing integrated goals in system diagnosis, data processing, engineering analysis, and professional judgment.

Task taxonomy. We characterize each task by its collaboration structure C and required expertise S. PARALLEL composition (C1) splits a task into independent pieces combined at the end, such as a transformation rule and a check for its result. SEQUENTIAL dependence (C2) chains them, so a later piece takes an earlier finding as input, as when a configuration change follows from where a timeout originates. Clue-based expert DISCOVERY (C3) hides the next expert's identity inside a predecessor finding, which the method must recognize. S counts the required experts, S1 (2–3), S2 (4–5), and S3 (6+), without capping how many agents a method calls. The 1,630 queries of C2 and C3 carry certified dependency graphs, 3,664 edges in all. As C and S vary independently, MAScope separates the effect of deeper dependency from that of broader expertise on the same method.



Task structure C	Required Expertise S1 (2-3)	S2 (4-5)	S3 (6+)	Total
C1 Parallel	454	390	292	**1,136**
C2 Sequential	394	314	236	**944**
C3 Discovery	296	236	154	**686**
Total	**1,144**	**940**	**682**	**2,766**

Figure 4: Expert pool and task taxonomy. `MAScope` holds 51 expert agents in six resource groups (**left**), and spreads its 2,766 queries over collaboration structure crossed with required expertise (**right**).

3.3 Evaluation Framework

Autonomous execution and outcome verification. A method receives q, the public profiles, a shared agent pool, tools, and a budget, and then recruits agents, assigns work, exchanges information, and submits a final answer entirely under its own mechanism, while the environment logs every agent call, message, local output, and token. The evaluator verifies the submitted answer against V to obtain task success, maps the evidence cited in the answer onto U to obtain evidence coverage, which records how much of the required content reached the answer even when the task fails, and sums tokens across planning, execution, aggregation, and memory operations to obtain execution cost, so all three outcome metrics are read from the same logged run.

Tracing contributions along the dependency chain. To locate where collaboration breaks down, we follow each required contribution along the chain that carries it into the answer. For every C2 and C3 query we fix in advance the certified dependency graph G, whose edges carry terms an accepted answer takes only from the predecessor, and a method is ready to act on an edge once it holds the predecessor finding and has recruited the downstream expert ($s^{(1)}$). Along it we measure four capabilities. **(i)** *Expert selection* covers the required expertise, measured over all queries by how completely and precisely the recruited agents A_q match the required experts R_q. **(ii)** *Task orchestration* turns the available finding into work assigned to that expert ($s^{(2)}$), and **(iii)** *information transfer* brings the expert the predecessor's evidence and key constraints intact, through the assignment or shared state it reads ($s^{(3)}$). The expert then produces a correct local result ($s^{(4)}$), recorded as local solve to separate an agent failing to solve from a team failing to adopt, and **(iv)** *result integration* carries that result into the answer ($s^{(5)}$). Each capability is decided per edge, charging an upstream loss where it occurs. With $s_e^{(k)} \in \{0, 1\}$ for edge e reaching stage k given stage $k-1$, the rates are

$$r_k = \frac{\sum_e s_e^{(k)}}{\sum_e s_e^{(k-1)}}, \qquad k = 2, \ldots, 5.$$

Each stage is decided by matching logged identifiers and preregistered terms, alike across classes.

4 Experiments

4.1 Experimental Setup

Compared methods. We evaluate twelve MAS methods grouped by how centralized their coordination is. (1) *Centralized* methods route every decision through one coordinator: AutoGen (Wu et al., 2024), LangGraph (Barda, 2025), CrewAI (Moura, 2023), Magentic-One (Fourney et al., 2024), and Flow (Niu et al., 2025). (2) *Semi-centralized* methods keep a planning, aggregation, or evaluation step while letting agents answer in parallel or talk directly: Anemoi (Ren et al., 2025), AgentVerse (Chen et al., 2023), MoA (Wang et al., 2024), and MAD (Du et al., 2024). (3) *Decentralized* methods have no fixed coordinator and let agents propagate information or route work themselves: GoA (Yun et al., 2026), AgentNet (Yang et al., 2025), and SelfOrg (Tastan et al., 2025). Each runs its native mechanism, deciding whom to recruit, how to organize the work, and when to stop. As a no-collaboration reference we add a single agent that searches all 51 corpora itself.

Table 1: Main results on MAScope. We report task success (%) crossing C1 parallel, C2 sequential, and C3 expert discovery with expertise S1 to S3. Background color marks the level of degradation from the C1 cell at the same scale, and Δ_{dep} and Δ_{scale} give the relative change in success from C1 to C2∪C3 (dependency) and from S1 to S3 (expertise scale). ± is one standard deviation over five runs.

Method	C1 Parallel			C2 Sequential			C3 Discovery			Overall			Cost	
	S1	S2	S3	S1	S2	S3	S1	S2	S3	Success ↑	Δ_{dep}	Δ_{scale}	Tok (k) ↓	Calls ↓
Centralized														
AutoGen	74.3	65.7	53.9	**57.6**	**44.0**	28.1	**49.6**	**34.4**	20.6	**52.1**±0.66	**−35.9**	−39.7	171	14.6
Magentic-One	72.5	64.9	52.7	53.4	41.0	24.7	48.4	33.5	**21.0**	50.2±0.55	−38.3	−39.9	208	23.2
Flow	75.0	65.8	53.3	45.5	34.0	20.5	38.7	26.1	16.5	46.6±0.39	−50.4	−39.4	126	14.4
CrewAI	**77.2**	**68.9**	55.3	43.2	27.0	15.8	33.6	21.1	12.1	44.9±0.51	−58.9	−41.1	119	11.8
LangGraph	64.8	54.2	41.2	47.9	33.5	22.6	36.1	27.4	15.8	42.3±0.30	−39.5	−43.6	127	16.7
Avg.	*72.8*	*63.9*	*51.3*	*49.5*	*35.9*	*22.4*	*41.3*	*28.5*	*17.2*	*47.2*±0.21	*−44.6*	*−40.8*	*150*	*16.1*
Semi-centralized														
Anemoi	64.3	57.4	54.2	42.6	33.4	25.3	39.3	31.3	17.8	44.3±0.70	−43.1	−28.5	118	15.5
AgentVerse	67.6	60.2	48.8	39.8	27.3	16.4	37.7	25.3	15.1	41.9±0.79	−51.5	−40.4	165	19.9
MoA	61.2	55.1	44.9	21.2	12.4	6.6	25.6	13.6	9.2	32.0±0.19	−70.9	−38.2	1,044	164.3
MAD	52.2	42.6	30.9	25.9	18.7	11.2	26.9	17.7	8.2	29.4±1.00	−54.6	−48.2	961	160.2
Avg.	*61.3*	*53.8*	*44.7*	*32.4*	*23.0*	*14.9*	*32.4*	*22.0*	*12.6*	*36.9*±0.24	*−55.0*	*−38.8*	*572*	*90.0*
Decentralized														
GoA	65.6	62.9	**60.5**	43.3	36.3	**29.4**	36.4	27.5	19.7	46.2±0.78	−46.1	**−19.5**	208	26.8
AgentNet	56.7	43.5	29.2	30.8	20.8	13.2	39.9	27.3	13.6	33.7±0.51	−42.7	−53.5	**88**	**7.9**
SelfOrg	50.5	43.5	42.7	29.4	19.4	19.2	27.0	20.5	11.8	32.3±0.49	−51.0	−25.8	324	175.8
Avg.	*57.6*	*50.0*	*44.1*	*34.5*	*25.5*	*20.6*	*34.4*	*25.1*	*15.1*	*37.4*±0.34	*−46.6*	*−33.0*	*206*	*70.2*
Single Agent	54.7	43.2	32.3	48.7	38.2	26.2	45.7	33.3	22.5	41.0±0.46	−15.1	−44.4	131	11.9

Implementation details. Every method and agent uses `claude-sonnet-5` (Anthropic, 2026) as its backbone and shares the same expert agents, profiles, and retrieval interface, and every query ends with the same line asking the final answer to cite the identifier of each passage it rests on. All experiments run five times over the 2,766 queries. Details are in Appendix D.

4.2 Main Results

① **A MAS does not reliably outperform a single agent.** Eight MAS methods clear the no-collaboration single agent's 41.0%, but by margins from 11.1 points for AutoGen to only 0.9 for AgentVerse, and the other four fall below it, as low as 29.4%, among them MoA, MAD, and SelfOrg with all 51 experts at work. Recruiting more experts buys no advantage by itself, and with corpora, budget, and backbone held fixed, what separates a MAS from a single agent working alone is how it organizes the work after dividing it, which § 4.3 traces stage by stage.

② **Work dependencies systematically degrade MAS performance.** When a MAS takes on tasks whose later work builds on earlier findings, average success falls from 57.8% to 29.8%. Every coordination class loses at least 44% of its success, so no organizational style is immune. Scaling the required expertise from 2–3 to 6+ experts cuts success by a further 38%. On the dependent tasks the single agent, which keeps every finding in the context that uses it, loses only 15%, and ten of the twelve methods fall below it. Yet the experts hold up, since once one receives the finding it needs, its local solve stays near 90% for every method (Table 2). The loss thus falls on the step that turns one agent's finding into the next agent's work.

③ **Higher cost does not translate into better performance.** Per-task cost spans an order of magnitude, from 88k tokens for AgentNet to 1,044k for MoA, tracking how many agents answer, not how well. Flow reaches 46.6% at 126k tokens and MoA 32.0% at 8× the cost. The performance–cost frontier is held mostly by the centralized class, where AutoGen leads at 1.3× the single agent's cost. Extra computation buys participation and communication and leaves the missing piece of work undone, as MoA, despite 1,044k tokens, assigns only 34.8% of the work its findings imply.

Table 2: Collaboration capabilities along the dependency chain (%). We report expert selection over all queries and the other capabilities as conditional rates over the 3,664 certified dependencies, each scored on its own. Background color marks how far a method trails its column's leader.

Method	Expert Selection		Collaboration Pipeline				Overall	
	Recall	Precision	Task Orchestration →	Information Transfer →	Local Solve →	Result Integration	Dep. Survival	Evidence Coverage
Centralized								
AutoGen	81.8	74.0	85.5	**89.8**	91.6	92.3	**64.9**±1.05	67.4
Magentic-One	80.4	71.6	86.5	83.5	91.3	92.2	60.8±0.67	65.7
Flow	83.1	71.9	83.2	85.4	90.4	90.1	57.9±0.74	63.6
CrewAI	81.9	73.7	70.9	81.8	89.4	91.9	47.6±0.88	61.8
LangGraph	77.7	68.6	77.3	85.3	89.7	**94.4**	55.8±0.79	58.1
Avg.	*81.0*	*72.0*	*80.7*	*85.2*	*90.5*	*92.2*	*57.4*±0.53	*63.3*
Semi-centralized								
Anemoi	78.9	69.4	**92.0**	80.5	**93.1**	76.4	52.6±1.15	58.5
AgentVerse	86.6	66.7	76.0	77.0	90.6	79.8	42.3±0.61	58.7
MoA	**100.0**	7.8	34.8	77.7	90.4	80.0	19.5±0.25	**72.1**
MAD	**100.0**	7.8	76.2	83.3	87.6	73.9	41.1±0.87	44.0
Avg.	*91.4*	*37.9*	*69.8*	*79.6*	*90.4*	*77.5*	*38.9*±0.19	*58.3*
Decentralized								
GoA	84.1	69.6	91.7	71.7	88.4	87.3	50.8±0.49	62.6
AgentNet	68.1	**87.0**	87.6	79.3	90.5	85.9	54.0±0.58	49.6
SelfOrg	**100.0**	7.8	85.5	65.6	86.9	81.2	39.6±0.95	49.2
Avg.	*84.1*	*54.8*	*88.3*	*72.2*	*88.6*	*84.8*	*48.1*±0.29	*53.8*

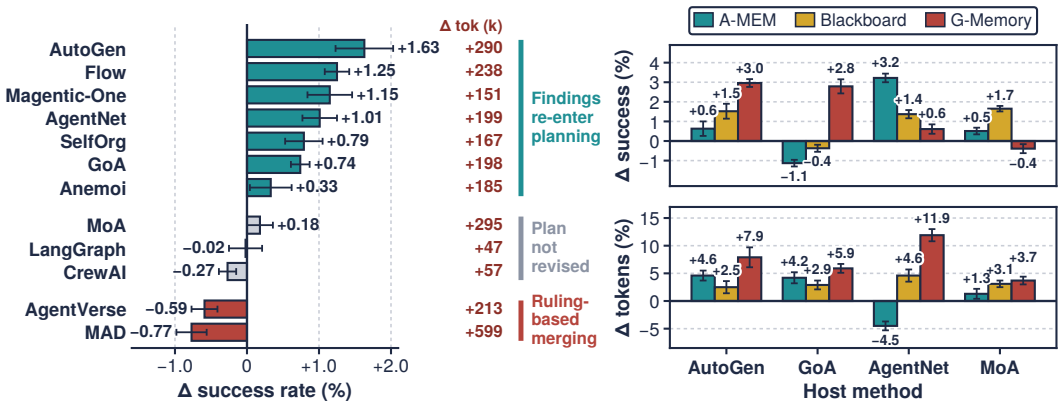
4.3 Why Does Collaboration Fail?

Tracing every certified dependency through the four capabilities of § 3.3 shows that agents complete the work they receive reliably, so the loss lies in three capability gaps between them (Table 2).

① **Agent findings struggle to revise an established plan.** Even with the finding in hand and the expert recruited, 21% of the dependencies never have the work they imply assigned, and we call this **planning rigidity** in task orchestration. Centralized methods that never revise a plan leave a new finding nowhere to go. CrewAI executes a task list fixed at kickoff and assigns 70.9% of the implied work, and LangGraph, whose supervisor keeps no task list, stops once a worker's message reads as complete and assigns 77.3%. Methods with a replanning trigger recover part of it, as Magentic-One and Flow revise on a stall or a failed step, at 86.5% and 83.2%. Orchestration holds up where the agent holding a finding passes the follow-up on itself, with Anemoi, GoA, and AgentNet between 87.6% and 92.0%. MoA is an extreme case at 34.8% despite recruiting all 51 experts, as its layers pass merged answers upward and assign follow-up only where a merge states it.

② **Agents tend to transfer conclusions without supporting evidence.** Of the work that is assigned, 20% arrives as a bare conclusion stripped of its evidence and constraints, and we call this **semantic compression** in information transfer. Decentralized methods shorten a finding or send it past the agent that needs it, as GoA's agents answer in the two or three sentences its prompt asks for and SelfOrg routes whole responses by their similarity to the receiver's own answer, and they deliver usable inputs in 71.7% and 65.6% of cases. AgentVerse, which compresses an assignment into one subtask sentence, reaches 77.0%. Transfer holds up wherever the predecessor's output passes on intact, whether a coordinator forwards the full conversation, as in AutoGen at 89.8%, or a graph edge carries the output, as in Flow at 85.4%.

③ **Correct local results are overridden during aggregation.** Once a correct local result exists, 14.5% of the time it never reaches the answer, and we call this **adjudicative aggregation** in result integration. Semi-centralized methods merge outputs in one step, a majority vote in MAD, a judge in AgentVerse, the planner in Anemoi, and the final synthesis in MoA, and each such step favors



(a) Gain under exhaustive collaboration (b) Memory modules on four host methods

Figure 5: Exhaustive collaboration and memory modules. We report the change in success **(a)** when a method must collaborate until it confirms completion, grouped by mechanism and annotated with the extra tokens spent in red, and **(b)** when three memory modules are added to four host methods.

what most outputs share, dropping a correct result held by one agent, so MAD retains only 73.9% and Anemoi 76.4%. Centralized methods assemble outputs into one answer instead, all above 90%.

Overall, each MAS class carries its own strength and its own blind spot. In centralized methods a coordinator holds the full context and assembles the answer, which secures transfer and integration but routes every new assignment through that one role. Decentralized methods remove that coordinator so any agent can pick up new work, yet the hop-by-hop relay that makes this possible thins the evidence along the way. Between the two, semi-centralized methods buy parallel progress with an arbitration step that lets agreement overwrite a correct result held by one agent.

4.4 Can Exhaustive Collaboration Recover Performance?

To test whether more collaboration closes these gaps, we force each method to keep going under its own mechanism until it confirms that every requirement is met and no agent can contribute further.

① **Exhaustive collaboration recovers little of the loss.** Collaborating to exhaustion improves success by at most 1.6%, lowers it beyond its run-to-run spread for three methods, and leaves three inside it, while costing 1.3× to 3.3× the native run. Set against the 28-point gap between parallel and dependent tasks, even the best gain recovers under 6% of it. The extra rounds re-run the same plan, regenerate the same answer, or adjudicate again, so none adds work the first pass left out.

② **Only methods that turn new findings into new work benefit.** Methods that let findings re-enter planning spend each extra round revisiting what has been found and dispatching what follows from it, and all of them gain, up to 1.63% for AutoGen. Anemoi gains least, 0.33% and inside its spread, because its planner's ruling discards the new findings again. Methods that never revise a plan (CrewAI, LangGraph, MoA) stay between -0.27% and $+0.18\%$, since another round repeats the same work and any gain is offset by fresh errors. The two methods whose ruling cannot act on a finding, AgentVerse's judge and MAD's vote, decline by 0.59% and 0.77%, as each round is another chance to overturn a correct result. Extra collaboration thus pays off only where a method routes findings into new work, and leaves the orchestration and integration gaps in place.

4.5 Combining Memory Modules with MAS Methods

Memory supplies the experience a MAS can draw on when selecting experts, making later decisions, and passing information between agents, which makes it another common enhancement. We adopt three representative modules into `MAScope`. A-MEM (Xu et al., 2025) keeps cross-task experience as structured notes on each agent, Blackboard (Han & Zhang, 2025) holds a shared working state that the team reads and writes within a task, and G-Memory (Zhang et al., 2025a) retrieves across tasks and steps through a three-level graph of experience, tasks, and interactions. We pair each module with AutoGen, GoA, AgentNet, and MoA, which span the three coordination classes.

① **Memory gains depend on the host.** Each module helps a host whose weak stage it covers, so no module helps every host. AgentNet routes work through each agent's local decisions, the level where A-MEM keeps its notes, and the match gains most, 3.2%. G-Memory's cross-step retrieval restores the links AutoGen and GoA fail to carry, giving 3.0% and 2.8%, and Blackboard's shared state lets MoA agents see each other's work, worth 1.7%. A mismatched module adds context to every call. Cross-step retrieval leaves MoA, which does no cross-step work, inside its spread, and notes and shared state keep GoA's findings as short as they were written, costing it 1.1% and 0.4%.

② **Memory gains are usually paid for in tokens.** Retrieval and writing add 1% to 12% of tokens in most pairings, and the cost does not follow the gain, as with G-Memory on AgentNet at 11.9% more tokens for 0.6% more success. AgentNet with A-MEM is the one pairing that pays for itself, cutting tokens by 4.5% while raising success by 3.2%, since knowing what each agent does well removes routing attempts that lead nowhere. Experience that enters a decision saves the trial and error it replaces, and experience that does not leaves only the cost of retrieval.

5 Related Work

MAS methods. MAS methods differ mainly in who decides how work is organized. *Centralized* methods give that decision to a coordinator, through fixed roles and standard procedures, speaker selection, shared-state routing, or dispatch over subtask graphs (Hong et al., 2023; Wu et al., 2024; Fourney et al., 2024; Niu et al., 2025; Barda, 2025). *Semi-centralized* methods keep a planning, aggregation, or evaluation step while agents answer in parallel or talk directly (Du et al., 2024; Liang et al., 2024; Wang et al., 2024; Ren et al., 2025). *Decentralized* methods drop the coordinator and let agents propagate information or route work themselves (Zhuge et al., 2024; Qian et al., 2024; Yun et al., 2026; Yang et al., 2025), and search-based methods learn the organization (Zhang et al., 2024; 2025b). The three classes thus give planning, relay, and merging to different roles, and `MAScope` evaluates twelve representative methods from all three on the same experts and tasks.

Evaluation of multi-agent collaboration. General agent benchmarks score a method by its final outcome (Mialon et al., 2023; Jimenez et al., 2023; Wu et al., 2025). Multi-agent benchmarks mostly study collaboration in a team they prescribe, some making it necessary through neighbor-only topologies (Grötschla et al., 2025), sharded inputs (Zhang et al., 2026), or enforced role separation (Kim et al., 2026), and others scoring milestones and coordination (Zhu et al., 2025; Khatua et al., 2026). Recent work turns to per-run trajectory analysis, measuring failure modes (Cemri et al., 2025), scaling conditions (Kim et al., 2025), and agent credit (Jiang et al., 2026), or to framework-level evaluation, comparing published frameworks by outcome on single-capability tasks (Orogat et al., 2026). `MAScope` combines necessary collaboration, a team each method forms, and a decision at every workflow step, so a required contribution is traced from where it is demanded to where the outcome adopts it.

Table 3: Comparison with multi-agent benchmarks. *Team* is fixed before the run (Preset) or formed by the method during it (Online), and *Granularity* is the unit an evaluation decides on.

Benchmark	Necessary	Team	Granularity
General agent	✗	–	Outcome
`MultiAgentBench`	✗	Preset	Per-run
`AgentsNet`	✓	Preset	Outcome
`SILO-BENCH`	✓	Preset	Per-run
`TeamBench`	✓	Preset	Per-run
`CooperBench`	✗	Preset	Per-run
`MAFBench`	✗	Preset	Outcome
`MAScope`	✓	**Online**	**Per-step**

6 Conclusion

We introduce `MAScope`, an environment of 51 expert agents with private knowledge and 2,766 tasks with certified dependencies, which traces every required contribution into the answer. Experiments on twelve representative MAS methods show that having the required experts does not deliver the task. Success nearly halves once work turns interdependent, over 3× that of a no-collaboration agent, and three gaps account for the loss: planning rigidity in task orchestration, semantic compression in information transfer, and adjudicative aggregation in result integration. Even exhaustive collaboration and added memory recover little, as neither changes how a method organizes work. `MAScope` measures each gap where it arises, so improvements can be tested where they act.

Reproducibility Statement

Acceptance in MAScope is settled by string against a frozen reference, so the same answer always receives the same verdict, and every rate follows from the verdicts. Every step behind the reported numbers is specified in the appendix. Appendix A gives the 51 communities, the filtering that yields each corpus, and the two surfaces an agent works through. Appendix B gives the construction pipeline, the deterministic rules that certify a dependency, and the checks a task clears before it is frozen. Appendix C gives the evaluation protocol, from the event stream an adapter produces to the five predicates that decide each certified dependency. Appendix D gives the twelve methods with their configurations, the no-collaboration reference and the two enhancement settings, and Appendix H prints the answer format and two of the three prompts MAScope adds. Appendix F reports the counts behind every rate together with its run-to-run spread, and Appendix I states what the release ships and under which licenses.

AI Use Statement

We used generative AI tools as a writing and presentation aid, to edit and polish prose written by the authors and to prepare the scripts that render our measured results into figures and tables. The authors reviewed all AI-assisted output and take responsibility for this paper.

References

Anthropic. Introducing Claude Sonnet 5. Anthropic, June 2026. URL https://www.anthropic.com/news/claude-sonnet-5. Accessed: 2026-09-16.

Vadym Barda. LangGraph Multi-Agent Supervisor. Software, 2025. URL https://github.com/langchain-ai/langgraph-supervisor-py. Accessed: 2026-09-16.

Mert Cemri, Melissa Z. Pan, Shuyi Yang, Lakshya A. Agrawal, Bhavya Chopra, Rishabh Tiwari, Kurt Keutzer, Aditya Parameswaran, Dan Klein, Kannan Ramchandran, Matei Zaharia, Joseph E. Gonzalez, and Ion Stoica. Why Do Multi-Agent LLM Systems Fail? *ArXiv*, abs/2503.13657, 2025. URL https://api.semanticscholar.org/CorpusID:277103715.

Weize Chen, Yusheng Su, Jingwei Zuo, Cheng Yang, Chenfei Yuan, Chi-Min Chan, Heyang Yu, Yaxi Lu, Yi-Hsin Hung, Chen Qian, Yujia Qin, Xin Cong, Ruobing Xie, Zhiyuan Liu, Maosong Sun, and Jie Zhou. AgentVerse: Facilitating Multi-Agent Collaboration and Exploring Emergent Behaviors in Agents. *ArXiv*, abs/2308.10848, 2023. URL https://api.semanticscholar.org/CorpusID:261048935.

Yilun Du, Shuang Li, Antonio Torralba, Joshua B. Tenenbaum, and Igor Mordatch. Improving Factuality and Reasoning in Language Models through Multiagent Debate. In *International Conference on Machine Learning*, 2024. URL https://api.semanticscholar.org/CorpusID:258841118.

Adam Fourney, Gagan Bansal, Hussein Mozannar, Cheng Tan, Eduardo Salinas, Erkang (Eric) Zhu, Friederike Niedtner, Grace Proebsting, Griffin Bassman, Jack Gerrits, Jacob Alber, Peter Chang, Ricky Loynd, Robert West, Victor Dibia, Ahmed Awadallah, Ece Kamar, Rafah Hosn, and Saleema Amershi. Magentic-One: A Generalist Multi-Agent System for Solving Complex Tasks. *ArXiv*, abs/2411.04468, 2024. URL https://api.semanticscholar.org/CorpusID:273877854.

Florian Grötschla, Luis Müller, Jan Tönshoff, Mikhail Galkin, and Bryan Perozzi. AgentsNet: Coordination and Collaborative Reasoning in Multi-Agent LLMs. *ArXiv*, abs/2507.08616, 2025. URL https://api.semanticscholar.org/CorpusID:280298162.

Bochen Han and Songmao Zhang. Exploring Advanced LLM Multi-Agent Systems Based on Blackboard Architecture. *ArXiv*, abs/2507.01701, 2025. URL https://api.semanticscholar.org/CorpusID:280139438.

Sirui Hong, Mingchen Zhuge, Jiaqi Chen, Xiawu Zheng, Yuheng Cheng, Ceyao Zhang, Jinlin Wang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, Liyang Zhou, Chenyu Ran, Lingfeng Xiao, Chenglin Wu, and Jürgen Schmidhuber. MetaGPT: Meta Programming for Multi-Agent Collaborative Framework. *ArXiv*, abs/2308.00352, 2023. URL https://api.semanticscholar.org/CorpusID:260351380.

Pengyi Jiang, Xiaoguang Zhu, and Quanyan Zhu. Semantic Cooperative Games for Contribution Attribution in LLM-Based Multi-Agent Systems. *ArXiv*, abs/2607.18255, 2026. URL https://api.semanticscholar.org/CorpusID:290395280.

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? *ArXiv*, abs/2310.06770, 2023. URL https://api.semanticscholar.org/CorpusID:263829697.

Sayash Kapoor, Benedikt Stroebl, Zachary S. Siegel, Nitya Nadgir, and Arvind Narayanan. AI Agents That Matter. *Transactions on Machine Learning Research*, 2025. URL https://api.semanticscholar.org/CorpusID:270870360.

Arpandeep Khatua, Hao Zhu, Peter Tran, Arya Prabhudesai, Frederic Sadrieh, Johann K. Lieberwirth, Xinkai Yu, Yicheng Fu, Michael J. Ryan, Jiaxin Pei, and Diyi Yang. CooperBench: Why Coding Agents Cannot be Your Teammates Yet. *ArXiv*, abs/2601.13295, 2026. URL https://api.semanticscholar.org/CorpusID:284910219.

Yubin Kim, Ken Gu, Chanwoo Park, Chunjong Park, Samuel Schmidgall, A. Ali Heydari, Yao Yan, Zhihan Zhang, Yuchen Zhuang, Yun Liu, Mark Malhotra, Paul Pu Liang, Hae Won Park, Yuzhe Yang, Xuhai Xu, Yilun Du, Shwetak Patel, Tim Althoff, Daniel McDuff, and Xin Liu. Towards a Science of Scaling Agent Systems. *ArXiv*, abs/2512.08296, 2025. URL https://api.semanticscholar.org/CorpusID:283712193.

Yubin Kim, Chanwoo Park, Taehan Kim, Eugene Park, Samuel Schmidgall, Salman Rahman, Chunjong Park, Cynthia Breazeal, Xin Liu, Hamid Palangi, Hae Won Park, and Daniel McDuff. TeamBench: Evaluating Agent Coordination under Enforced Role Separation. *ArXiv*, abs/2605.07073, 2026. URL https://api.semanticscholar.org/CorpusID:288111239.

Guohao Li, Hasan Abed Al Kader Hammoud, Hani Itani, Dmitrii Khizbullin, and Bernard Ghanem. CAMEL: Communicative Agents for "Mind" Exploration of Large Language Model Society. In *Advances in Neural Information Processing Systems*, 2023a. URL https://api.semanticscholar.org/CorpusID:268042527.

Huao Li, Yu Quan Chong, Simon Stepputtis, Joseph Campbell, Dana Hughes, Michael Lewis, and Katia Sycara. Theory of Mind for Multi-Agent Collaboration via Large Language Models. In *Conference on Empirical Methods in Natural Language Processing*, 2023b. URL https://api.semanticscholar.org/CorpusID:264172518.

Tian Liang, Zhiwei He, Wenxiang Jiao, Xing Wang, Yan Wang, Rui Wang, Yujiu Yang, Shuming Shi, and Zhaopeng Tu. Encouraging Divergent Thinking in Large Language Models through Multi-Agent Debate. In *Conference on Empirical Methods in Natural Language Processing*, 2024. URL https://api.semanticscholar.org/CorpusID:258967540.

Chang Ma, Junlei Zhang, Zhihao Zhu, Cheng Yang, Yujiu Yang, Yaohui Jin, Zhenzhong Lan, Lingpeng Kong, and Junxian He. AgentBoard: An Analytical Evaluation Board of Multi-turn LLM Agents. *ArXiv*, abs/2401.13178, 2024. URL https://api.semanticscholar.org/CorpusID:267199917.

Grégoire Mialon, Clémentine Fourrier, Craig Swift, Thomas Wolf, Yann LeCun, and Thomas Scialom. GAIA: a benchmark for General AI Assistants. *ArXiv*, abs/2311.12983, 2023. URL https://api.semanticscholar.org/CorpusID:265351664.

João Moura. CrewAI. Software, 2023. URL https://github.com/crewAIInc/crewAI. Accessed: 2026-09-16.

Boye Niu, Yiliao Song, Kai Lian, Yifan Shen, Yu Yao, Kun Zhang, and Tongliang Liu. Flow: Modularized Agentic Workflow Automation. In *International Conference on Learning Representations*, 2025. URL https://api.semanticscholar.org/CorpusID:275515996.

Abdelghny Orogat, Ana Rostam, and Essam Mansour. Architectural Design, Not Only Model Intelligence, Governs Multi-Agent LLM Performance. *ArXiv*, abs/2602.03128, 2026. URL https://arxiv.org/abs/2602.03128.

Long Phan, Alice Gatti, Ziwen Han, Nathaniel Li, Josephina Hu, Hugh Zhang, Chen Bo Calvin Zhang, Mohamed Shaaban, John Ling, Sean Shi, Michael Choi, Anish Agrawal, Arnav Chopra, et al. A benchmark of expert-level academic questions to assess AI capabilities. *Nature*, 649:1139–1146, 2026. URL https://api.semanticscholar.org/CorpusID:275906652.

Chen Qian, Zihao Xie, YiFei Wang, Wei Liu, Kunlun Zhu, Hanchen Xia, Yufan Dang, Zhuoyun Du, Weize Chen, Cheng Yang, Zhiyuan Liu, and Maosong Sun. Scaling Large-Language-Model-based Multi-Agent Collaboration. *ArXiv*, abs/2406.07155, 2024. URL https://api.semanticscholar.org/CorpusID:270379482.

Xinxing Ren, Caelum Forder, Qianbo Zang, Ahsen Tahir, Roman J. Georgio, Suman Deb, Peter Carroll, Önder Gürcan, and Zekun Guo. Anemoi: A Semi-Centralized Multi-agent System Based on Agent-to-Agent Communication MCP server from Coral Protocol. *ArXiv*, abs/2508.17068, 2025. URL https://api.semanticscholar.org/CorpusID:280711792.

Stack Exchange, Inc. Stack Exchange Data Dump. Internet Archive, 2014. URL https://archive.org/details/stackexchange. Dataset; accessed September 16, 2026.

Nurbek Tastan, Samuel Horváth, and Karthik Nandakumar. Stochastic Self-Organization in Multi-Agent Systems. *ArXiv*, abs/2510.00685, 2025. URL https://api.semanticscholar.org/CorpusID:281706587.

Harsh Trivedi, Niranjan Balasubramanian, Tushar Khot, and Ashish Sabharwal. MuSiQue: Multi-hop Questions via Single-hop Question Composition. *Transactions of the Association for Computational Linguistics*, 10:539–554, 2022. URL https://api.semanticscholar.org/CorpusID:236771976.

Harsh Trivedi, Tushar Khot, Mareike Hartmann, Ruskin Manku, Vinty Dong, Edward Li, Shashank Gupta, Ashish Sabharwal, and Niranjan Balasubramanian. AppWorld: A Controllable World of Apps and People for Benchmarking Interactive Coding Agents. *ArXiv*, abs/2407.18901, 2024. URL https://api.semanticscholar.org/CorpusID:271516633.

Junlin Wang, Jue Wang, Ben Athiwaratkun, Ce Zhang, and James Zou. Mixture-of-Agents Enhances Large Language Model Capabilities. *ArXiv*, abs/2406.04692, 2024. URL https://api.semanticscholar.org/CorpusID:270357878.

Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, Ahmed Hassan Awadallah, Ryen W. White, Doug Burger, and Chi Wang. AutoGen: Enabling Next-Gen LLM Applications via Multi-Agent Conversation. In *Conference on Language Modeling*, 2024. URL https://arxiv.org/abs/2308.08155.

Zijian Wu, Xiangyan Liu, Xinyuan Zhang, Lingjun Chen, Fanqing Meng, Lingxiao Du, Yiran Zhao, Fanshi Zhang, Yaoqi Ye, Jiawei Wang, Zirui Wang, Jinjie Ni, Yufan Yang, Arvin Xu, and Michael Qizhe Shieh. MCPMark: A Benchmark for Stress-Testing Realistic and Comprehensive MCP Use. *ArXiv*, abs/2509.24002, 2025. URL https://api.semanticscholar.org/CorpusID:281674925.

Wujiang Xu, Zujie Liang, Kai Mei, Hang Gao, Juntao Tan, and Yongfeng Zhang. A-MEM: Agentic Memory for LLM Agents. *ArXiv*, abs/2502.12110, 2025. URL https://api.semanticscholar.org/CorpusID:276421617.

Yingxuan Yang, Huacan Chai, Shuai Shao, Yuanyi Song, Siyuan Qi, Renting Rui, and Weinan Zhang. AgentNet: Decentralized Evolutionary Coordination for LLM-based Multi-Agent Systems. *ArXiv*, abs/2504.00587, 2025. URL https://api.semanticscholar.org/CorpusID:277468263.

Shunyu Yao, Noah Shinn, Pedram Razavi, and Karthik Narasimhan. τ-bench: A Benchmark for Tool-Agent-User Interaction in Real-World Domains. *ArXiv*, abs/2406.12045, 2024. URL https://api.semanticscholar.org/CorpusID:270562578.

Sukwon Yun, Jie Peng, Pingzhi Li, Wendong Fan, Jie Chen, James Zou, Guohao Li, and Tianlong Chen. Graph-of-Agents: A Graph-based Framework for Multi-Agent LLM Collaboration. In *International Conference on Learning Representations*, 2026. URL https://api.semanticscholar.org/CorpusID:287634739.

Guibin Zhang, Muxin Fu, Guancheng Wan, Miao Yu, Kun Wang, and Shuicheng Yan. G-Memory: Tracing Hierarchical Memory for Multi-Agent Systems. *ArXiv*, abs/2506.07398, 2025a. URL https://api.semanticscholar.org/CorpusID:279250852.

Guibin Zhang, Luyang Niu, Junfeng Fang, Kun Wang, Lei Bai, and Xiang Wang. Multi-agent Architecture Search via Agentic Supernet. *ArXiv*, abs/2502.04180, 2025b. URL https://api.semanticscholar.org/CorpusID:276161505.

Jiayi Zhang, Jinyu Xiang, Zhaoyang Yu, Fengwei Teng, Xionghui Chen, Jiaqi Chen, Mingchen Zhuge, Xin Cheng, Sirui Hong, Jinlin Wang, Bingnan Zheng, Bang Liu, Yuyu Luo, and Chenglin Wu. AFlow: Automating Agentic Workflow Generation. *ArXiv*, abs/2410.10762, 2024. URL https://api.semanticscholar.org/CorpusID:273345847.

Yuzhe Zhang, Feiran Liu, Yi Shan, Xinyi Huang, Xin Yang, Yueqi Zhu, Xuxin Cheng, Cao Liu, Ke Zeng, Terry Jingchen Zhang, and Wenyuan Jiang. Silo-Bench: A Scalable Environment for Evaluating Distributed Coordination in Multi-Agent LLM Systems. In *Annual Meeting of the Association for Computational Linguistics*, 2026. URL https://api.semanticscholar.org/CorpusID:286222964.

Kunlun Zhu, Hongyi Du, Zhaochen Hong, Xiaocheng Yang, Shuyi Guo, Zhe Wang, Zhenhailong Wang, Cheng Qian, Xiangru Tang, Heng Ji, and Jiaxuan You. MultiAgentBench: Evaluating the Collaboration and Competition of LLM agents. *ArXiv*, abs/2503.01935, 2025. URL https://api.semanticscholar.org/CorpusID:276766372.

Mingchen Zhuge, Wenyi Wang, Louis Kirsch, Francesco Faccio, Dmitrii Khizbullin, and Jürgen Schmidhuber. Language Agents as Optimizable Graphs. In *International Conference on Machine Learning*, 2024. URL https://api.semanticscholar.org/CorpusID:268032156.

Appendix Contents

A Expert Environment **15**
A.1 Communities and Corpora 15
A.2 Public Profile and Private Retrieval 15
A.3 Source Identifiers and Memorization 16

B Task Construction **17**
B.1 From Community Posts to a Composed Task 17
B.2 Certifying a Task's Dependency Graph 18
B.3 Quality Control and Family Structure 20
B.4 Dataset Composition 20

C Evaluation Protocol **21**
C.1 Execution and Logging 21
C.2 Outcome Verification 23
C.3 Expert Selection 25
C.4 Stage Decisions along the Dependency Chain 25
C.5 Can a Dependency Be Bypassed? 26

D Compared Methods **27**
D.1 Integration 27
D.2 The Single Agent Reference 28
D.3 Exhaustive Collaboration 28
D.4 Memory Modules 28

E Preliminary Study **30**
E.1 Tasks and Settings 30
E.2 Masking a Required Contribution 30
E.3 Locating a Discarded Finding 31

F Full Results **31**
F.1 Where Certified Dependencies Go 31
F.2 Exhaustive Collaboration 32
F.3 Memory Modules 33
F.4 The No-Collaboration Reference 34

G Qualitative Examples **35**
G.1 One Task in Full 35
G.2 Planning Rigidity 35
G.3 Semantic Compression 37
G.4 Adjudicative Aggregation 37

H Prompts **38**

I Limitations and Data **39**
I.1 What the Findings Cover 39
I.2 What the Environment Leaves Out 40
I.3 Data and Release 40

Table 4: The 51 expert communities. Each agent is built from one Stack Exchange site, and the resource groups organize the pool for reporting. Tasks cross the grouping freely, so a single query may draw on one group or on five.

Resource group	Sites	Communities
Software & Web	13	Stack Overflow, Code Review, Software Engineering, SQA, Code Golf, Game Development, Webmasters, WordPress, Drupal, Magento, SharePoint, Salesforce, Software Recommendations
Systems & Networks	12	Unix & Linux, Server Fault, Ask Ubuntu, Super User, Network Engineering, DevOps, Emacs, Vi, Android, Apple, Security, Reverse Engineering
Data & Computation	10	Data Science, Cross Validated, Artificial Intelligence, Database Administrators, Computer Science, Theoretical CS, Mathematics, Scientific Computing, Operations Research, Cryptography
Hardware & Eng.	7	Electrical Engineering, Arduino, Raspberry Pi, Robotics, Signal Processing, Engineering, Space Exploration
Professional Practice	5	Law, Project Management, English Language, TEX, User Experience
Natural & Spatial Sci.	4	Physics, Chemistry, Biology, Geographic Information Systems

A Expert Environment

MAScope builds its agent pool from 51 experts whose knowledge does not overlap. This section gives the communities behind them (§ A.1), the two surfaces a method and an agent work through (§ A.2), and the mechanism that keeps a memorized copy of Stack Exchange from passing as evidence (§ A.3).

A.1 Communities and Corpora

Each expert agent is built from one Stack Exchange community (Stack Exchange, Inc., 2014), taken from the data dump of 1 March 2026. We keep a question–answer pair when the question has an accepted answer, a body of at least 80 tokens, and at least one topic tag. The first condition gives every retained pair a community-endorsed resolution, and the other two remove stubs that carry no retrievable content. We then remove near-duplicates within a community, treating two pairs as duplicates when the MinHash similarity of their question bodies is at least 0.90 and keeping the one whose accepted answer has the higher score, so that a single widely reposted issue cannot be retrieved as several independent findings.

Retained pairs are capped at 20,000 per community, sampled uniformly at random. Without the cap Stack Overflow alone would hold two orders of magnitude more content than the smallest community, making one agent a near-universal oracle and turning recruitment into a question about corpus size. After capping, the pool holds 766,930 pairs, with a median of 18,271 per community and a minimum of 3,247. Twenty-three of the 51 corpora sit at the cap. Every retained pair is stored with the identifier introduced in § A.3. Table 4 lists the 51 communities in the six resource groups used throughout the paper, and Table 5 reports the size and the pair length of the resulting corpora.

The six groups describe the pool. A task may draw on one group or on five, and Figure 4 shows how the 2,766 queries spread over structure and scale once composed.

A.2 Public Profile and Private Retrieval

An agent exposes two surfaces, and the boundary between them is what turns recruitment into a measurable decision. Figure 6 shows both on one agent.

Public profile. The profile is the only thing a MAS method reads before recruiting. It carries the fields in Table 6 and never contains an answer, so recruiting well requires inferring the needed expertise from the question itself. The expertise scope is one sentence that states both what the agent

Table 5: Corpus statistics by resource group. We report the question–answer pairs each group retains after filtering and near-duplicate removal, under a cap of 20,000 pairs per community that 23 of the 51 corpora reach. The bar carries the group total against the largest group. Lengths are question plus accepted answer in backbone tokens, and the per-site columns are medians and ranges over the communities of a group.

Resource group	Sites	Q–A pairs	Median / site	Min–max / site	Median length
Software & Web	13	214,076	20,000	4,412–20,000	731
Systems & Networks	12	181,345	18,421	3,914–20,000	629
Data & Computation	10	134,709	13,480	3,247–20,000	669
Hardware & Eng.	7	86,440	9,832	5,274–20,000	671
Professional Practice	5	79,602	18,271	6,428–20,000	636
Natural & Spatial Sci.	4	70,758	18,921	12,916–20,000	673
Total	**51**	**766,930**	**18,271**	**3,247–20,000**	**671**

Table 6: Fields of a public profile. These four fields are everything a method knows about an agent when it decides whom to recruit, and none of them carries an answer.

Field	Content
Site name	The community the agent is built from, for example *Server Fault*.
Expertise scope	One sentence stating what the agent covers and what falls outside it.
Topic tags	The twelve most frequent tags of the retained corpus, sketching its sub-areas.
Corpus size	The number of question–answer pairs the agent can retrieve from.

covers and what falls outside it, and the topic tags are the twelve most frequent tags of the retained corpus, which sketch its sub-areas without naming any individual post.

Retrieval interface. During execution an agent queries its own corpus in natural language and receives at most eight passages, each a retained question–answer pair returned with its source identifier. The call ranks the corpus with BM25 over the question and accepted answer of each pair ($k_1 = 0.9$, $b = 0.4$) and returns the eight highest, so the same query returns the same passages on every run and to every method, and anyone can recompute the ranking from the released corpora. Dependency certification decides reach on this same call (§ B.2). An agent cannot query another agent's corpus, and nothing in the interface reveals which other experts have been recruited or how many agents the team holds. Retrieval is the only way evidence enters a trajectory, and every call is charged to the task budget, so retrieval competes with the rest of the run for the same allowance. Whatever a method learns about its teammates it learns from their messages, which is what § 3.3 measures as information transfer.

A.3 Source Identifiers and Memorization

Stack Exchange is almost certainly in the pretraining data of any current backbone. Our design makes memorization insufficient by construction.

Every information unit in U is bound to an identifier that we mint inside the benchmark. An identifier is the prefix `MS-` followed by twelve characters of Crockford's base-32 alphabet, derived from a keyed hash of the community, the post, and the unit index, under a key generated once at construction time and never released. Two properties follow. The identifier postdates any pretraining corpus, so no backbone holds it in its weights, and it depends on a key that never leaves construction time, so a memorized copy of the post gives no way to reconstruct it. Acceptance under V requires the final answer to carry the identifier of each required unit, which means a backbone that has memorized the underlying content still cannot produce an accepted answer without an agent retrieving from the corpus first. Memorization may help an agent phrase a finding. It cannot manufacture the token that proves the finding was obtained.

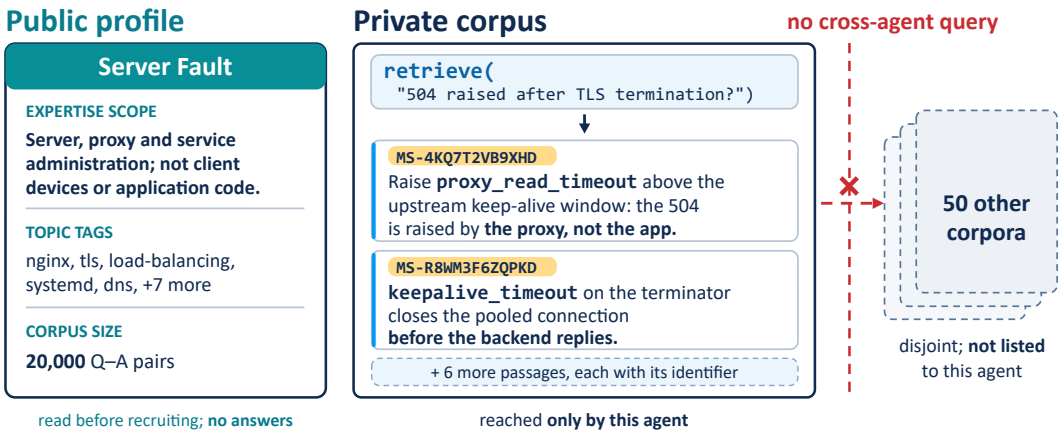


Figure 6: The two surfaces of one expert agent. A method reads the public profile (**left**) before recruiting, and the profile contains no answer. The recruited agent reaches its own corpus (**middle**) through a natural-language call that returns passages carrying benchmark-internal identifiers. No retrieval call reaches another agent's corpus (**right**), so evidence held by one expert can enter the answer only if the team carries it there.

Empirical check. We verify this directly. From the 2,766 queries we draw 270, stratified over the nine $C \times S$ cells in proportion to cell size so that the subsample carries the same difficulty mix as the benchmark, and run AutoGen on them twice, once as in the main experiments and once with the corpus of one required agent made unreachable, chosen uniformly among that query's required experts, with every other condition unchanged. The required expertise is still present in the pool and the agent still joins the team, and only its evidence is gone. Averaged over five runs, task success falls from 52.9% to 4.7%, or 142.8 to 12.6 of 270 queries, and the unmodified condition reproduces the 52.1% of Table 1 to within the noise of a 270-query subsample. Every success under removal comes from a query whose reference records an equivalent unit in another recruited corpus, so an alternative route covers the missing evidence, and no accepted answer in this condition cites an identifier from the removed corpus. Evidence therefore enters a trajectory through the corpus alone.

Licensing and anonymity. Stack Exchange content is released under CC BY-SA (Stack Exchange, Inc., 2014). We redistribute derived corpora under the same terms and keep the attribution the license requires, shipping the community, post identifier and revision date of every retained pair so each can be traced to its source. User names, profile links and other author-identifying metadata are stripped before a corpus enters the environment, so an agent never sees who wrote it.

B Task Construction

Two properties define a `MAScope` task. Every required unit is bound to one corpus, and every dependency among units holds for any correct solution of the query that carries it. Construction enforces both. This section gives the pipeline that produces a task (§ B.1), the protocol that certifies a task's whole dependency graph (§ B.2), the checks a task clears before it is frozen (§ B.3), and the composition of the released set (§ B.4).

B.1 From Community Posts to a Composed Task

Construction runs as a pipeline of four stages, each closing on a gate that discards the candidates failing it. Figure 7 traces one task through all four and records how many candidate groups each gate lets through.

Selecting candidate units. A candidate group starts from one accepted answer and grows by retrieving, from the other 50 corpora, answers whose content a single user goal would also require. A group is kept when its members come from at least two communities and none of them subsumes another. Subsumption is decided by solving. A unit u_j is subsumed by u_i when u_j is accepted with only the passages supporting u_i in context, so u_j adds no expertise to the group.

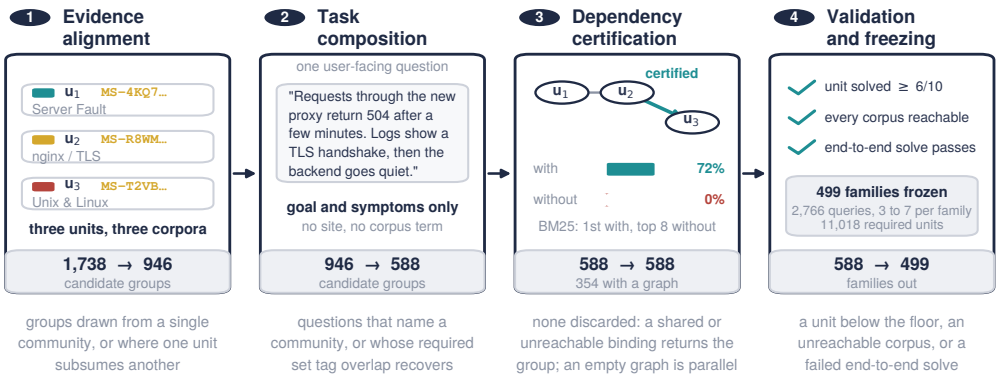


Figure 7: The task construction pipeline. A candidate group of accepted answers becomes information units, the units become one user-facing question, the dependencies among them are certified by two deterministic rules, and the task is frozen once an end-to-end solve passes. The tinted box in each panel carries the candidate groups entering and leaving the stage, and the line beneath gives the rule that discards a candidate. Certification discards no group, since a group with a shared or unreachable binding returns to composition and a group whose certified graph is empty continues as a parallel task.

Composing the question. The units of a group are rewritten into one question that states a goal and the symptoms a user would observe, names no community, and uses no vocabulary confined to a single corpus. Composition is model-assisted, and two of the authors review every composed question independently against a four-item checklist, with a third resolving disagreements. The checklist asks that no community be named, that only the goal and observable symptoms appear, that no term occur in a single corpus, and that every required unit be demanded without being named. A question is also rejected when its surface gives the required expertise away, which a tag-overlap baseline tests. Each of the 51 public profiles is scored by the overlap between its topic tags and the tokens of the question, and the question is rejected when the top $|R_q|$ profiles under that score already contain the full required set. Of the 358 groups discarded at this stage, 211 fail the checklist and 147 have their required set recovered by the baseline. The baseline reads only the surface of the question, and whether the expertise a question does not hide can be inferred from it is checked by the end-to-end solve of validation (§ B.3).

B.2 Certifying a Task's Dependency Graph

A unit is dependent when its inputs or its objective can be fixed only once a predecessor finding is in hand. We read which units those are off the reference itself, working backward from what acceptance demands, and no model takes part in deciding an edge. Certification is decided once per candidate group, and every query composed from the group carries the graph it returns (§ B.3). Inside a group every other required unit is a candidate predecessor of every unit, the two rules below decide every candidate pair, and the gate they form returns a group whose bindings they cannot settle, so that a released query carries every dependency its reference implies and nothing more.

Certification protocol. The graph is derived backward from V. For each unit u_j, the terms $V(u_j)$ requires that neither q, in any query of the group, nor u_j's own source passage states are its bound terms, what an accepted statement of u_j has to carry and its own evidence cannot supply. A bound term is specific by rule, a quantity with its unit, a configuration key, a version or a named component, and never a general concept. A candidate u_i is certified as a predecessor of u_j when two conditions hold. The first, binding, holds when u_i is the one unit of the task whose source passage states a bound term of u_j. The second, reach, holds when u_j's source passage, and every equivalent the reference records for it, stays outside the top eight of u_j's corpus under a fixed query battery and enters the top eight once u_i's finding is added. The battery holds q, each sentence of q, and u_j's objective with its bound terms removed, each also joined with the findings of every other candidate. The battery is built for every query of the group, since each has its own q, and reach has to hold on all of them. Reach is decided on the environment's own BM25 ranking (§ A.2), the call every

method makes and the only way evidence enters a trajectory. Both conditions read frozen text and a deterministic ranking, so the certified graph is a property of the benchmark and its retrieval interface, and it does not move with a prompt, a harness, a backbone or a sampling seed.

Two levels of necessity follow, and they rest on different grounds. The first is a property of the reference. An accepted answer states u_j with its bound terms, and within the task those terms are stated only in u_i's source, so every accepted answer carries information that only u_i supplies, whatever route the run took to it. The second is a property of the environment. The identifier of u_j arrives only with its passage, reach keeps that passage out of the eight an agent sees for every query in the battery until u_i's finding is added, and the battery is built from the question, its sentences and the other findings, which is what an agent has to search from before the finding exists. Reach is what makes the handoff necessary under the environment's own retriever, and since an agent may phrase a query the battery does not contain, § C.5 measures the residue at run time and finds it on a fifth of a percent of edges pooled over the methods and below half a percent for every one of them. Because the battery already holds every other candidate's finding, the rules return immediate predecessors only, and with u_i's finding in hand, withholding something u_i itself depends on leaves u_j neither unbound nor out of reach. The certified in-edges of every unit form the query's dependency graph G_q. Neither rule asks whether u_j can be solved once its findings arrive. That is the local-solvability check of § B.3, the one check around a dependency in which a model solves, and it establishes that the task can be done, while the two rules establish what it requires.

Returned groups. Certification is a gate like the two before it, except that a failing group returns to composition and stays in the pipeline. A group goes back to composition when a bound term of one of its units is stated by two of the others, or by none, when a pair binds but leaves the downstream passage within reach, or when its bindings would close a cycle. Composition then rewrites the objective that carried the term, or the question that gave the passage away, and the group re-enters. Every released query thus has the property the evaluation needs. A unit's acceptance requires a term from outside its own source and q only through a certified edge, so the graph a query carries is its whole dependency structure, with no dependency left unscored and no released graph cyclic. Completeness here is relative to the reference, in that every term acceptance takes from another unit has its certified edge. The equivalent evidence and alternative routes a task admits are recorded in the reference as well (§ C.2), and a run that takes one is scored on the units it carried. A group none of whose units carries a bound term leaves with an empty graph and continues as a parallel task. That is what a C1 query is, one whose units are accepted independently of one another, so that dividing it costs nothing but the division.

Of the 588 groups that reach the gate, 72 are returned on their first pass, 38 for a term two units state, 3 for a term no unit states, 27 for reach and 4 for a cycle, and 6 of them are returned once more before all 588 clear it. Counted over the dependent queries, where a query with k required units has $k(k-1)$ ordered candidate pairs, each of the 23,758 pairs in the released set falls in one of two classes, 20,094 that carry no bound term of the downstream unit and 3,664 that pass both rules. A certified edge carries 1.54 bound terms on average. Without its finding the downstream passage sits at a median best rank of 75 over the query's battery and never nearer than 11th, well outside the eight passages an agent sees, and with the finding added it ranks first for 72.2% of edges and within the eight for all of them. Table 7 reports both by cell. Since the battery supplies every other candidate's finding, a larger task brings the passage closer without its own predecessor, from a median rank of 101 at S1 in C2 to 59 at S3 in C3, and never inside the eight.

What a task carries. The 1,630 dependent queries carry 3,664 certified edges between them, 2.25 each, rising with the work a query requires from 1.30 at S1 in C2 to 4.10 at S3 in C3. A graph runs one to four edges deep: 57% of the dependent queries carry a graph one edge deep, 29% two, and the rest three or four. Each query carries its family's G_q and is scored on every edge of it (§ 3.3). Certification also shapes the released set. Of the 588 groups that clear it, 354 leave with a non-empty graph and the other 234, whose units carry no bound term, continue to validation as parallel tasks.

Expert discovery. When the finding of u_i is also what identifies the community holding u_j, we record that edge as an expert-discovery relationship, and a task whose graph carries at least one of them enters C3. The check is deterministic as well and uses the tag-overlap score of § B.1. Scored against q, the downstream community's profile must fall outside the top $|R_q|$ of the 51, so

Table 7: Dependency certification. We report the two deterministic rules that fix each dependent query's graph. A candidate u_i is a predecessor of u_j when it is the one unit whose source states a term that $V(u_j)$ requires and that neither q nor u_j's own source states, and when u_j's source passage is out of BM25 reach from the fixed query battery until u_i's finding is added. Counts are over the dependent queries, where a query with k units has $k(k-1)$ ordered *Candidates* and carries its family's graph. The bar carries a cell's certified edges against the largest cell. *Per query* is the certified edges a dependent query carries, *Bound* the mean number of bound terms an edge carries, *Rank w/o* the median best rank of u_j's passage over the query's battery without the finding, never inside the eight an agent sees, and *Top-1* the share of edges whose passage ranks first once the finding is added.

		Certified dependencies			Binding and reach		
C	S	**Candidates**	**Certified**	**Per query**	**Bound**	**Rank w/o**	**Top-1**
C2	S1	1,512	514	1.30	1.55	101	78.6
C2	S2	3,936	710	2.26	1.44	85	74.9
C2	S3	8,886	914	3.87	1.65	64	68.3
C3	S1	1,008	388	1.31	1.38	102	77.1
C3	S2	2,904	506	2.14	1.51	74	69.8
C3	S3	5,512	632	4.10	1.62	59	68.4
Total		**23,758**	**3,664**	**2.25**	**1.54**	**75**	**72.2**

the question alone does not point to it, and scored against the finding of u_i it must rank within the top three, so the clue sits inside the finding and a method reaches the downstream expert through u_i. Across the families that pass, the downstream community ranks at a median of 21st against q and never nearer than 9th, and first against the finding for 88.9% of them. Of the 300 dependent families, 126 pass this check and form C3, and the other 174 form C2.

B.3 Quality Control and Family Structure

Local solvability. Every required unit of every family is answered against its own corpus alone, ten samples, with no other agent present and with the findings of its certified predecessors in its work, covering 1,994 unit occurrences, one per unit and family, since a unit's work is the family's. A unit solved in fewer than $6/10$ samples is revised or dropped. Across the released set the median unit is solved in $9/10$ samples and the fifth percentile in $7/10$, a mean of 90.9%, which fixes an agent-side floor for everything measured downstream. Validation also solves every task end to end with the expert prompt of § H, and a task is revised or dropped there when one of its corpora cannot be reached or when an expert that no finding hides cannot be inferred from q.

Families. Queries composed from the same candidate group form a family. All members of a family share the same units, the same required experts, the same collaboration structure and the same certified graph, and differ in which observable symptoms the question surfaces and how they are worded, so no query restates another. A family holds three to seven queries, 5.5 on average: 26 families hold three, 56 four, 147 five, 161 six and 109 seven. All 2,766 queries are scored, and the methods of § 4 run in their native configuration without tuning on the benchmark. The family label ships with every query, because members of a family share evidence. Work that tunes on part of the set should hold out whole families, so that no variant of a scored task is seen.

Yield. The pipeline freezes 499 families from 1,738 candidate groups, 199 of them parallel and 300 dependent, with the attrition concentrated in the first two gates.

B.4 Dataset Composition

Table 8 reports the released set over the nine cells of collaboration structure and required expertise. Every cell carries at least 154 queries, and the six dependency cells together hold 1,630 queries, the population over which the three pipeline capabilities of § 3.3 are measured, carrying 3,664 preregistered dependencies between them. Within a scale bin the exact expert count follows whichever candidate groups survived certification. S2 is realized as four required experts in all but forty-two

Table 8: Composition of the released set. We report the 2,766 queries over collaboration structure C (■ C1 parallel, ■ C2 sequential, ■ C3 expert discovery) crossed with required expertise S. The bar carries the query count of a cell against the largest cell. A family is the 3 to 7 queries composed from one candidate group, which share its units, experts, structure and dependency graph and differ in the symptoms they surface and their wording, and *Per family* is the mean number of queries a family holds. *Experts* gives the expert counts a cell realizes and *Mean* their average per query, which agrees across the three structures at every scale to within 0.14 experts, so C and S can be read independently. The six C2 and C3 cells carry a certified dependency graph per query, and the 1,630 queries they hold are the population over which task orchestration, information transfer and result integration are measured.

		Released set			Required expertise		
C	S	Queries	Families	Per family	Experts	Mean	Units
■ C1	S1	454	82	5.5	2–3	2.38	1,082
■ C1	S2	390	67	5.8	4–5	4.03	1,572
■ C1	S3	292	50	5.8	6–8	6.55	1,913
■ C2	S1	394	73	5.4	2–3	2.46	969
■ C2	S2	314	56	5.6	4–5	4.07	1,277
■ C2	S3	236	45	5.2	6–8	6.61	1,560
■ C3	S1	296	53	5.6	2–3	2.35	696
■ C3	S2	236	42	5.6	4–5	4.04	953
■ C3	S3	154	31	5.0	6–8	6.47	996
Total		**2,766**	**499**	**5.5**	**2–8**	**3.98**	**11,018**

queries, twelve of them in C1, twenty-one in C2 and nine in C3, which require five, and S3 spreads over six, seven and eight. The dependency drop of § 4 does not turn on that spread. At S2, where a dependent query and a parallel one require the same four experts in all but those forty-two, the twelve methods lose a mean of 52.0% of their parallel success on dependent tasks and the single agent 16.5%, the same picture as over all three scales. The per-cell counts of Table 8 are the denominators used throughout § 4.

C Evaluation Protocol

An evaluation decision in `MAScope` is a match between a logged trajectory and the reference of the task that produced it. Source identifiers and preregistered terms carry the match, and every verdict is stored with the trajectory. This section gives what a method receives and what the environment records (§ C.1), the three outcome metrics (§ C.2), the two expert-selection metrics (§ C.3), the five-stage decision on every certified edge of a query's dependency graph (§ C.4), and how often a method gets around a certified edge at run time (§ C.5).

C.1 Execution and Logging

What a method controls. A method receives the question q, the 51 public profiles, the shared agent pool, the retrieval interface of § A.2, and a budget. The question arrives with one line of answer format appended, the same for every method and every query (§ H), asking that the final answer state the identifier of every passage it rests on. The line says what an answer has to carry and nothing about how to produce it. Everything after that belongs to the method. It decides whom to recruit, how to divide the work, what each agent is told, how many rounds to run, and when to submit. `MAScope` supplies no coordinator, no communication topology and no termination rule of its own, so what the evaluation observes is the method's own mechanism.

Budget and runs. Each task is capped at 2M tokens and 256 model calls, and the consumption we report is the one actually incurred. A run that reaches either cap submits whatever answer it holds at that point, and its trajectory is scored as it stands. Every model call in the benchmark decodes at temperature 0.3 with top-p 0.95 and at most 4,096 output tokens, and the construction-time solves of § B run under the same settings. Every method runs five times over all 2,766 queries, with the sampling seed of the backbone as the only difference between them. Queries, references,

Table 9: How often a run reaches a budget cap. Share of queries whose run reached the 2M-token or the 256-call cap of § C.1 and submitted what it held, over five runs, overall and among the S3 queries. The row for the other ten methods gives the largest share among them, which is zero, since no run of theirs reached either cap in five runs over 2,766 queries.

Method	Token cap (%)	Call cap (%)	Either, at S3 (%)
MoA	0.46	0	1.29
MAD	0.38	0	1.14
Other ten methods	0	0	0
Single Agent	0	0	0

Table 10: The common event stream. One adapter per method rewrites its native log into these nine events, changing no behavior and adding no field the framework did not produce. An event a method has no counterpart for is an event its adapter never emits, and every predicate of Table 13 is decided from these fields alone.

Event	Fields	Written when
`recruit`	agent, round	a method adds an agent to the team for this query
`assign`	from, to, task text, carried units	a message delegates work to a named agent, whose carried units are the identifiers readable to the recipient as it takes the work
`message`	from, to, content, cited identifiers	any other agent-to-agent or coordinator message
`retrieve`	agent, query text, returned identifiers	an agent queries its own corpus
`local_output`	agent, content, cited identifiers	an agent returns a result to the team
`aggregate`	actor, inputs, output	a method merges results into a single artifact
`memory`	module, operation, items	a memory module is read or written (§ 4.5)
`submit`	answer, cited identifiers	the method submits its final answer, or a cap is reached
`usage`	call, prompt tokens, completion tokens	every model call and every retrieval call

preregistered dependencies, profiles, corpora and budget are identical across the five, so the spread they show is decoding variance together with the scheduling nondeterminism it induces inside a method. The standard deviations in Table 1 and Table 2 are taken over those five runs. A cap is reached rarely (Table 9). MoA and MAD, the two methods in which each of 51 agents reads fifty peers' answers, reach the token cap on 0.46% and 0.38% of their queries, 1.29% and 1.14% of their S3 queries, and no run of the other ten methods or of the single agent reaches either cap. The cap thus accounts for at most one in fifty of MoA's and MAD's S3 failures and for none at S1 and S2.

The event stream. Twelve frameworks record their work in twelve formats, and the evaluator reads none of them directly. An adapter per method rewrites the native log into the common event stream of Table 10, changing no behavior and adding no field the framework did not already produce. An event a framework never performs is an event the adapter never emits, and a predicate that needs it is recorded as unreached. A native construct that packages several events into one is unpacked into them. A group-addressed message becomes one `assign` per recipient the reference requires, and a coordinator turn that merges results and delegates the next step becomes an `aggregate` followed by an `assign`. The carried units of an `assign` are defined by what the recipient can read when it takes the work, namely the identifiers and terms in the assignment text, in the messages addressed to it so far, in the peer responses a framework places in its prompt, as GoA and SelfOrg do, and in any shared store it reads at that point, the thread of Anemoi or a Blackboard module. A finding that reaches an agent through a peer's response or a shared thread thus counts as delivered, and what the transfer rate of Table 2 charges is content the recipient could not read anywhere, whether it was written short (§ G.3) or routed to someone else. Table 11 gives, for every method, the native construct behind each of the four events whose form differs across frameworks, so a rate of Table 2 can be read back to the construct it was decided on. Figure 3 shows one such stream on a service-diagnosis task, with the checks of this section marked where they apply.

Table 11: What each adapter rewrites into the four team-level events. `retrieve` is the one tool call of every agent and `local_output` an agent's reply to its team, the same in every framework, and the four events here are the ones whose native form differs. Where a construct packages several events, it is unpacked into them, so an aggregator's prompt to its proposers is one `assign` per recipient the reference requires, and a coordinator turn that merges and delegates is an `aggregate` followed by an `assign`. An event a framework never performs is never emitted.

Method	`recruit`	`assign`	`aggregate`	`submit`
Centralized				
AutoGen	the manager's first selection of an agent as speaker	the manager's speaker selection and the message that prompts it; the transcript is visible to every participant	the manager's closing summary	the termination message
Magentic-One	a specialist the orchestrator names in the task ledger	the orchestrator's next-speaker instruction	a progress-ledger update merging results	the orchestrator's final answer
Flow	a worker allocated a node of the subtask graph	a node dispatched to its worker with the outputs of the nodes it depends on	a graph update merging completed nodes	the output of the last node
CrewAI	the manager's first delegation to a crew member	the manager's delegation of a task from the list fixed at kickoff	the manager's synthesis of task outputs	the kickoff result
LangGraph	the supervisor's first handoff to a worker	the supervisor's handoff to a worker, with the message history it passes	the supervisor's turn after a worker returns	the supervisor's answer to the user
Semi-centralized				
Anemoi	an agent the planner adds to the thread	a plan item, or a message addressed to the agent in the thread, which it reads in full	the participants' consensus vote on a candidate answer	the answer-finding agent's submission
AgentVerse	a role staffed in the recruit stage	an action fixed in the decide stage, one per acting agent	the evaluate stage's merged verdict	the evaluator accepting the result
MoA	every agent, as a proposer of the first layer	a layer's prompt to an agent, carrying every output of the previous layer	the final aggregator's synthesis	the aggregator's output
MAD	every agent of the pool, which answers the query in round one	the round-two prompt to an agent, carrying every other agent's round-one answer	the majority vote over the round-two answers	the answer the vote selects
Decentralized				
GoA	a node the meta-LLM samples from the profiles	the source-to-target or target-to-source prompt that hands a node its neighbours' responses, weighted by their scores	the pooling step: the top source's refined response or the meta-LLM's weighted synthesis	the pooled response
AgentNet	an agent the router forwards or splits a task to	the forwarded task, unchanged, or the split with the parts already done	an agent's merge of what returns to it	an agent's final answer
SelfOrg	every agent of the pool, which answers the query in round 0	the next-round prompt carrying the responses of an agent's predecessors in the contribution-scored DAG	the contribution-weighted centroid that selects one existing response	the selected response after the last round

C.2 Outcome Verification

Task success. Acceptance is decided by V, which lists for every required unit the source identifier the final answer must carry and the preregistered terms its content must state, together with the equivalent evidence and the alternative routes recorded during construction. A run succeeds when the answer satisfies every entry of V. Two kinds of match decide an entry, and both are decided by

Table 12: Acceptance with and without the identifier requirement. *Identifier-free* rescores every submitted answer with the identifier requirement of § C.2 dropped, so that an answer stating every required unit's preregistered terms is accepted whether or not it carries the identifiers, and Δ is the points that adds. Δ_{dep} is the relative drop in success from C1 to C2∪C3 under each rule. Means over five runs.

	Success (%)			Δ_{dep} (%)	
Method	**Strict**	**Identifier-free**	Δ	**Strict**	**Identifier-free**
Centralized					
AutoGen	52.1	53.0	+0.9	−35.9	−34.9
Magentic-One	50.2	51.2	+1.0	−38.3	−36.9
Flow	46.6	47.9	+1.3	−50.4	−48.0
CrewAI	44.9	46.4	+1.5	−58.9	−56.3
LangGraph	42.3	43.3	+1.1	−39.5	−38.1
Semi-centralized					
Anemoi	44.3	46.0	+1.7	−43.1	−40.5
AgentVerse	41.9	43.1	+1.2	−51.5	−49.6
MoA	32.0	34.8	+2.9	−70.9	−65.4
MAD	29.4	31.9	+2.4	−54.6	−48.8
Decentralized					
GoA	46.2	47.7	+1.6	−46.1	−43.7
AgentNet	33.7	35.6	+1.9	−42.7	−39.3
SelfOrg	32.3	35.1	+2.8	−51.0	−46.0
Reference					
Single Agent	41.0	41.6	+0.7	−15.1	−14.6

string, so acceptance is a function of the submitted answer and the reference alone. An identifier matches as an exact string, the fifteen characters of § A.3, anywhere in the submitted answer. No paraphrase produces one, which is what ties acceptance to retrieval. A preregistered term matches when the answer states it verbatim or in one of the surface variants recorded with the term during construction, which cover case, inflection, and the unit-bearing forms of a quantity. The variants are written when the task is built, frozen with it, and shipped with the reference, and no model reads an answer to decide whether it succeeded. Writing $V_q(y_q) \in \{0, 1\}$ for the verdict on the submitted answer y_q of query q, and with $b(q, u) = 1$ when unit u appears in that answer with its identifier and its preregistered terms,

$$\text{Success} = \frac{1}{N} \sum_q V_q(y_q), \qquad \text{Evidence coverage} = \frac{1}{N} \sum_q \frac{\sum_{u \in U_q} b(q, u)}{|U_q|}.$$

Success is the conjunction of b over U_q, so evidence coverage reads the same predicate one unit at a time and records what a failed run assembled. Acceptance asks whether every required contribution reaches the answer with its identifier and its preregistered terms, and it does not judge the reasoning that connects them. That reasoning runs on the same backbone for every method, so what acceptance separates between two methods is which contributions arrived, and the stages of § C.4 say where the others were lost. The identifier requirement is what ties an accepted statement to a retrieval (§ A.3), and rescoring every answer without it shows what it adds. With only the preregistered terms required, success rises by between 0.9 points for AutoGen and 2.9 for MoA, and by 0.7 for the single agent (Table 12). The rise is largest where a finding is restated before the writer sees it, by a layer, a vote or a peer's rewrite, and smallest where the transcript reaches the writer intact, so a lost identifier marks a finding that was restated on the way. The twelve methods keep the same ranking under both rules, and the mean dependency drop moves from 48.6% to 45.6%.

Table 1 reports success and Table 2 reports coverage, and the distance between them is what the failed answers still delivered, the units that reached an answer that fell short elsewhere.

Execution cost. Cost sums the tokens of every model call a method makes, across planning, execution, aggregation and memory, together with the retrieval calls charged to the task. Table 1 reports the per-task mean in thousands of tokens and the model calls over the five runs.

Table 13: Stage decisions on the certified edges of a query's dependency graph G_q. Each predicate is evaluated on one edge $u_i \to u_j$ of G_q, and only where the predicate above it holds on that edge, so every rate r_k is a ratio of counts over the edges that reached the previous stage, and $s^{(1)}$ is the entry condition that gives r_2 its denominator. Local solve is an agent-side reference, and it separates an agent failing to solve from a team failing to deliver.

	Capability	Rate	Predicate	Matched against
$s^{(1)}$	**Ready**	–	the method holds the finding of u_i and has recruited the expert holding u_j	a `local_output` citing u_i, and a `recruit` for that expert
$s^{(2)}$	**Task orchestration**	r_2	the finding is turned into work for that expert	an `assign` to that expert, later than the finding, whose task text carries the objective terms of u_j
$s^{(3)}$	**Information transfer**	r_3	the assignment carries the predecessor evidence and its key constraints	the identifier of u_i and its constraint terms present in what the recipient can read as it takes the work, the carried units of that `assign`
$s^{(4)}$	**Local solve**	r_4	the expert produces a correct result for u_j	a `local_output` from that expert citing u_j with its preregistered terms
$s^{(5)}$	**Result integration**	r_5	that result reaches the final answer	the identifier and terms of u_j present in the `submit` answer

C.3 Expert Selection

Let R_q be the experts a query requires and A_q the distinct agents a method recruits. For a framework that registers the pool as its group, crew or worker set (§ D.1), an agent is recruited the first time the run calls it, whatever the work, and stays recruited for the rest of the run. A framework that staffs a team recruits an agent when it adds it. An agent the reference records as holding equivalent expertise counts toward R_q when the method recruits it in place of the primary expert. Over all 2,766 queries,

$$\text{Recall} = \frac{1}{N}\sum_q \frac{|A_q \cap R_q|}{|R_q|}, \qquad \text{Precision} = \frac{1}{N}\sum_q \frac{|A_q \cap R_q|}{|A_q|}.$$

Recall on its own rewards recruiting the pool. A method that calls all 51 agents reaches a recall of 1 at a precision of 7.8%, the mean of $|R_q|/51$. The two are therefore reported together, and neither enters the conditional rates below.

C.4 Stage Decisions along the Dependency Chain

For every C2 and C3 query the whole dependency graph G_q is certified before any method runs (§ B.2), and all twelve methods are scored on the same 3,664 edges of the same 1,630 queries. Table 13 gives the five predicates, each evaluated on one edge $e = u_i \to u_j$, and what each one is matched against. An edge is scored on its own, so $s_e^{(k)} \in \{0, 1\}$ records that edge alone. It enters the chain, $s_e^{(1)} = 1$, when the team holds u_i's finding and has recruited the expert holding u_j. A finding of a source unit is held once its expert has retrieved it, and a finding that is itself downstream of another edge is held once that edge has passed $s^{(4)}$, since a local solve is the only place it comes into being. A failure upstream is thus charged to the upstream edge, and the edges below it never become ready, so they count as neither passed nor failed. The four rates of § 3.3 and the dependency survival of Table 2 are

$$r_k = \frac{\sum_e s_e^{(k)}}{\sum_e s_e^{(k-1)}}, \quad k = 2, \ldots, 5, \qquad \text{and} \qquad \prod_{k=2}^{5} r_k = \frac{\sum_e s_e^{(5)}}{\sum_e s_e^{(1)}},$$

with both sums over the certified edges of the dependent queries.

Ratios of counts weight a cell by the edges it contributes, 1.30, 2.26 and 3.87 per C2 query and 1.31, 2.14 and 4.10 per C3 query at S1, S2 and S3 (Table 7), so the rates of a method lean toward the larger tasks, where most of the edges are. Two rewritings of § C.1 settle most of the ambiguity before it reaches a predicate, in that a group-addressed assignment becomes one assignment per

Table 14: How often a certified dependency is bypassed at run time. For every certified edge $u_i \to u_j$ we read the trajectory at the first retrieval that returns u_j's passage to the agent holding u_j. *Retrieved* counts the edges where that happens, and *Bypassed* the share of the method's 3,664 certified edges on which it happens before either u_i's identifier or a bound term of the edge is in any content that agent had received. *Answered on a broken edge* counts the accepted dependent queries in which such an edge never passed $s^{(2)}$ and $s^{(3)}$, against all *Answered* ones. Counts are means over five runs. The single agent has no handoff to break.

Method	Retrieved	Bypassed (%)	Answered on a broken edge	Answered
Centralized				
AutoGen	1,936.2	0.06	0.6	691.0
Magentic-One	1,805.0	0.08	0.4	652.2
Flow	1,664.8	0.05	0.0	536.2
CrewAI	1,361.4	0.29	0.4	460.4
LangGraph	1,523.6	0.10	0.4	543.2
Semi-centralized				
Anemoi	1,892.8	0.12	0.4	550.2
AgentVerse	1,431.8	0.16	0.6	475.8
MoA	677.6	0.45	1.2	260.2
MAD	1,560.8	0.33	0.4	321.4
Decentralized				
GoA	1,604.2	0.40	2.0	556.8
AgentNet	1,174.8	0.03	0.2	421.2
SelfOrg	1,300.6	0.31	0.2	368.4
Reference				
Single Agent	2,295.2	0.09	–	622.0
Twelve methods	**17,933.6**	**0.20**	**6.8**	**5,837.0**

required recipient, and a local output citing no identifier is matched on its preregistered terms alone. An edge that survives both undecided is labeled so and leaves the numerator and the denominator of its rate together, so each rate is read over the edges the trajectory settles. Table 17 gives the counts at every stage and the undecided edges beside them.

C.5 Can a Dependency Be Bypassed?

Reach keeps u_j's passage out of the eight a retrieval returns for the fixed battery of § B.2, but an agent may phrase any query it likes. A query that happened to land on u_j before u_i's finding arrived would let a method succeed while the edge it is scored on carried nothing, so we measure how often that happens from the logs themselves. For every certified edge of every dependent query and run, we read the trajectory at the first retrieval that returns u_j's source passage, or a recorded equivalent, to the agent holding u_j, and call the edge bypassed when neither u_i's identifier nor a bound term of the edge is yet in any content that agent has received, a message, an assignment or a retrieved passage. Text the agent writes itself does not count as received, so a bound term recalled from the backbone's own knowledge leaves the edge bypassed. Both checks are string matches on the log, as every stage predicate is.

Bypass is rare. Over the 43,968 certified edges the twelve methods meet per run, 3,664 each, 0.20% are reached before the predecessor's finding, from 0.03% for AgentNet to 0.45% for MoA (Table 14). Two things have to happen for an edge to be bypassed: the agent holding u_j has to search before the finding arrives at all, and a search in its own words has to land on a passage the fixed battery keeps outside the eight. The first is a property of the mechanism, and the shares are highest where agents search from q before any finding exists, as every agent does in MoA, MAD, SelfOrg and GoA and as CrewAI's planned tasks do. The second is a property of the environment, and it rises with scale, from 0.10% at S1 to 0.27% at S3, in step with the reach margin of Table 7. The single agent, with no handoff at all, bypasses 0.09% of its edges when it retrieves in the wrong order, inside the range of the teams. A bypassed edge can still pass $s^{(2)}$ and $s^{(3)}$ when the finding arrives with a later assignment. The case that matters, an accepted query in which a bypassed edge never passes them, occurs in 6.8 of the 5,837.0 dependent queries the twelve methods answer per run, 0.12% of

them. Such a query counts as answered and not in $s^{(5)}$ (Table 17), so a success that an edge did not carry stays visible in the counts.

D Compared Methods

A method enters `MAScope` through one interface. It reads the 51 public profiles, calls agents through the retrieval-backed interface of § A.2, and submits a final answer. Everything between those two points is the method's own code running the mechanism its authors published. This section gives how the twelve methods are integrated and what each decides for itself (§ D.1), the single agent reported alongside them (§ D.2), the stricter stopping rule of § 4.4 (§ D.3), and the three memory modules of § 4.5 (§ D.4).

D.1 Integration

What the integration replaces. Every framework ships with its own notion of an agent, usually a role, a system prompt and a set of tools. The integration replaces that notion with a `MAScope` expert. The role becomes the public profile, and the tool set becomes the single retrieval call into that expert's corpus. The coordinator, the message routing, the planning loop and the stopping rule run as published, so a difference between two rows of Table 2 is a difference between two mechanisms. Each method runs from its authors' public release, and the revision we ran ships with our code together with the adapter that rewrites its log into the event stream of § C.1. One framework carries state of its own across queries. AgentNet builds a routing memory as it works, and the main experiments run with that memory switched off, so all twelve rows of Table 1 stand on within-query behavior and the modules of § D.4 are the only place cross-task state enters.

What a method decides. All twelve see the same pool and the same profiles, and each applies its own rule to them. Table 15 gives that rule alongside the condition under which a run ends, and every rule below is the one the framework's paper or documentation specifies, with the notion of an agent replaced by an expert and the expert pool registered as its agents wherever a framework takes its team as given. The rules differ first in whether a framework has a staffing step at all. Five choose experts by reading the profiles: AgentVerse's recruiter, Anemoi's planner, which adds workers to its thread, Flow's allocation of subtask nodes, Magentic-One's orchestrator, which names in its task ledger the specialists its plan needs, and GoA's node sampling, in which a meta-LLM fills eight node slots from the profiles for the query and may repeat a profile it finds highly relevant (Yun et al., 2026). Eight is the most experts a task requires, and on a smaller task the meta-LLM fills the spare slots with repeats of the relevant experts, so the distinct team grows with the task, from 3.2 experts at S1 to 5.0 at S2 and 7.2 at S3, 4.8 on average. Three have no selection step of their own, so the pool is registered as the group, the crew or the worker set, and the team is the agents the mechanism actually calls: AutoGen's manager selects a speaker each turn from the 51, CrewAI's manager delegates each task of the list fixed at kickoff to one of the 51 crew members, and LangGraph's supervisor routes each step to one of the 51 workers (Barda, 2025). AgentNet's team is whoever its routers forward or split a task to, 3.1 agents on average and more on the larger tasks. MoA, MAD and SelfOrg have every agent answer the query, so with the pool registered all 51 take part, which is why the three sit at a recall of 1 and a precision of 7.8% in Table 2.

The rules differ next in how a finding moves. AutoGen broadcasts every message, so whoever speaks next reads the whole transcript. Magentic-One, Flow and CrewAI hand an agent an instruction written by the coordinator, and the agent knows what that instruction carries. LangGraph's supervisor keeps no task list. At every step it hands the message history to one worker or answers the user, deciding from the messages so far, so a finding that reads as closing the question ends the run. Anemoi's agents share a thread and address one another in it. MoA gives every agent of a layer every output of the layer before, and a final aggregator writes the answer. MAD (Du et al., 2024) has every agent answer and then revise with every other agent's answer in view, for the two rounds its authors use, and returns the answer most agents hold after the last round. GoA has the sampled agents answer and score one another, passes responses once from the higher-scored sources to the targets and back, and returns the top source's refined response or the meta-LLM's weighted synthesis (Yun et al., 2026). Its prompt asks for two or three sentences of reasoning, so what travels along an edge was short when it was written. SelfOrg has every agent answer, scores each response by the

cosine similarity of its embedding to the mean response, and in the next round lets each agent read in full the responses of its two most similar higher-scored peers, returning after at most three rounds the existing response closest to the contribution-weighted centroid (Tastan et al., 2025). Which response an agent receives thus follows its similarity to the receiver's own answer and the sender's score. AgentNet's routers forward a task, split it, or execute it, so an agent receives the task with whatever its predecessors completed (Yang et al., 2025).

Where a mechanism fixes a structure we keep the one its paper specifies. MoA runs three layers. GoA prunes by its relevance threshold, with the profiles as its model cards and the backbone as its meta-LLM. SelfOrg runs its default of two neighbours and at most three rounds, with the embedding model of its release scoring the responses. AgentVerse runs at most three recruit, decide, act and evaluate rounds. Anemoi's planner, critique and answer-finding roles run on the backbone as released, and the answer-finding agent submits once the participants approve a candidate. MAD runs its two rounds over the pool, and its vote needs no agent. Every other round, hop and retry limit keeps the framework's published default, and a run that reaches a limit submits what it holds, as a run that reaches the budget does (§ C.1).

Tools and the backbone. Every method and every agent runs on `claude-sonnet-5` (Anthropic, 2026) under the decoding settings of § C.1. Retrieval into a corpus is the only tool the environment exposes, so a framework that ships with web search, code execution or file handling runs with those tools removed and its prompts adjusted to the one call that remains. Every agent receives that call under the same declaration, a natural-language query over its own corpus returning at most eight passages with their source identifiers. Where a framework's prompt names a tool the environment does not expose, the sentence naming it is deleted, so every prompt a framework runs is a subset of the prompt it ships with. The answer format of § C.1 travels with the question, so it adds nothing to any prompt.

D.2 The Single Agent Reference

The reference removes the division of labor altogether. One agent receives the question and the 51 public profiles, reaches every corpus through the retrieval interface of § A.2, and writes the answer itself, so no finding ever has to pass from one agent to another. It is the expert of no community and runs a prompt of its own (§ H) in a plain tool loop, where each model call either retrieves from a corpus it names or writes the final answer, with no planner or reflection step added. Deciding where to look is its problem as much as a team's, and every passage it retrieves stays in the one context that writes, so what a team has to carry between agents it already holds. Its question carries the same answer format as every method's (§ C.1). It runs under the same backbone, decoding and caps as every method and uses a mean of 11.9 calls and 131k tokens per task. Its 41.0% in Table 1 is what one context holding the whole task delivers, and § F.4 reads the teams against it.

D.3 Exhaustive Collaboration

§ 4.4 replaces each method's stopping rule with a stricter one. A run continues until the method confirms that every requirement of q is met and that no recruited agent can add to the answer. The confirmation is made by the method itself, at the point where its own mechanism already decides whether to continue. A coordinator makes it where it would otherwise emit its termination message, a planner or an evaluator where it would otherwise declare the plan complete, a judge in place of its verdict, and a decentralized method, or MAD before its vote, locally in each agent, with the run ending once the last agent reports nothing to add. The judgment therefore stays inside the method, and what a run confirms is the method's own notion of completeness. Everything else is unchanged, so a method keeps planning, routing and merging as always and only stops later. The extra rounds run without the budget of § C.1, their tokens are charged to the task, and Figure 5 reports the multiplier.

D.4 Memory Modules

Three modules attach to a host method at three different points. A-MEM (Xu et al., 2025) keeps cross-task experience as structured notes on each agent with links between them, written when a task closes and read when that agent is called again. Blackboard (Han & Zhang, 2025) holds shared

Table 15: The twelve methods and the decisions each one owns. Every mechanism runs as published, and the integration replaces only the notion of an agent. *Team* is the measured mean number of distinct agents a method recruits per query, so a staffing rule can be read against the team it produces. The reference agent recruits nobody and searches all 51 corpora from one context.

Method	Team	Staffing rule	How the work is organized	A run ends when
Centralized				
AutoGen	4.4	the pool is the group; the manager selects speakers	coordinator picks the next speaker each turn; every message is visible to all	the coordinator emits its termination message
Magentic-One	4.5	the orchestrator picks from the pool at each step	task and progress ledgers, re-planned on a stall	the progress ledger marks the request satisfied
Flow	4.6	subtask nodes are allocated to experts from the pool	explicit subtask graph, updated during execution	every node of the graph is complete
CrewAI	4.4	the pool is the crew; the manager delegates	task list fixed at kickoff; the manager delegates each task once	the last planned task returns
LangGraph	4.5	the pool is the worker set; the supervisor routes	supervisor hands off to one worker at a time, or answers; no task list	the supervisor answers the user
Semi-centralized				
Anemoi	4.5	the planner adds experts to its thread	agents talk directly in the thread; the plan is revised there	the participants approve a candidate and the answer agent submits
AgentVerse	5.2	recruiter staffs a team each round	recruit, decide, act, evaluate, then repeat	the evaluator declines another round
MoA	51.0	every agent proposes; no staffing step	three layers; each agent sees every output of the layer before	the final aggregator returns
MAD	51.0	every agent debates; no staffing step	two rounds; each agent sees every other answer before revising	the round-two answers are put to a majority vote
Decentralized				
GoA	4.8	a meta-LLM fills eight node slots from the profiles, repeats allowed	peer-scored graph; responses pass source to target and back	the backward pass ends and the pooled response returns
AgentNet	3.1	agents route to one another	each agent's router forwards, splits or executes	an agent returns a final answer
SelfOrg	51.0	every agent answers; no staffing step	contribution-scored DAG rebuilt each round routes peer responses	three rounds end and the most central response is chosen
Reference				
Single Agent	–	no recruitment	one execution context with retrieval into any corpus	the agent writes its answer

working state that the team reads and writes inside one task, and it starts empty on every query. G-Memory (Zhang et al., 2025a) keeps a three-level graph of experience, tasks and interactions, and retrieves from it across tasks and across the steps of the current one.

Two of the three accumulate across tasks, so what a module holds at a given query is a function of the queries that came before it. That accumulation happens on the evaluation stream itself. Every module starts empty and grows only from the trajectories its host produces while being measured, which is the behavior the module exists to provide.

Accumulation makes the order of the queries part of the experimental condition, and two rules fix it. The order is drawn once and frozen, and the same order runs for every host, every module and all five runs, so a query meets the same amount of accumulated experience whichever combination is being measured. An entry written while solving one query also stays out of reach of the other queries of its family. Members of a family share units and so share identifiers (§ B.3), and holding

them apart keeps a sibling's answer out of the store, so the identifier acceptance turns on still arrives through retrieval. Blackboard needs neither rule, since it lives and dies inside one query.

With nothing populated in advance, every token the modules cost is spent inside a scored query. Reads, writes, the retrieval a module performs over its own store, and the extra context it adds to a call are charged to the query where they happen, which is what Figure 5 reports as a module's cost.

E Preliminary Study

§ 2 asks whether the final outcome of a MAS reflects the collaboration behind it. We ask it on two existing benchmarks, MuSiQue (Trivedi et al., 2022) for multi-hop question answering and GAIA (Mialon et al., 2023) for general assistant work, and we read the answer out of the trajectories they produce under their own mechanisms. This section gives the tasks and settings (§ E.1), the masking protocol behind the first observation (§ E.2), and the rule behind the second (§ E.3).

E.1 Tasks and Settings

Tasks come from the published splits of both benchmarks, 200 four-hop questions sampled from the MuSiQue-Ans development split and all 165 tasks of the GAIA validation split. The two benchmarks differ in where evidence comes from. MuSiQue evidence is textual and sits in passages an agent reads, while GAIA evidence comes from file processing and web retrieval, so it enters a trajectory only when an agent performs the operation that produces it. That contrast is what the first observation turns on.

Three representative methods run on both, AutoGen (Wu et al., 2024), CrewAI (Moura, 2023) and Anemoi (Ren et al., 2025). Each runs with four agents and a budget of 150k tokens per task, on one shared backbone, `claude-sonnet-5` (Anthropic, 2026) at temperature 0.3 with top-p 0.95, the settings of § C.1. The tool set is the one each benchmark presumes. On MuSiQue the agents read only the paragraph pool the question ships with, so every finding is a passage one of them retrieved. On GAIA they hold the task's attached file, a web search tool and a Python interpreter, which is the standard GAIA setup and the reason a finding there exists only once an agent has run the operation that produces it. Every configuration runs five times, and Figure 2 reports the mean.

E.2 Masking a Required Contribution

A run is scored twice from the same trajectory. We freeze every agent record at the point before aggregation, then re-aggregate under the method's own rule in two conditions. Condition (I) reads the complete record. An agent counts as required when it holds a step the task cannot be answered without, and both benchmarks state those steps themselves. MuSiQue ships the supporting paragraphs of every hop, and GAIA ships the annotator's own solution steps together with the tools each one took, so in both cases the agent that covers a listed step is required. Condition (II) reads the same record with the evidence of one required agent removed, and which agent that is comes from the query hash before any run starts, so all three methods lose the same contribution on the same task. Removal covers the passages that agent retrieved, the paraphrases of them that appear in its messages, and any direct statement of its finding inside a summary. A span goes when it states one of the masked agent's required findings, wherever it sits. Two of the authors mark the spans on the frozen record before either condition is scored, and a third resolves disagreements. The agent's participation, its other messages and every other agent's record stay as they were, so what condition (II) takes away is one contribution.

Re-aggregation is a model call, so it is not deterministic, and a small number of queries come out correct under (II) after failing under (I). Table 16 reports those queries as *Only II*, between 2.2 and 5.2 per configuration, against the 9.2 to 42.2 of *Only I* that go the other way. The drop we report is the net difference between the two conditions, so both directions enter it.

The distance between the two conditions is what the team loses by losing one required contribution. Averaged over the three methods and five runs, it is 12.5% on MuSiQue and 48.7% on GAIA (Figure 2, left). Per method the relative drop is 5.4%, 18.1% and 13.9% on MuSiQue for AutoGen, CrewAI and Anemoi, which is 3.5, 11.3 and 8.1 points of accuracy, and 35.8%, 56.7% and 53.6% on GAIA, which is 14.9, 22.4 and 18.1 points. The ordering is the same on both benchmarks and

Table 16: The paired counts behind Figure 2. We report both aggregation conditions on the same frozen trajectory, so every query carries two verdicts, with n of 200 on MuSiQue and 165 on GAIA and every value a mean over five runs. *Only I* counts the queries the complete record answers and the masked record misses, *Only II* counts the reverse, and their difference over n is the drop that Δ states as a share of full-record accuracy, taken over methods in the averaged rows. *Lost* counts the failures whose record already held a required finding the answer never adopted, and *Share* takes it over pooled failures.

	Aggregation conditions					Findings lost		
Method	**Full**	**Masked**	**Only I**	**Only II**	**Δ**	**Failures**	**Lost**	**Share**
MuSiQue								
AutoGen	64.3	60.8	9.2	2.2	−5.4	71.4	18.6	**26.1**
CrewAI	62.5	51.2	25.0	2.4	−18.1	75.0	22.2	**29.6**
Anemoi	58.2	50.1	20.0	3.8	−13.9	83.6	27.8	**33.3**
Avg.	*61.7*	*54.0*	*18.1*	*2.8*	***−12.5***	*76.7*	*22.9*	***29.8***
GAIA								
AutoGen	41.7	26.8	28.0	3.4	−35.8	96.2	38.6	**40.1**
CrewAI	39.5	17.1	42.2	5.2	−56.7	99.8	43.8	**43.9**
Anemoi	33.7	15.6	33.8	4.0	−53.6	109.4	56.6	**51.7**
Avg.	*38.3*	*19.8*	*34.7*	*4.2*	***−48.7***	*101.8*	*46.3*	***45.5***

follows how much of the record reaches the aggregator, which is a different ordering from final accuracy. AutoGen carries the whole transcript and loses least, while CrewAI's agents deliver to a manager and never see each other, and it loses most.

E.3 Locating a Discarded Finding

The second observation reads only the runs that failed under condition (I) and asks whether the answer they needed was already in the record. A failed run holds the decisive finding when some agent's output states the content the reference requires for the step the run got wrong. The match is decided against the reference's own statement of that step, and a finding counts when it supplies what the step needs, in whatever words it is phrased. The same two authors mark it independently and the third resolves disagreements, as for the masking spans. The check reads the frozen record on its own, so a finding counts whether or not the aggregator ever referred to it.

Among the failed runs, 29.8% on MuSiQue and 45.5% on GAIA hold the decisive finding (Figure 2, right). Per method the share is 26.1%, 29.6% and 33.3% on MuSiQue for AutoGen, CrewAI and Anemoi, and 40.1%, 43.9% and 51.7% on GAIA. This ordering differs from the one above. It follows how a method delivers what its agents produce, and it runs opposite to final accuracy. Anemoi summarizes a free discussion into an answer, and it both scores lowest and discards the most, while AutoGen scores highest and discards the least. The two observations separate two properties of the same record, and neither is visible in the outcome a benchmark reports.

F Full Results

The main text reports rates, and this section reports the counts they come from. § F.1 follows the 3,664 certified dependencies of the 1,630 dependent queries stage by stage, edge by edge, so every rate of Table 2 can be recomputed from adjacent columns. § F.2 and § F.3 give the two enhancement experiments in full, with the success and cost that Figure 5 plots as differences, and § F.4 reads the twelve methods against the single agent, the one system in the paper with no handoff at all.

F.1 Where Certified Dependencies Go

Table 17 gives the population at each stage, counted over the 3,664 certified edges of the 1,630 dependent queries. Every edge is scored on its own. It is ready ($s^{(1)}$) when the team holds its predecessor's finding and has recruited the expert holding its downstream unit, and it is then followed through the four capabilities. A predecessor that is a source unit is in hand once its own expert has

Table 17: Where the 3,664 certified dependencies go, edge by edge. Counts are means over five runs. Every edge $u_i \to u_j$ of every dependent query is scored on its own. *Finding held* counts the edges whose predecessor finding the team obtained at all, by retrieving u_i when it is a source unit and by solving it when it is downstream of another edge, which is the retrieval that has to succeed before any capability applies, and $s^{(1)}$ adds that the expert holding u_j was recruited and is the denominator of r_2. Each rate of Table 2 is the ratio of two adjacent columns. *Graph carried* counts the dependent queries whose every edge reached $s^{(5)}$, and *Answered* those the method solves, which sits below it because a carried graph still leaves the rest of the query to satisfy, and a query answered on a bypassed edge (Table 14) counts in *Answered* only. *Undecided* counts the edges the two rewritings of § C.1 leave undecided, which leave every column. The stage populations vary between 7.5 and 52.0 edges across the five runs, the spread of the rates they produce is in Table 2, and the spread of the outcome is given here.

	Before	Stage populations (edges)					Outcome (queries)		Left out
Method	**Finding held**	$s^{(1)}$	$s^{(2)}$	$s^{(3)}$	$s^{(4)}$	$s^{(5)}$	**Graph carried**	**Answered**	**Undecided**
Centralized									
AutoGen	2,926.4	2,618.2	2,238.2	2,010.2	1,841.8	1,700.2	781.4	691.0±8.7	15.6
Magentic-One	2,879.4	2,571.4	2,224.4	1,857.8	1,695.4	1,562.8	729.8	652.2±14.4	17.4
Flow	2,750.0	2,453.8	2,041.0	1,743.0	1,575.4	1,419.8	669.6	536.2±4.0	14.6
CrewAI	2,689.0	2,418.2	1,714.2	1,401.8	1,253.2	1,151.4	544.6	460.4±12.8	19.0
LangGraph	2,850.4	2,441.4	1,886.4	1,608.4	1,443.4	1,363.2	637.6	543.2±10.0	22.8
Semi-centralized									
Anemoi	3,008.8	2,631.0	2,420.4	1,948.8	1,813.4	1,385.0	654.6	550.2±13.1	30.4
AgentVerse	2,863.0	2,566.6	1,951.0	1,502.6	1,361.0	1,086.0	518.2	475.8±22.0	46.2
MoA	2,619.6	2,619.6	912.0	708.2	640.4	512.0	275.6	260.2±3.7	77.0
MAD	2,564.6	2,564.6	1,955.0	1,629.0	1,426.8	1,054.4	455.2	321.4±20.3	26.2
Decentralized									
GoA	2,752.0	2,520.6	2,312.6	1,659.2	1,466.0	1,279.8	613.8	556.8±9.7	59.2
AgentNet	2,161.0	1,779.4	1,559.2	1,236.4	1,118.6	960.6	464.4	421.2±15.9	28.2
SelfOrg	2,363.4	2,363.4	2,020.2	1,325.0	1,151.0	934.8	460.0	368.4±16.0	80.8

retrieved it, and one that is itself downstream of another edge is in hand once that edge has reached local solve, the only way its finding comes to exist. An upstream failure is thus scored on the upstream edge, and the edge below it never becomes ready, since there is nothing yet to orchestrate. One column stands before $s^{(1)}$ and two behind $s^{(5)}$. The first counts the edges whose predecessor finding the team holds at all. The last two return to the query: the dependent queries whose every edge reached $s^{(5)}$, and those the method answers. The second sits below the first for every method, because carrying a graph into the answer still leaves the rest of the query to be satisfied, and the few answered queries in which a bypassed edge carried nothing (§ C.5) sit inside that gap.

Reading down a column shows where each class thins out. MoA holds the finding on 2,619.6 edges and is ready on all of them, because recruiting the whole pool means the downstream expert is never missing, and it keeps 912.0 after orchestration. Most of the edges it holds are ones whose predecessor is a source unit, since a second hop needs a first hop to have been dispatched, which is what its 34.8% orchestration rarely does. AgentNet holds 2,161.0 and is ready on 1,779.4, the fewest of the twelve, and the gap between the two columns is the recruitment it forgoes. It then keeps 1,559.2 of them, so the two arrive at a similar place from opposite ends, one by recruiting everyone and then failing to dispatch, the other by recruiting few and dispatching what it has. The rates of Table 2 report the second half of each of those stories, and these counts report the first. The undecided column is largest for SelfOrg, MoA and GoA, the three methods whose native constructs the adapter has the most to unpack, and stays below 2.3% of the edges for every method.

F.2 Exhaustive Collaboration

Table 18 gives the native and exhaustive success of every method alongside the token ratio between the two rules. The ratio runs from 1.28 to 3.28. The three cheapest continuations are CrewAI, LangGraph and MoA, which never revise a plan, since another round re-executes what each already holds, and SelfOrg and MAD, whose rounds are a fixed pool answering again, come next. The

Table 18: Exhaustive collaboration in full. We report each method under its native protocol and under the requirement to collaborate until it confirms completion, as means over five runs on all 2,766 queries, with cost in thousands of tokens per task. Δ is the difference in success rate, and the two conditions share a seed within a run, so its spread is that of the paired difference. *Mechanism* is how a method handles a finding produced at run time, the grouping Figure 5 plots by, and the token ratio is the exhaustive cost over the native one.

		Successes			Cost (k tokens)		
Method	**Mechanism**	**Native**	**Exhaustive**	Δ	**Native**	**Exhaustive**	$\times$
Centralized							
AutoGen	Re-plans	1,442.2	1,487.4	**1.63**$^{\pm 0.40}$	170.9	460.4	2.69
Magentic-One	Re-plans	1,388.4	1,420.2	**1.15**$^{\pm 0.31}$	208.4	358.9	1.72
Flow	Re-plans	1,289.0	1,323.6	**1.25**$^{\pm 0.17}$	126.2	363.7	2.88
CrewAI	Plan not revised	1,241.0	1,233.6	$-$**0.27**$^{\pm 0.12}$	118.8	175.4	1.48
LangGraph	Plan not revised	1,169.0	1,168.4	$-$**0.02**$^{\pm 0.23}$	127.4	173.9	1.36
Semi-centralized							
Anemoi	Re-plans	1,224.6	1,233.6	**0.33**$^{\pm 0.29}$	118.3	303.7	2.57
AgentVerse	Ruling merges	1,160.0	1,143.6	$-$**0.59**$^{\pm 0.18}$	165.2	378.3	2.29
MoA	Plan not revised	884.2	889.2	**0.18**$^{\pm 0.18}$	1,044.3	1,339.4	1.28
MAD	Ruling merges	814.4	793.2	$-$**0.77**$^{\pm 0.21}$	961.5	1,560.7	1.62
Decentralized							
GoA	Re-plans	1,276.8	1,297.4	**0.74**$^{\pm 0.13}$	208.2	406.6	1.95
AgentNet	Re-plans	933.4	961.2	**1.01**$^{\pm 0.24}$	87.5	286.6	3.28
SelfOrg	Re-plans	892.2	914.0	**0.79**$^{\pm 0.26}$	323.6	490.9	1.52

success the extra rounds buy runs from -0.77 to $+1.63$ points. All seven methods that let findings re-enter planning land on the positive side, and the four that lose ground, CrewAI, LangGraph, AgentVerse and MAD, either never revise a plan or settle the answer by a ruling, a judge's or a vote's.

The run-to-run spread says which of these effects hold, since a sign alone does not separate an effect from noise. Nine of the twelve clear twice their own spread, and they divide without exception. Six are gains, every one from a method that lets findings re-enter planning: Flow, AutoGen, Magentic-One, AgentNet, SelfOrg and GoA. Three are losses, every one from a mechanism that cannot act on a finding at all: CrewAI with its task list fixed at kickoff, AgentVerse with its evaluator's ruling, and MAD with its vote, where another round is another chance for the majority to outvote the one agent holding a correct result. The three that stay inside two spreads are LangGraph, MoA and Anemoi. LangGraph and MoA have no plan to revise, so their change is expected to be nothing, since LangGraph's supervisor confirms completion from the same messages that ended its run and MoA's extra layer reads the same proposals. Anemoi is the one such method we cannot distinguish from zero, and Table 2 says why. Its orchestration is already the highest of the twelve at 92.0%, and its integration the second lowest at 76.4%. The extra rounds do produce further findings for it, and the merge then discards them.

F.3 Memory Modules

Table 19 gives all twelve host and module pairings with the success and the cost of each. The host row repeats in every block, so a pairing can be read against its own baseline and against the other two modules on the same host. Cost moves in a narrow band, between -4.5% and $+11.9\%$, while success moves between -1.13 and $+3.22$ points, so the whole spread in success comes from which module supplies what the host lacks, at close to the same cost throughout. The four largest gains are the four pairings that place a module at the level of the host's weakest stage, A-MEM on AgentNet, G-Memory on AutoGen and on GoA, and Blackboard on MoA, and each of them clears its own run-to-run spread by between 7.7 and 15.1 times. The largest loss clears it by 6.5, GoA with A-MEM at -1.13, where per-agent notes attach to a method whose agents already share a graph and add context to every call without adding reach. Cost tracks none of this. A-MEM saves AgentNet

Table 19: Memory modules in full. Each host appears with its own run and with the three modules attached to it, all means over five runs on 2,766 queries. Δ is the success rate against that host's own baseline in points, and the last column the change in per-task tokens. Both deltas carry their spread over the five runs, taken on the paired difference, since a module and its baseline share a seed within a run. The modules accumulate on the evaluation stream under the rules of § D.4.

		Success			Cost	
Host	**Module**	**Successes**	**Rate**	**Δ**	**k tokens**	**Δ%**
AutoGen	*none*	1,442.2	52.14	–	170.9	–
	A-MEM	1,459.6	52.77	**0.63**$\pm$0.37	178.8	4.6$\pm$0.9
	Blackboard	1,484.2	53.66	**1.52**$\pm$0.38	175.2	2.5$\pm$1.1
	G-Memory	1,524.0	55.10	**2.96**$\pm$0.20	184.4	7.9$\pm$1.8
GoA	*none*	1,276.8	46.16	–	208.2	–
	A-MEM	1,245.6	45.03	−**1.13**$\pm$0.17	217.0	4.2$\pm$1.0
	Blackboard	1,266.6	45.79	−**0.37**$\pm$0.18	214.3	2.9$\pm$0.8
	G-Memory	1,354.0	48.95	**2.79**$\pm$0.36	220.5	5.9$\pm$0.8
AgentNet	*none*	933.4	33.75	–	87.5	–
	A-MEM	1,022.4	36.96	**3.22**$\pm$0.22	83.6	−4.5$\pm$0.8
	Blackboard	971.4	35.12	**1.37**$\pm$0.21	91.5	4.6$\pm$1.1
	G-Memory	950.2	34.35	**0.61**$\pm$0.24	97.9	11.9$\pm$1.1
MoA	*none*	884.2	31.97	–	1,044.3	–
	A-MEM	898.2	32.47	**0.51**$\pm$0.17	1,058.1	1.3$\pm$0.9
	Blackboard	929.8	33.62	**1.65**$\pm$0.14	1,076.2	3.1$\pm$0.6
	G-Memory	873.4	31.58	−**0.39**$\pm$0.23	1,083.0	3.7$\pm$0.7

4.5% of its tokens while G-Memory costs AutoGen 7.9% and GoA 5.9%, so what a module delivers follows from the level it attaches at.

F.4 The No-Collaboration Reference

The single agent of § D.2 solves every task without a handoff, which makes it the control for what dividing the work costs and what it buys. Dependency costs it little. Its success falls by 15.1% from C1 to C2 and C3, against a mean of 48.6% over the twelve methods, and every dependent cell of Table 1 sits within 11% to 31% of its parallel cell at the same scale. The reason is the one property that defines the paradigm, that the finding a downstream unit needs is produced in the same context that goes on to retrieve that unit, so nothing has to be carried anywhere, and what remains of the cost is finding the order of retrievals and, in C3, reading off the finding which community to query next. Teams meet the same tasks with a handoff between the two steps, and lose more than three times as much. The difference between the two drops, 33.5 points of relative success, is the part the handoffs add, and it is what the four capabilities of Table 2 locate stage by stage.

Read cell by cell, the two paradigms trade places. On parallel tasks eleven of the twelve methods beat the reference, since dividing independent work across agents relieves one context of retrieving every community itself. On dependent tasks only two do, AutoGen at 42.4% and Magentic-One at 40.0% against the reference's 38.2%, and the other ten fall below an agent working alone, MoA and MAD by about half. The two that stay ahead hold the highest dependency survival of the twelve, 64.9% and 60.8% (Table 2), which is to say they lose the least in the handoff.

Scale costs the reference more than it costs a team. It falls by 44.4% from S1 to S3 against 38.2% for the mean team, and inside C1, where no finding has to travel at all, it trails the mean team by 16.1% at S1 and by 31.7% at S3. One context holding every community of a task pays more the more communities the task needs, and dividing the work across agents is what relieves it. Division of labor thus buys back scale and pays for it in handoffs, and which effect dominates depends on the structure of the task.

Read against the reference, a method's dependency drop is a net effect, the relief that dividing the work brings minus what its handoffs drop. In Table 1 the excess of a method's drop over the

reference's runs from 20.8 points for AutoGen to 55.8 for MoA, and across the twelve methods the further a method's drop exceeds the reference's, the lower its dependency survival in Table 2, with a Pearson correlation of -0.89 and a Spearman correlation of -0.87, or -0.78 and -0.83 with MoA set aside. The methods that fall furthest behind the reference are the ones whose handoffs drop the most dependencies, which is the loss the four capabilities locate. The reference is a common yardstick for all twelve methods and a counterfactual for none of them.

G Qualitative Examples

The three mechanisms of § 4.3 are statements about what a trajectory contains, and this section shows the trajectories. § G.1 gives one task in full, so every excerpt that follows can be read against what the reference asks for. § G.2 to § G.4 then pair a run that loses one of that task's preregistered dependencies with a run that carries it, on the same edge, so the two boxes differ only in the mechanism that handles the finding. Excerpts are the events of Table 10 with their content abridged, and identifiers, the terms the reference preregisters and the order of events are as logged.

G.1 One Task in Full

The example is a service-diagnosis query from C2 at scale S3, the cell of Table 8 that holds 236 queries over six to eight required experts. This one requires six. Two of them the question reaches on its own, and the other four are scoped only once the findings upstream of them are in hand. Certification (§ B.2) returns six edges between them, and all six are preregistered. Figure 8 gives the task in full, with the graph the runs below are scored against.

G.2 Planning Rigidity

CrewAI runs a task list fixed at kickoff, so a finding produced at run time reaches the manager with nowhere to go. In the first box the Security agent has already been called for a planned task of its own, which is what makes it recruited (§ C.3). The finding then arrives, the next planned task is issued to someone else, and the work the finding implies is never issued at all. The second box is AutoGen on the same edge, where the manager reads the same finding and selects the expert it points to.

CrewAI, the finding arrives and the plan continues

`assign` **manager → Security** *plan item 1 of 4, before any finding*
"List the TLS settings the proxy listener enforces." carried units: none

Security answers from its corpus and is thereby recruited. The task names no remedy, since the plan was written before the diagnosis existed.

`local_output` **Network Engineering** `MS-9TJ5XB2WQNMF` *plan item 2 of 4 returns*
Reachability and response speed are different tests. The direct check succeeds while the proxied path stalls for 40 s against a 30 s read timeout, so the path is not the fault and the remedy has to change timing.

`assign` **manager → Database Administrators** *plan item 3 of 4*
"Investigate the 12 s lock wait on the reporting query." carried units: `MS-4KQ7T2VB9XHD`

The remaining turns issue plan item 4 and the final assembly. No `assign` *naming the TLS remedy is issued, and Security, on the team since plan item 1, is never asked for it.*

`submit` **manager**
The answer carries `MS-4KQ7T2VB9XHD`, `MS-9TJ5XB2WQNMF` and `MS-Z5RB8KM2XHDQ`, and states a timeout increase with no verification constraint.

Stage decision. $s^{(1)}$ holds, the finding is in hand and Security has been called. $s^{(2)}$ fails, no assignment later than the finding carries the work it implies.

A C2 query at scale S3, with six required experts

The C2 / S3 cell contains 236 queries over six to eight required experts; this task requires six.

Query q

"A reporting service started timing out after we moved it behind a reverse proxy. Direct access to the service still works. The proxy logs a gateway timeout at 30 s, the upstream goes quiet for about 40 s before that, the client retries three times, and one report query waits about 12 s on a lock. Explain the cause and give a remedy that covers the proxy, the TLS layer, the retries and the database together, and that we can put into production without a second outage."

Required units U

E1 · Server Fault `MS-4KQ7T2VB9XHD`
The gateway timeout is raised by the proxy: 40 s of upstream silence exceeds the 30 s read timeout set by proxy_read_timeout. *Preregistered terms:* proxy-side timeout, 40 s > 30 s.

E2 · Network Engineering `MS-9TJ5XB2WQNMF`
Reachability and response speed are different tests. The upstream_response_time field measures response timing; a working direct path leaves the proxied one untested, so the remedy is a timing change. *Preregistered terms:* reachability ≠ response speed, timing remedy.

E3 · Security `MS-H6PD3RK8ZVCM`
Certificate verification stays on. The source raises the read timeout to 45 s, above the observed upstream silence, and leaves the handshake budget unchanged. *Preregistered terms:* verification preserved, timeout raised to 45 s; proxy_read_timeout and upstream_response_time from E1 and E2.

E4 · Stack Overflow `MS-W2NC7FQ4JB6T`
Retries are bounded with proxy_next_upstream_tries so that attempts times timeout stays inside the new budget, and each retry repeats the handshake E3 keeps. *Preregistered terms:* bounded retries, retry cost includes the handshake; 45 s budget from E3.

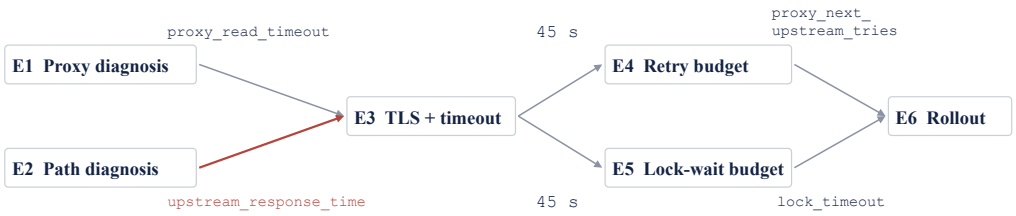
E5 · Database Administrators `MS-Z5RB8KM2XHDQ`
The 12 s lock wait belongs to a blocking session that must be identified. The source names lock_timeout for the lock-wait limit; the wait must fit inside the budget the proxy ends up with. *Preregistered terms:* blocking session identified, 12 s inside the budget; 45 s budget from E3.

E6 · DevOps `MS-B7XQ4MZ2HPKT`
The new timeout reaches production by draining in-flight connections and reloading the proxy on one node at a time, so applying the remedy does not itself drop requests. *Preregistered terms:* drain before reload, staged rollout; proxy_next_upstream_tries and lock_timeout from E4 and E5.

Dependency graph G_q

Each arrow: a term the head must state that only the tail's source gives; without the tail's finding, BM25 cannot reach the head's passage.

Reachable from the question *Scoped only once their prerequisite findings are available*



All six edges are certified and scored one by one, and an edge waits until its predecessor finding exists.
E1 and E2 establish timing on the proxy side. E3 fixes the budgets for E4 and E5, and E6 waits until that budget is final.
E2 → E3 is the edge examined by the three trajectory pairs in G.2–G.4.
E1 → E5 is not an edge: nothing E5 must state comes only from E1, and its budget term comes from E3.

Acceptance V

The answer carries all six identifiers and states the preregistered terms of each unit, including its incoming bound terms. One equivalence is recorded: naming the blocking session by query or by session identifier satisfies E5. No alternative route is recorded, so every unit must be retrieved from its own corpus.

Figure 8: One task in full. The query a method receives, the six information units a solution requires with the community holding each one, its source identifier and its preregistered terms, the certified dependency graph G_q, and the acceptance criteria. An arrow is a certified dependency, whose tail unit is the one source of a term the head unit must state, and without its finding the head's passage stays out of retrieval reach (§ B.2). All six edges are scored one by one, and the excerpts of § G.2 to § G.4 read the edge in red.

AutoGen, the same finding reaches the speaker selection

`local_output` **Network Engineering** `MS-9TJ5XB2WQNMF`
Identical content, same position in the run.

`message` **coordinator, speaker selection**
"The path is clear, so the fix is timing on the proxy. Whether the budget can be raised depends on what the TLS layer requires. Next speaker: Security."

`assign` **coordinator → Security**
"Give a timing-only remedy for the 30 s read timeout against 40 s of upstream silence, and state what has to hold for certificate verification." carried units: `MS-4KQ7T2VB9XHD`, `MS-9TJ5XB2WQNMF`

Stage decision. $s^{(2)}$ holds, the assignment follows the finding and carries the objective of E3.

G.3 Semantic Compression

GoA's agents answer in the two or three sentences its prompt asks for, and a node's input in message passing is its neighbours' responses (§ D.1), so what travels along an edge was compressed when it was written. The first box puts the passage the Network Engineering agent retrieved beside the response it wrote from it and the prompt in which that response reaches the Security agent, with the constraint terms that never left the passage marked. The second box is AutoGen on the same edge, where the transcript reaches the next agent unchanged.

GoA, the finding is written short and travels short

`retrieve` **Network Engineering** `MS-9TJ5XB2WQNMF`
"…the direct check succeeds while the proxied path stalls for **40 s** of upstream silence against a **30 s** read timeout, so a remedy has to fit inside that budget or raise it deliberately."

`local_output` **Network Engineering** *initial response, brief reasoning by prompt*
"Reachability and speed are different tests; the direct path works and the proxied one stalls, so the fault is timing on the proxy, not the path." cites `MS-9TJ5XB2WQNMF`

Never written into the response: the 40 s and 30 s figures and the constraint that a remedy must fit or raise that budget.

`assign` **source-to-target → Security**
"Refine your answer by considering other models' responses. Network Engineering (high relevance): …the fault is timing on the proxy, not the path." carried units: `MS-9TJ5XB2WQNMF`

Stage decision. $s^{(2)}$ holds, the response reaches the Security node. $s^{(3)}$ fails, the identifier arrives but the constraint terms of E2 do not, and nothing else the node can read carries them.

AutoGen, the same finding delivered intact

`assign` **coordinator → Security**
"Give a timing-only remedy for the 30 s read timeout against 40 s of upstream silence, and state what has to hold for certificate verification." carried units: `MS-4KQ7T2VB9XHD`, `MS-9TJ5XB2WQNMF`

The assignment restates both figures and the budget constraint, and the transcript of the round remains readable to the recipient.

Stage decision. $s^{(3)}$ holds, the identifier and the constraint terms of E2 are present in the carried units.

G.4 Adjudicative Aggregation

MAD closes with a majority vote over the answers its agents hold after the last round, so a correct local result enters the final step as one answer among fifty-one, and it is kept only if enough agents adopted it when they revised. The first box is the vote's record on a query where the correct result was on the table, held by the one agent whose corpus contains it, and the answer most agents held was returned. The second box is LangGraph on the same query, where the agents' results are assembled into one answer and the correct one keeps its place.

MAD, a correct result loses the vote

`local_output` **Security** `MS-H6PD3RK8ZVCM` *round 1*
"Keep certificate verification. Raise the read timeout to 45 s, above the observed 40 s of silence, and leave the handshake budget where it is."

`assign` **round 2 → every agent**
"Using the solutions from other agents as additional information, give an updated answer." carried units: `MS-4KQ7T2VB9XHD`, `MS-9TJ5XB2WQNMF`, `MS-H6PD3RK8ZVCM`, . . .

`local_output` **Security** *round 2, unchanged*
"Raise the read timeout to 45 s and keep verification as it is." cites `MS-H6PD3RK8ZVCM`

`local_output` **Server Fault, Network Engineering, and 37 others** *round 2*
"Relax strict verification so the handshake finishes inside the current 30 s." cites nothing for that step

`aggregate` **majority vote over the round-2 answers**
39 of 51 answers relax verification, 3 raise the timeout and keep it, and 9 propose something else.

`submit` **the majority answer**
The answer carries `MS-4KQ7T2VB9XHD` and `MS-9TJ5XB2WQNMF`. `MS-H6PD3RK8ZVCM` is absent, and the answer contradicts the term E3 preregisters.

Stage decision. $s^{(4)}$ holds, the expert produced the correct result. $s^{(5)}$ fails, the result never reached the answer.

LangGraph, the same result assembled into the answer

`local_output` **Security** `MS-H6PD3RK8ZVCM`
Identical content.

`aggregate` **supervisor, assembly step**
"Proxy: raise the read timeout to 45 s `MS-4KQ7T2VB9XHD`. TLS: verification unchanged `MS-H6PD3RK8ZVCM`. Retries: bounded at three within the 45 s budget `MS-W2NC7FQ4JB6T`. Database: the blocking session is identified and its 12 s wait fits `MS-Z5RB8KM2XHDQ`. Rollout: drain and reload one node at a time `MS-B7XQ4MZ2HPKT`."

The supervisor writes each worker's result into its own slot, so no result competes with another for a place in the answer.

Stage decision. $s^{(5)}$ holds, the identifier and the terms of E3 are present in the submitted answer.

The three losses sit at three different stages of the same edge. In each one the Security agent was on the team and never asked, was asked without what it needed, or answered correctly and was outvoted, so the expertise was present throughout. Each is charged to this edge at the stage where it happens, and any one of them costs the query its answer. That is the distinction Table 17 counts and Table 2 rates.

H PROMPTS

`MAScope` adds three prompts and one line of its own. The expert prompt, the answer format and the single-agent prompt reach scored runs. The expert prompt is what every one of the 51 expert agents receives, where the corpus boundary and the identifier requirement of § A.2 become instructions an agent can follow. The answer format is one line appended to every question q, for every method and for the single agent alike, so whichever agent writes the final answer is told to keep the identifiers, whatever framework it runs in. The single-agent prompt runs only the no-collaboration reference of § D.2. The third prompt, a composer that is not printed, runs at construction and drafts a composed question from a group of units, which the author checklist of § B.1 keeps or discards. No prompt decides a dependency or an expert-discovery relationship, since both are fixed by the deterministic rules of § B.2. Each method keeps its own prompts as its authors wrote them (§ D.1) and each memory module keeps its own, so these are the whole of the text `MAScope` adds. Acceptance itself uses no prompt, since every entry of V is settled by string against the reference (§ C.2). The answer format tells a method what an answer has to carry, and only the string match reads it.

The expert agent

```
You are the expert agent for <site>, one of 51 experts a method may
recruit for a user's question.  Your scope:  <expertise scope>.

You hold a private corpus of accepted question-and-answer pairs from
<site>.  It is the only knowledge you may use, and it is the only corpus
you can reach.

TOOL
 retrieve(query) -> up to 8 passages, each returned as
             [<identifier>] <question> / <accepted answer>
 The query is natural language.  Call it as often as the work needs.

WHEN YOU ARE GIVEN WORK
 1.  Retrieve from your corpus what the work needs.
 2.  Answer the work you were given, and nothing beyond it.
 3.  State the identifier of every passage your answer rests on, in
   the form [MS-XXXXXXXXXXX]. The team can use only what
   arrives with its identifier.
 4.  If your corpus does not cover the work, say so and stop.
```

The answer format, appended to every question

```
Your final answer must state the identifier of every passage it rests
on, in the form [MS-XXXXXXXXXXX], exactly as the passage was returned.
```

The single agent

```
You are answering a user's question on your own.  You may search the
question-and-answer corpora of 51 Stack Exchange communities, whose
public profiles are listed below.  These corpora are the only knowledge
you may use.

QUESTION   <q, with the answer format every method receives>
PROFILES   <site name, expertise scope and topic tags, 51 of them>

TOOL
 retrieve(site, query) -> up to 8 passages from that site's corpus,
    each returned as [<identifier>] <question> / <accepted answer>
 Name one site per call.  Call it as often as the question needs.

Retrieve what the question needs, then write one answer.
```

The single-agent prompt carries no plan and no advice on how to use one finding to find the next. Every method brings its own mechanism for that, and the reference is the case with none, so whatever it does with a finding it does with one context and its backbone alone. It is told about identifiers only through the answer format, as a team's final writer is.

Construction reuses the agent prompt wherever it solves. A unit is solved for local solvability (§ B.3) by giving its own expert agent its work together with the findings of its certified predecessors, and the subsumption test of § B.1 is the same call with another unit's passages supplied in place of retrieval. Certification itself calls no model. All three prompts and the answer format ship with the code, together with the profiles and the corpora they read from, so every model call `MAScope` makes on its own behalf can be replayed.

I Limitations and Data

I.1 What the Findings Cover

Every number we report on `MAScope` comes from one family of tasks, one backbone and one set of method configurations. The preliminary study of § E is the one exception and runs on two existing benchmarks. The tasks are information-solving, where a contribution is a finding retrieved from a corpus and adoption means carrying that finding into an answer. The backbone is `claude-sonnet-5` throughout, for the twelve methods, the 51 agents, the construction-time

solves and the preliminary runs alike. The configurations are each method's published mechanism, with the expert pool registered as its agents wherever a framework takes its team as given (§ D.1). A capability rate read here therefore describes one method, at this scale of task, on this backbone, in its published shape.

The five runs differ only in the sampling seed (§ C.1), so the spread we report is the run-to-run variation of one configuration. The three mechanisms of § 4.3 are properties of what a mechanism does with a finding produced at run time, and that is the level at which we state them.

I.2 What the Environment Leaves Out

A contribution in `MAScope` is a piece of retrieved knowledge, and it is complete the moment an agent states it with its identifier. Collaboration also happens over things that change while the team works, and three kinds of it sit outside this environment. Agents write no code, so no contribution is a patch that a later agent has to build on. They share no tool state, so what one agent does to the world never constrains what another can do next. And nothing carries across queries by default, so a team never accumulates a way of working, which § D.4 adds as an option and measures separately. Each of these gives a contribution a lifetime longer than one message, and following one would need a reference that tracks state as well as text. Extending the same tracing to them is the next step.

I.3 Data and Release

The corpora derive from public Stack Exchange posts under CC BY-SA. We redistribute them under the same license, keep the attribution it requires, and strip user names, profile links and every other author-identifying field before a corpus enters the environment (§ A.3). The release ships the 51 corpora with their identifiers, the 2,766 queries with their references, family labels and certified dependency graphs, the retrieval interface, the twelve adapters and the evaluator, so any method can be run and scored under the protocol of § C.

Releasing the corpora publishes the identifiers with them, which bounds the memorization argument of § A.3. The argument holds for any backbone trained before the release, since the identifiers did not exist earlier. A backbone trained afterwards could have read them, and that is why we keep the minting key. With the key the evaluator re-mints the identifier set of any released task, so a later model can be scored against identifiers it cannot have seen, on the same corpora and the same references. The code is released under the MIT license and the derived corpora under CC BY-SA, both at the anonymized repository linked from the abstract.