TANGO: TASK-ADAPTIVE AGENT ORCHESTRATION ACROSS THE MULTI-AGENT LIFECYCLE

Anonymous authors
Paper under double-blind review

ABSTRACT

Multi-agent systems can improve complex reasoning in LLMs, but many frameworks predetermine one or more stages of the orchestration lifecycle: worker construction, communication, or final aggregation. We present TANGO, a task-adaptive framework that manages this orchestration lifecycle from creation to coordination and conclusion. For each task, the meta-agent creates workers with complementary reasoning strategies and tool configurations. Each worker maintains a structured memory state that records its progress, hypotheses, unresolved questions, and supporting evidence. At periodic checkpoints, the meta-agent coordinates the workers by constructing a sparse directed communication topology and selecting the information to route. It then concludes the process by reasoning over terminal answers and structured memories to synthesize the final answer. We evaluate TANGO with Gemini-2.5-Flash across general agentic, scientific, mathematical, and code-generation tasks using GAIA, GPQA-Diamond, Omni-MATH, and HumanEval. We report mean accuracy across three runs, Pass@3, and a four-benchmark macro-average for both metrics. TANGO attains the highest observed macro-averages in our comparison. Our method achieves the highest scores on GAIA, Omni-MATH, and GPQA-Diamond, while being competitive on HumanEval. We further investigate how meta-agent control across the multi-agent lifecycle affects performance, how these components interact, and how structured memory enables the meta-agent to orchestrate workers more effectively. The code is available here.

1 INTRODUCTION

Multi-agent orchestration can improve LLM problem-solving by designing specialized workers (Hu et al., 2025; Zhang et al., 2025b; Zhou et al., 2026), structuring communication between them (Lu et al., 2026; Yun et al., 2026), and improving final-answer aggregation (Du et al., 2023; Li et al., 2024; Smit et al., 2024). Their effectiveness, however, depends on coupled decisions across the orchestration lifecycle: which workers are created, how they exchange information during execution, and how their outputs are combined into a final answer. Yet many existing methods concentrate adaptation in one stage while leaving other decisions fixed across diverse tasks. Some emphasize task-specific worker or workflow construction (Chen et al., 2024a; Hu et al., 2025), others adapt communication over a fixed worker pool (Liu et al., 2024; Lu et al., 2026; Zhang et al., 2025a), and others improve aggregation over outputs produced by a predetermined execution process (Chen et al., 2023b; Wang et al., 2024a; Chen et al., 2024b). This separation can create a mismatch between the orchestration policy and the task. A fixed worker portfolio may contain irrelevant or redundant roles, while a fixed communication topology may fail to adapt to a task. Effective orchestration should therefore adapt to both the task and the evolving worker states, motivating a framework that manages worker creation, coordination, and conclusion together.

To enable such adaptation throughout execution, we ask: *how can a meta-agent manage the lifecycle of a multi-agent system?* We organize this lifecycle into three stages: *Create*, which determines what workers should exist for the task, *Coordinate*, which determines what information should be exchanged and between whom, and *Conclude*, which determines how terminal answers and worker states should be combined. This decomposition provides a unified structure for adapting worker construction, information exchange, and final synthesis to each task.

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.

We introduce TANGO, a framework that instantiates these three stages through separate role-conditioned calls to a shared meta-agent backbone. The meta-author first designs a task-conditioned set of workers with complementary tool-use capabilities and reasoning strategies. Each worker solves the task independently and maintains a compact, evolving structured memory, which is a running summary of its progress, hypotheses, open questions, and supporting evidence. During execution, the meta-router inspects these states and constructs a sparse, directed communication topology that routes selected information to workers that may benefit from it. Once all the worker agents have finished execution, the meta-synthesizer concludes by comparing the workers' terminal answers and recorded states and generates the final answer. We evaluate how well our meta-agent-guided orchestration performs across general agentic, scientific, mathematical, and code-generation tasks using four benchmarks: GAIA, GPQA-Diamond, Omni-MATH, and HumanEval. TANGO attains the highest observed four-benchmark A@3 and P@3 macro-averages, has the highest observed A@3 on GAIA, GPQA-Diamond, and Omni-MATH, and obtains the highest observed P@3 on all four benchmarks. These results demonstrate that a shared meta-agent can adaptively manage the three stages of the multi-agent lifecycle. Additional experiments compare configurations that assign the meta-agent control over the different orchestration stages and analyze communication policy, checkpoint frequency, worker scaling, and final synthesis. Our contributions are as follows.

1. We formulate task-adaptive multi-agent orchestration as a coupled *Create–Coordinate–Conclude* lifecycle and study whether a meta-agent can manage all three stages and design a task-adaptive multi-agent system.
2. We introduce TANGO, a training-free framework in which a meta-agent creates dynamically authored tool-use workers, coordinates directed information routing via reasoning over structured memories, and concludes with evidence-backed synthesis.
3. We evaluate TANGO on GAIA, GPQA-Diamond, Omni-MATH, and HumanEval, and analyze authoring, memory representation, communication policy, checkpoint frequency, worker count, and final synthesis on GAIA and Omni-MATH to understand the capability of the meta-agent in managing each lifecycle component.

2 Related Work

Inference-time reasoning: Additional inference-time reasoning has been used to improve LLM performance in multiple domains (Snell et al., 2024; Madaan et al., 2023; Welleck et al., 2023; Yang et al., 2022; Ji et al., 2026; Brown et al., 2024). Early methods increased inference-time computation through explicit reasoning and planning (Wei et al., 2023; Kojima et al., 2023; Wang et al., 2023a), iterative feedback and refinement (Madaan et al., 2023; Shinn et al., 2023; Ji et al., 2023), or structured search over thoughts (Yao et al., 2023; Besta et al., 2024). Recent proprietary and open-source reasoning models further learn chain-of-thought reasoning and self-reflection behaviors through reinforcement learning (Jaech et al., 2024; Comanici et al., 2025; Guo et al., 2025).

Many other approaches generate multiple reasoning paths (Wang et al., 2023b; Chen et al., 2023b) or select outputs using learned outcome or process verifiers (Cobbe et al., 2021; Lightman et al., 2023; Wang et al., 2024b). Mixture-of-Agents (Wang et al., 2024a) and Graph-of-Agents (Yun et al., 2026) extend this idea by sharing answers between LLMs. Symbolic-MoE (Chen et al., 2026) routes queries among heterogeneous LLMs, whereas other approaches employ different agents with the same LLM (Zhang et al., 2024b; Chen et al., 2025a; Li et al., 2025b; Chen et al., 2025b). However, these methods generally do not adapt communication within an agent's ongoing trajectory (Cemri et al., 2025). Agents typically complete a response independently, after which their outputs are aggregated, shared, or used to initialize another refinement round. Consequently, collaboration takes place primarily after trajectories are finished rather than during the generation process.

Multi-agent orchestration: A new line of research studies how to design these systems to improve LLM task-solving capabilities. Early approaches such as CAMEL (Li et al., 2023) and MetaGPT (Hong et al., 2024) assign role-specific agents and prescribe structured communication protocols, while AutoGen (Wu et al., 2023) supports customizable interaction patterns. AgentVerse (Chen et al., 2023a) explores generated roles and iterative collaboration. Another line of work uses a central orchestrator that delegates tasks to specialized subagents (Zhang et al., 2024c; Alzu'bi et al., 2026; Ruan et al., 2026). Other multi-agent approaches improve the final response through discussion, critique, or debate between multiple agents working on the same task (Du et al., 2023; Liang et al.,

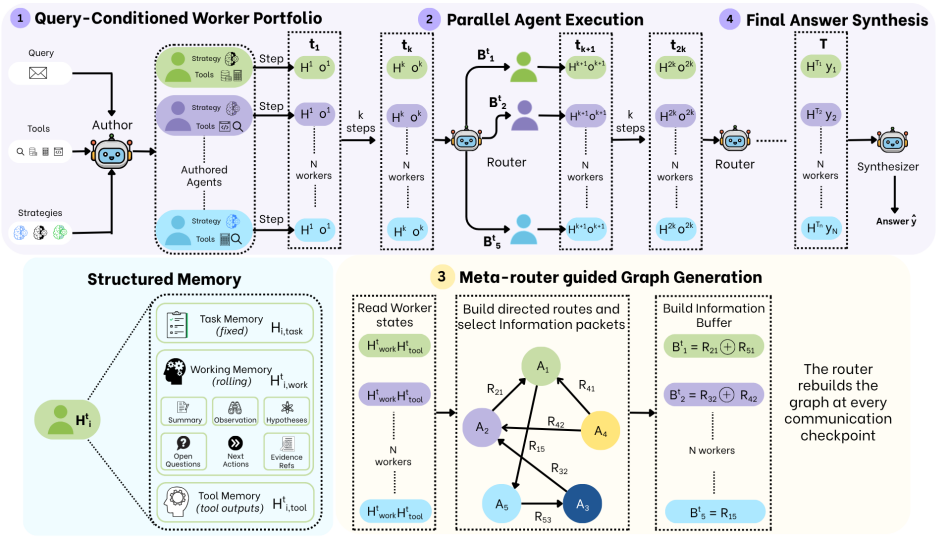


Figure 1: Overview of our method. (1) The meta-author designs a task-adaptive portfolio of workers. (2) The workers execute their roles in parallel. (3) At each checkpoint, the meta-router selects directed information transfers between workers. (4) Finally, the meta-synthesizer integrates the workers' terminal responses and structured states to generate the final answer.

2024; Chen et al., 2024b; Wang et al., 2024c), while others combine agent responses using different aggregation methods (Jiang et al., 2023; Sun et al., 2024; Wang et al., 2024a). These methods demonstrate that the benefits of agent scaling depend not only on the number of agents but also on the communication topology, agent diversity, and aggregation method.

Recent work has increasingly focused on making agent collaboration adaptive. DyLAN (Liu et al., 2024) selects agents based on an importance score over a temporal network, while GPTSwarm (Zhuge et al., 2024), G-Designer (Zhang et al., 2025a), and AgentPrune (Zhang et al., 2024a) optimize graph connectivity among agents. These methods generally select or connect agents from a fixed pool of roles, and many require task-oriented training or an offline architecture search. DyTopo (Lu et al., 2026), a parallel work, updates the communication topology as the task progresses but does not synthesize task-specific workers. AutoAgents (Chen et al., 2024a) and ADAS (Hu et al., 2025) automate agent design through task-specific team generation and validation-guided workflow search, respectively. However, while these methods make agent design or collaboration adaptive, they typically leave at least one other stage of the orchestration lifecycle predetermined. TANGO instead studies task-conditioned control over worker creation, information exchange, and final synthesis within one framework.

3 Methodology

The overview of our method is shown in Figure 1, and the complete procedure is summarized in Algorithm 1.

3.1 Preliminaries and Notation

Given a user query q, an agentic system generates an answer $\hat{y}$. The system consists of a meta-agent $\mathcal{M}$ and a set of N worker agents $\mathcal{A} = \{a_1, \ldots, a_N\}$. Let $\mathcal{T}$ denote the set of tools available to the system. The meta-agent is implemented through role-conditioned calls to an LLM and acts as the meta-author, meta-router, and meta-synthesizer.

Each worker receives the same query and starts solving it independently until they reach a final answer or the maximum step budget. The workers do not directly broadcast their full contexts to one another. Instead, every exchange is selected by the meta-router based on the current state of all

workers. Thus, a worker agent is represented as

$$a_i = \left(\rho_i, \mathcal{T}_i, q, H_i^t\right), \qquad \mathcal{T}_i \subseteq \mathcal{T}, \tag{1}$$

where ρ_i is its reasoning strategy and system prompt, $\mathcal{T}_i$ is its assigned tool set, and H_i^t is its memory after step t.

3.2 Query-conditioned Worker Portfolio

Prior works such as TUMIX (Chen et al., 2025b) uses a fixed set of agents across all queries. However, the reasoning strategies that are useful for one query may be redundant for another. We therefore let the meta-author synthesize the worker agents separately for each query:

$$\mathcal{A} = \mathcal{M}_{\text{author}}\left(q, \mathcal{T}, N\right). \tag{2}$$

For each worker, the meta-author specifies a reasoning strategy ρ_i and selects a subset of tools $\mathcal{T}_i$. The reasoning strategy describes how the worker should approach the question, combining query-specific planning with reasoning.

The meta-author is instructed to produce workers that approach a query through different reasoning and tool-use configurations. This is intended to encourage trajectory diversity. For example, one worker may verify claims through code, while another may prioritize evidence obtained through search. This diversity is important because repeatedly sampling highly correlated agents provides limited additional coverage (Yang et al., 2026). Each synthesized prompt is augmented with instructions on how to invoke the worker's assigned tools and the required response format.

3.3 Structured Agent Memory

For the meta-router to guide communication, it must first understand the state of every worker. Passing complete trajectories, raw tool outputs, or concatenated responses is not suitable for this purpose. These histories grow throughout execution, useful evidence can be buried inside long tool outputs, and combining multiple workers' histories can overwhelm the meta-router's context.

The context exposed by each worker should satisfy three properties. First, it should be *compact*, so that the states of all workers can be read together without overwhelming the context. Second, it should be *comparable*, so that the meta-router can determine what each worker has accomplished and where it is struggling despite differences in reasoning style. DyTopo (Lu et al., 2026) addresses this problem using short "need" and "offer" descriptors. We retain this legibility while maintaining a richer representation of the worker state. Third, the context should be *sufficient*, so it should retain the task requirement, task-relevant progress, unresolved questions, and references to evidence supporting the worker's current beliefs.

Motivated by hierarchical working-memory approaches such as HiAgent (Hu et al., 2024) and WebResearcher (Qiao et al., 2025), we represent the memory of a worker agent a_i as

$$H_i^t = \left(H_{i,\text{task}}, H_{i,\text{work}}^t, H_{i,\text{tool}}^t\right). \tag{3}$$

Task memory. $H_{i,\text{task}}$ contains the user query and the task specification. It is fixed throughout execution and defines what the worker is attempting to solve.

Working memory. $H_{i,\text{work}}^t$ is a cumulative report of the worker's current state:

$$H_{i,\text{work}}^t = \left(s_i^t, O_i^t, P_i^t, U_i^t, A_i^t, E_i^t\right), \tag{4}$$

where s_i^t is a free-text summary, O_i^t is an ordered set of observations, P_i^t contains current hypotheses, U_i^t contains open questions, A_i^t contains planned next actions, and E_i^t contains references to evidence supporting the worker's current beliefs. These fields give the meta-router a standardized view across workers. Observations and supporting evidence describe what the worker can offer, open questions and next actions describe what it needs, and hypotheses reveal disagreements before they are hidden by a final answer.

The working memory evolves over time, with each worker updating it based on the previous state. At every step, the worker outputs an updated structured report alongside its action

$$\left(o_i^{t+1}, H_{i,\text{work}}^{t+1}\right) \sim a_i\left(q, H_i^t, B_i^t, \mathcal{T}_i\right), \tag{5}$$

where o_i^{t+1} is the worker's next action or answer, and B_i^t is the information routed to it by the meta-router. The working memory is a bounded summary of information that the worker currently considers relevant, allowing the exposed state to remain compact even when the worker requires many steps.

Tool memory. $H_{i,\text{tool}}^t$ records the tool actions executed by the worker and their observations, including failures. While the working memory represents the worker's interpretation of its progress, the tool memory preserves the underlying evidence on which that interpretation is based. Keeping both allows the meta-router to distinguish a supported conclusion from an unsupported claim.

3.4 Checkpoint-Gated Evidence Routing

Existing multi-agent systems commonly use a fixed communication rule. TUMIX, for example, provides each worker with the previous answers of all other workers over multi-round refinement, while GoA constructs exchanges trajectory information over a weighted graph of self-generated agent importance scores. Our method instead uses a model-based policy at fixed checkpoints to decide whether communication is useful, how the communication graph should be structured, and what information should be exchanged.

To describe this process, we distinguish the worker's internal memory H_i^t from its incoming information buffer B_i^t. As defined in Equation 3, H_i^t comprises the task, working, and tool memories carried forward from the preceding worker step. In contrast, B_i^t stores evidence received from other workers at the most recent communication checkpoint. This separation is intended to reduce the chance that routed information is treated as automatically correct, while making its effect on the subsequent agent state explicit. The incoming information is presented as evidence to be evaluated instead of being directly inserted into the recipient's state. A recipient can therefore verify, reject, or ignore the information routed to it.

Workers execute k steps between communication checkpoints. At each checkpoint, the meta-router reads the workers' tool and working memory, then jointly produces all communication decisions:

$$C^t = \mathcal{M}_{\text{router}}\left(q, \left\{H_{i,\text{work}}^t, H_{i,\text{tool}}^t\right\}_{i=1}^N\right). \tag{6}$$

Each decision in C^t is a directed route

$$c_{j\to i}^t = \left(a_j, a_i, R_{j\to i}^t\right), \tag{7}$$

where a_j is the source agent, a_i is the recipient agent, and $R_{j\to i}^t$ contains information packets owned by a_j. The meta-router may return an empty set C^t when the worker agents currently do not benefit from information exchange. The source and recipient in Equation 7 induce a directed communication graph G^t:

$$G^t = \left(\mathcal{A}, E^t\right), \qquad E^t = \left\{(a_j, a_i) \mid c_{j\to i}^t \in C^t\right\}. \tag{8}$$

The graph is reconstructed at every checkpoint and can therefore change as the workers make progress. A route may copy a working-memory observation, a hypothesis, an open question, a plan, a raw tool result, or a bounded model response. Multiple references can be routed together.

The selected packets are appended to the recipient's incoming-evidence buffer:

$$B_i^t \leftarrow B_i^t \bigoplus_{\substack{j\neq i:\\ c_{j\to i}^t \in C^t}} R_{j\to i}^t, \tag{9}$$

where $\oplus$ denotes an append operation. This delivery does not directly overwrite $H_{i,\text{work}}^t$. Instead, the post-routing buffer becomes the incoming information for the next worker step. Equation 5 then produces the next action and rolling working memory report conditioned on this updated buffer.

3.5 Final Answer Synthesis

Once all workers reach a terminal state by answering or exhausting their budgets, the meta-synthesizer receives their candidate answers and terminal memories and generates the final answer

$$\hat{y} = \mathcal{M}_{\text{synthesizer}}\left(q, \left\{y_i, H_i^{T_i}\right\}_{i=1}^{N}\right), \tag{10}$$

where T_i is worker a_i's terminal turn, $H_i^{T_i}$ is its terminal memory, and y_i is its candidate answer.

The meta-synthesizer is asked to identify the answer that is best supported by the collected evidence. By reading the workers' terminal structured memories, it can compare not only their candidate answers but also the observations, hypotheses, and the supporting evidence for each answer.

4 Experiments

Benchmarks and metrics: We test TANGO across four domains. GAIA (Mialon et al., 2023) evaluates general agentic problem solving through 165 validation tasks requiring real-world reasoning, file handling, multimodal understanding, and tool use. GPQA-Diamond (Rein et al., 2023) evaluates scientific reasoning through 198 expert-written questions in biology, physics, and chemistry. Omni-MATH (Gao et al., 2024a) evaluates Olympiad-level mathematical reasoning. Following Lin & Xu (2025), we use a curated subset of Omni-MATH comprising the 512 hardest problems that support reliable rule-based evaluation. HumanEval (Chen et al., 2021) evaluates code generation through 164 programming problems with executable unit tests. For every benchmark, A@3 is the mean accuracy across three independent runs, and P@3 is the fraction of tasks solved in at least one run. We report the population standard deviation of run-level accuracy alongside A@3. We focus the ablation studies on GAIA and Omni-MATH, whose complementary coverage spans heterogeneous coding, search, and reasoning tasks as well as challenging mathematical reasoning.

Baselines: We compare against a Gemini-2.5-Flash agent with Python, external Serper search, and Gemini grounded search. The five- and fifteen-sample Majority Vote controls repeat this same tool-enabled agent, and Self-MoA (Li et al., 2025a) aggregates 15 of its responses. We also evaluate TUMIX, which uses 15 text-, search-, and code-based agents with answer refinement over multiple rounds. We divide the remaining methods by their dominant control pattern. Manager-led decomposition methods like Flash Searcher (Qin et al., 2025), AOrchestra, AutoAgents, and Magentic-One (Fourney et al., 2024) use a planner or orchestrator to decompose and delegate work. Knowledge GoT (Besta et al., 2025) builds a knowledge graph using multiple tools before solving the task. Collaborative-communication methods like AgentScope (Gao et al., 2024b), Graph-of-Agents (GoA), GPTSwarm, and DyTopo coordinate agents through broadcast or graph-structured information exchange. Methods retain their method-specific tools, inference budgets, search procedures, and aggregation rules, so Table 1 compares complete systems in their intended configurations. Appendix A provides all hyperparameters, and Appendix E describes baseline tools and settings.

5 Results

5.1 Main Results and Component Contributions

Table 1 compares TANGO with test-time scaling, manager-led decomposition, and collaborative-communication methods. TANGO obtains the highest observed Overall A@3 of 77.62% and Overall P@3 of 85.44%, has the highest observed A@3 on GAIA, GPQA-Diamond, and Omni-MATH, and is competitive with the strongest baselines on HumanEval.

Among test-time scaling methods, simply increasing the number of repeated samples yields limited gains: moving from 5 to 15 samples raises Majority Vote A@3 from 40.00% to 43.23% on GAIA, 83.67% to 84.34% on GPQA-Diamond, and 69.34% to 70.96% on Omni-MATH. TUMIX uses 15 specialized agents and multi-round trajectory evolution, yet its Overall A@3 remains below both Majority Vote (15) and TANGO. TANGO reaches 55.15% GAIA A@3, 85.69% on GPQA-Diamond, and 71.88% Omni-MATH A@3 with only five task-conditioned workers. Notably, TANGO achieves higher Overall A@3 than Majority Vote (15), showing that task-adaptive orchestration can be more effective than naively scaling independent samples alone. TANGO also

Table 1: A@3 is the mean accuracy across three runs and P@3 is pass@3. Overall A@3 and P@3 are macro-averages over all benchmarks and are reported only when all four values are available. Best values are bold and second-best values are underlined.

	GAIA		GPQA-Diamond		Omni-MATH		HumanEval		Overall	
Method	A@3	P@3	A@3	P@3	A@3	P@3	A@3	P@3	A@3	P@3
Test-time scaling										
ReAct Agent	$33.54_{\pm 0.76}$	49.09	$78.79_{\pm 1.89}$	88.89	$63.02_{\pm 1.68}$	75.00	$80.08_{\pm 3.00}$	93.90	63.86	76.72
Majority Vote (5)	$40.00_{\pm 1.78}$	52.73	$83.67_{\pm 1.26}$	88.89	$69.34_{\pm 1.12}$	77.15	$96.54_{\pm 0.29}$	98.78	72.39	79.39
TUMIX	$38.99_{\pm 2.29}$	51.52	$84.01_{\pm 1.26}$	90.91	$69.79_{\pm 1.06}$	79.10	$97.76_{\pm 0.29}$	98.17	72.64	79.92
Self-MoA (15)	$38.18_{\pm 1.78}$	50.91	$83.16_{\pm 1.56}$	90.91	$70.12_{\pm 1.27}$	78.52	$96.95_{\pm 0.00}$	98.17	72.10	79.63
Majority Vote (15)	$43.23_{\pm 0.76}$	50.30	$84.34_{\pm 1.80}$	89.39	$70.96_{\pm 0.18}$	75.20	$\mathbf{97.97_{\pm 0.29}}$	98.17	74.12	78.27
Manager-led decomposition										
AOrchestra	$36.57_{\pm 0.57}$	47.27	$80.13_{\pm 1.33}$	89.39	$60.22_{\pm 0.92}$	73.24	$78.05_{\pm 3.03}$	97.56	63.74	76.87
Knowledge GoT	$38.38_{\pm 2.23}$	52.12	$73.06_{\pm 1.67}$	82.32	$57.62_{\pm 1.28}$	68.95	$93.09_{\pm 0.57}$	96.95	65.54	75.09
Flash Searcher	$42.83_{\pm 1.87}$	58.79	$76.94_{\pm 1.45}$	86.36	$58.46_{\pm 1.92}$	72.07	$97.76_{\pm 0.29}$	98.78	69.00	79.00
Magentic-One	$49.90_{\pm 1.74}$	65.45	$75.59_{\pm 3.15}$	88.89	$62.30_{\pm 1.05}$	73.44	$95.93_{\pm 0.29}$	98.17	70.93	81.49
AutoAgents	–	–	$75.08_{\pm 0.63}$	90.40	$54.04_{\pm 1.04}$	68.75	$81.50_{\pm 0.76}$	96.95	–	–
Collaborative communication										
AgentScope	$22.22_{\pm 0.29}$	35.76	$73.06_{\pm 1.56}$	86.36	$53.19_{\pm 0.49}$	68.75	$96.95_{\pm 0.86}$	98.78	61.36	72.41
GoA	$45.05_{\pm 2.73}$	60.00	$69.70_{\pm 2.58}$	88.38	$64.32_{\pm 0.33}$	72.85	$95.73_{\pm 0.50}$	97.56	68.70	79.70
GPTSwarm	$32.53_{\pm 2.06}$	47.27	–	–	–	–	$96.75_{\pm 1.04}$	**99.39**	–	–
DyTopo	–	–	$77.95_{\pm 1.86}$	87.37	$64.39_{\pm 0.66}$	74.80	$82.72_{\pm 0.57}$	95.12	–	–
Ours										
TANGO	$\mathbf{55.15_{\pm 0.49}}$	**68.48**	$\mathbf{85.69_{\pm 1.04}}$	**92.42**	$\mathbf{71.88_{\pm 0.73}}$	**81.45**	$97.76_{\pm 0.57}$	**99.39**	**77.62**	**85.44**

compares favorably with manager-led decomposition methods. It obtains higher observed A@3 than Magentic-One on all four benchmarks despite relying on a smaller tool catalog. TANGO also exceeds Flash Searcher on GAIA, GPQA-Diamond, and Omni-MATH while remaining competitive on HumanEval. Among collaborative-communication methods, TANGO obtains higher observed A@3 than all reported methods. These methods improve how agents communicate but generally retain predetermined worker construction or final aggregation. TANGO instead couples communication with task-conditioned worker authoring and evidence-aware synthesis. Taken together, the comparisons indicate that orchestration quality depends on the interaction between worker composition, communication, and conclusion rather than on any one stage in isolation.

Table 2 separates the three lifecycle stages under fixed and authored worker portfolios. For the fixed condition, we selected a five-agent portfolio from TUMIX before evaluation to cover text-only, code, search, and combined search-and-code strategies, and kept this portfolio unchanged across tasks. The delta columns report each configuration's absolute percentage-point change from Fixed + MV. Authoring produces its largest gain on GAIA: replacing the fixed portfolio with Author + MV raises GAIA A@3 from 39.19% to 45.66%, while the Omni-MATH gain is slight. Routing also appears more effective when paired with authored workers, with the gains from selective communication being consistently larger under the authored portfolio on both benchmarks as compared to the fixed portfolio

Table 2: Lifecycle component analysis. Delta columns report percentage-point changes from Fixed + MV. Best values are bold and second-best values are underlined.

	GAIA				Omni-MATH			
Method	A@3	ΔA@3	P@3	ΔP@3	A@3	ΔA@3	P@3	ΔP@3
Fixed + MV	$39.19_{\pm 0.57}$	–	52.73	–	$68.03_{\pm 0.09}$	–	75.00	–
Fixed + Router + MV	$41.62_{\pm 0.29}$	+2.43	58.18	+5.45	$68.88_{\pm 0.33}$	+0.85	76.56	+1.56
Fixed + Synthesizer	$46.46_{\pm 4.46}$	+7.27	60.00	+7.27	$69.47_{\pm 0.56}$	+1.44	77.73	+2.73
Fixed + Router + Synthesizer	$44.04_{\pm 1.03}$	+4.85	58.18	+5.45	$69.66_{\pm 0.79}$	+1.63	79.10	+4.10
Author + MV	$45.66_{\pm 0.76}$	+6.47	57.58	+4.85	$68.23_{\pm 0.37}$	+0.20	76.17	+1.17
Author + Router + MV	$50.30_{\pm 1.31}$	+11.11	65.45	+12.72	$71.03_{\pm 0.79}$	+3.00	79.88	+4.88
Author + Synthesizer	$51.92_{\pm 3.37}$	+12.73	65.45	+12.72	$70.90_{\pm 0.48}$	+2.87	79.88	+4.88
TANGO	$\mathbf{55.15_{\pm 0.49}}$	**+15.96**	**68.48**	**+15.75**	$\mathbf{71.88_{\pm 0.73}}$	**+3.85**	**81.45**	**+6.45**

Synthesis provides substantial gains for fixed and authored workers. The improvement is consistent across both GAIA and Omni-MATH, indicating that stronger final-answer integration contributes meaningfully regardless of how the worker portfolio is constructed. Adding routing to the fixed-worker synthesizer configuration reduces GAIA A@3 from 46.46% to 44.04% and P@3 from 60.00% to 58.18%, while changing Omni-MATH A@3 by only 0.19 points. In contrast, when routing is combined with the full authored-worker and synthesis configuration, GAIA and Omni-MATH A@3 improve by 3.23 and 0.98 points, respectively. These results suggest that routing benefits more from task-conditioned worker diversity than from a fixed portfolio, and the combination of all three stages produces the highest GAIA and Omni-MATH A@3 and P@3. Further analysis of authored agents, routing, and synthesis is provided in Appendices B, C, and D, respectively.

5.2 Structured memory and communication topology

In this section, we study whether selective routing can replace dense information sharing and whether structured memory improves the state representation on which routing decisions are made. Table 3 compares dense all-to-all sharing with selective meta-agent routing over either raw trajectories or structured memory. Raw trajectories contain all previous agent steps, while structured memory exposes a compact, standardized view of worker progress and evidence. When routing over raw trajectories, the router treats each complete worker step as a single unit of transfer, whereas structured memory allows it to route individual memory fields or tool results. We keep the authored workers fixed and use majority voting in every row so that the comparison isolates context representation and communication policy.

Table 3: Dense all-to-all sharing versus selective routing over raw trajectories or structured memory. Worker tokens, edges, payload tokens, and cost are reported per-task. Cost includes all worker, meta-agent, and search charges incurred by each configuration.

	GAIA						Omni-MATH					
Policy	A@3	P@3	Worker tok.	Edges	Payload	Cost	A@3	P@3	Worker tok.	Edges	Payload	Cost
Raw trajectories												
All-to-all	$46.46_{\pm 1.74}$	60.61	442.07K	14.62	117.36K	\$0.33	$69.73_{\pm 0.55}$	77.34	260.26K	3.59	33.78K	\$0.38
Router	$48.89_{\pm 2.49}$	61.82	207.41K	3.82	6.38K	\$0.37	$69.53_{\pm 0.83}$	78.32	178.32K	0.76	1.81K	\$0.39
Structured memory												
All-to-all	$45.86_{\pm 1.03}$	57.58	274.21K	13.25	46.95K	\$0.28	$69.01_{\pm 1.72}$	78.71	251.65K	10.22	28.68K	\$0.34
Router	$50.30_{\pm 1.31}$	65.45	307.82K	2.15	1.57K	\$0.42	$71.03_{\pm 0.79}$	79.88	214.19K	3.58	1.82K	\$0.43

Neither selective routing nor structured memory consistently improves performance in isolation. Routing over raw trajectories improves GAIA A@3 but has little effect on Omni-MATH, while structured memory with all-to-all communication yields no improvement on either benchmark. However, the router consistently reduces communication edges by 65–84% and uses 93–97% fewer payload tokens compared to all-to-all sharing, indicating that it favors selective information exchange. Combining structured memory with routing produces the highest observed A@3 and P@3 on both benchmarks while keeping communication sparse and payload volumes low. This pattern is consistent with structured memory making task-relevant information easier for the router to identify and target. However, this gain comes with additional cost because the meta-router makes extra LLM calls to reason jointly over the structured states of all workers at each checkpoint. Appendix C.1 analyzes communication efficiency over the trajectory and discusses adaptive checkpoint schedules that could reduce this overhead.

5.3 Checkpoint and worker-count sensitivity

Figure 2 examines two dimensions of TANGO's inference-time configuration on GAIA and Omni-MATH. We vary the checkpoint interval k over $(1, 2, 3, 4)$ while holding the worker count at five, and vary the number of authored workers N over $(3, 5, 7, 9)$ while holding the checkpoint interval at two.

Checkpoint timing has a similar non-monotonic effect on both benchmarks. The k=2 setting gives the highest P@3 on GAIA and Omni-MATH and the highest Omni-MATH A@3. One possible

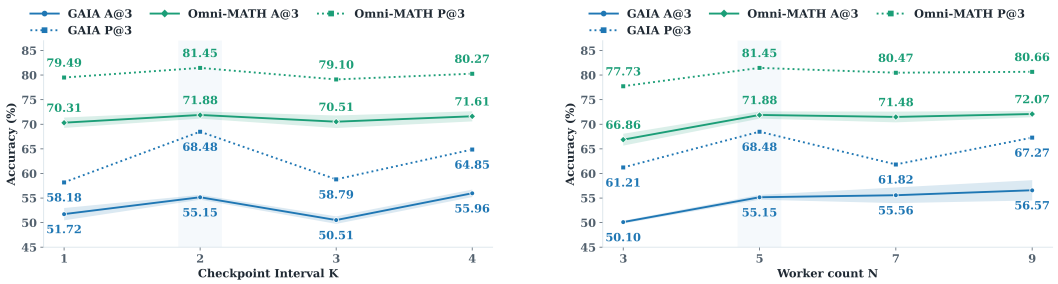


Figure 2: Checkpoint-interval and worker-count sensitivity. Left: performance as the checkpoint interval k changes. Right: performance as the number of worker agents N changes. Shaded bands denote population standard deviation.

explanation is that two turns support a natural tool-use cycle: an agent can request a tool in one turn, then inspect its output and update structured memory before the communication checkpoint. With k=1, information may be routed before the worker has integrated the tool result, while larger intervals delay potentially useful evidence.

Worker-count sensitivity shows clear diminishing returns after $N = 5$. On GAIA, A@3 rises from 50.10% with three workers to 55.15% with five, but reaches only 56.16% with nine. On Omni-MATH, A@3 rises from 66.86% to 71.88% between three and five workers, while seven and nine workers remain near this level. The pre-specified five-worker configuration therefore lies near the observed knee of both curves. A stronger meta-agent could potentially construct larger portfolios with more complementary workers, making joint adaptation of worker count and portfolio composition an interesting direction for future work.

6 Limitations and Future Work

First, we evaluate TANGO using a single proprietary backbone, leaving its generalizability to other models unexplored. Second, TANGO invokes the meta-router at fixed checkpoints, even though most communication occurs early in execution (Figure 8). Routing overhead could be reduced by using an adaptive checkpoint interval schedule or stopping routing when repeated checkpoints yield no communication. Third, TANGO manages three central orchestration stages but still keeps certain parameters fixed before each run, including the fixed tool catalog, maximum turns, and worker count. It therefore does not currently manage every aspect of the operational agent lifecycle. Future work should let the meta-agent construct tools and exercise broader control over the orchestration lifecycle. Finally, the oracle results in Table 4 highlight the potential for further improvements through better evidence selection and integration.

7 Conclusion

How can a meta-agent manage the three central stages of multi-agent orchestration? TANGO addresses this question through role-conditioned controllers that create task-specific workers, coordinate selective information exchange through structured memory, and conclude over the workers' terminal reasoning. Across four benchmarks, TANGO achieves the highest observed Overall A@3 and P@3 macro-averages in our comparison. The expanded component study shows that authoring, routing, and synthesis interact rather than contributing uniformly: synthesis provides a larger isolated gain than routing within the fixed-worker branch, routing helps more with authored workers, and combining all three stages yields the strongest overall results. Structured memory is most effective when paired with selective routing, giving the meta-router compact worker states while keeping routed payloads small. Together, the results support three-stage meta-agent control as a promising direction for adaptive multi-agent systems while leaving broader resource and tool adaptation to future work.

AI USE STATEMENT

We used generative AI tools to assist with implementing and debugging parts of the code, polishing manuscript text, formatting tables, preparing the submitted codebase, and suggesting layouts for some illustrative graphics. All methodological and experimental decisions were made by the authors, and all quantitative graphs were created by the authors from the experimental results. The authors manually reviewed all AI-assisted text and code, tested the implementation, and checked reported values, tables, and graphics against the underlying results. The authors take full responsibility for the paper, code, figures, claims, and experimental results. Generative AI did not determine the research question, methodology, experimental design, or conclusions. The authors manually curated and verified all citations.

ETHICS STATEMENT

This work evaluates LLM orchestration on established benchmarks and does not involve human subjects, crowdsourcing, or the collection of new personal or sensitive data. The models, datasets, and baseline implementations are credited to their original creators and remain subject to their respective licenses and access terms. Multi-agent tool use can propagate unsupported evidence or access unintended information if deployed without safeguards. We therefore recommend restricting deployments to authorized data and tools, sandboxing code execution, validating retrieved sources, and requiring human review for consequential applications.

REPRODUCIBILITY STATEMENT

Sections 3 and 4, Algorithm 1, and Appendices A, E, and F document the method, evaluation protocol, principal implementation settings, baseline configurations, and prompt templates. We provide the complete anonymized codebase with environment-setup instructions, benchmark adapters, and scripts needed to reproduce the main experiments and ablations. Reproduction requires access to Gemini, Serper, and any benchmark resources governed by their original access terms.

REFERENCES

Salaheddin Alzu'bi, Baran Nama, Arda Kaz, Anushri Eswaran, Weiyuan Chen, Sarvesh Khetan, Rishab Bala, Tu Vu, and Sewoong Oh. Roma: Recursive open meta-agent framework for long-horizon multi-agent systems, 2026. URL https://arxiv.org/abs/2602.01848.

Maciej Besta, Nils Blach, Ales Kubicek, Robert Gerstenberger, Michal Podstawski, Lukas Gianinazzi, Joanna Gajda, Tomasz Lehmann, Hubert Niewiadomski, Piotr Nyczyk, and Torsten Hoefler. Graph of thoughts: Solving elaborate problems with large language models. *Proceedings of the AAAI Conference on Artificial Intelligence*, 38(16):17682–17690, March 2024. ISSN 2159-5399. doi: 10.1609/aaai.v38i16.29720. URL http://dx.doi.org/10.1609/aaai.v38i16.29720.

Maciej Besta, Lorenzo Paleari, Jia Hao Andrea Jiang, Robert Gerstenberger, You Wu, Jón Gunnar Hannesson, Patrick Iff, Ales Kubicek, Piotr Nyczyk, Diana Khimey, Nils Blach, Haiqiang Zhang, Tao Zhang, Peiran Ma, Grzegorz Kwaśniewski, Marcin Copik, Hubert Niewiadomski, and Torsten Hoefler. Affordable ai assistants with knowledge graph of thoughts, 2025. URL https://arxiv.org/abs/2504.02670.

Bradley Brown, Jordan Juravsky, Ryan Ehrlich, Ronald Clark, Quoc V. Le, Christopher Ré, and Azalia Mirhoseini. Large language monkeys: Scaling inference compute with repeated sampling, 2024. URL https://arxiv.org/abs/2407.21787.

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?, 2025. URL https://arxiv.org/abs/2503.13657.

Guangyao Chen, Siwei Dong, Yu Shu, Ge Zhang, Jaward Sesay, Börje Karlsson, Jie Fu, and Yemin Shi. Autoagents: A framework for automatic agent generation. In Kate Larson (ed.), *Proceedings*

of the Thirty-Third International Joint Conference on Artificial Intelligence, IJCAI-24, pp. 22–30. International Joint Conferences on Artificial Intelligence Organization, 8 2024a. doi: 10.24963/ijcai.2024/3. URL https://doi.org/10.24963/ijcai.2024/3. Main Track.

Jiefeng Chen, Jie Ren, Xinyun Chen, Chengrun Yang, Ruoxi Sun, Jinsung Yoon, and Sercan Ö Arık. Sets: Leveraging self-verification and self-correction for improved test-time scaling, 2025a. URL https://arxiv.org/abs/2501.19306.

Justin Chen, Swarnadeep Saha, and Mohit Bansal. ReConcile: Round-table conference improves reasoning via consensus among diverse LLMs. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 7066–7085, Bangkok, Thailand, August 2024b. Association for Computational Linguistics. doi: 10.18653/v1/2024.acl-long.381. URL https://aclanthology.org/2024.acl-long.381/.

Justin Chih-Yao Chen, Sukwon Yun, Elias Stengel-Eskin, Tianlong Chen, and Mohit Bansal. Skill-based mixture-of-experts: Adaptive routing for heterogeneous reasoning via inferred skills, 2026. URL https://arxiv.org/abs/2503.05641.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. Evaluating large language models trained on code, 2021.

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, 2023a. URL https://arxiv.org/abs/2308.10848.

Xinyun Chen, Renat Aksitov, Uri Alon, Jie Ren, Kefan Xiao, Pengcheng Yin, Sushant Prakash, Charles Sutton, Xuezhi Wang, and Denny Zhou. Universal self-consistency for large language model generation, 2023b. URL https://arxiv.org/abs/2311.17311.

Yongchao Chen, Jiefeng Chen, Rui Meng, Ji Yin, Na Li, Chuchu Fan, Chi Wang, Tomas Pfister, and Jinsung Yoon. Tumix: Multi-agent test-time scaling with tool-use mixture, 2025b. URL https://arxiv.org/abs/2510.01279.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. Training verifiers to solve math word problems, 2021. URL https://arxiv.org/abs/2110.14168.

Gheorghe Comanici, Eric Bieber, Mike Schaekermann, Ice Pasupat, Noveen Sachdeva, Inderjit Dhillon, Marcel Blistein, Ori Ram, Dan Zhang, Evan Rosen, et al. Gemini 2.5: Pushing the frontier with advanced reasoning, multimodality, long context, and next generation agentic capabilities. *arXiv preprint arXiv:2507.06261*, 2025. URL https://arxiv.org/abs/2507.06261.

Yilun Du, Shuang Li, Antonio Torralba, Joshua B. Tenenbaum, and Igor Mordatch. Improving factuality and reasoning in language models through multiagent debate, 2023. URL https://arxiv.org/abs/2305.14325.

Adam Fourney, Gagan Bansal, Hussein Mozannar, Cheng Tan, Eduardo Salinas, Erkang, 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, 2024. URL https://arxiv.org/abs/2411.04468.

Bofei Gao, Feifan Song, Zhe Yang, Zefan Cai, Yibo Miao, Qingxiu Dong, Lei Li, Chenghao Ma, Liang Chen, Runxin Xu, Zhengyang Tang, Benyou Wang, Daoguang Zan, Shanghaoran Quan, Ge Zhang, Lei Sha, Yichang Zhang, Xuancheng Ren, Tianyu Liu, and Baobao Chang. Omni-math: A universal olympiad level mathematic benchmark for large language models, 2024a. URL https://arxiv.org/abs/2410.07985.

Dawei Gao, Zitao Li, Xuchen Pan, Weirui Kuang, Zhijian Ma, Bingchen Qian, Fei Wei, Wenhao Zhang, Yuexiang Xie, Daoyuan Chen, Liuyi Yao, Hongyi Peng, Zeyu Zhang, Lin Zhu, Chen Cheng, Hongzhu Shi, Yaliang Li, Bolin Ding, and Jingren Zhou. Agentscope: A flexible yet robust multi-agent platform, 2024b. URL https://arxiv.org/abs/2402.14034.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, et al. DeepSeek-R1: Incentivizing reasoning capability in LLMs via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025. URL https://arxiv.org/abs/2501.12948.

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 a multi-agent collaborative framework, 2024. URL https://arxiv.org/abs/2308.00352.

Mengkang Hu, Tianxing Chen, Qiguang Chen, Yao Mu, Wenqi Shao, and Ping Luo. Hiagent: Hierarchical working memory management for solving long-horizon agent tasks with large language model, 2024. URL https://arxiv.org/abs/2408.09559.

Shengran Hu, Cong Lu, and Jeff Clune. Automated design of agentic systems, 2025. URL https://arxiv.org/abs/2408.08435.

Aaron Jaech, Adam Kalai, Adam Lerer, Adam Richardson, Ahmed El-Kishky, Aiden Low, Alec Helyar, Aleksander Madry, Alex Beutel, Alex Carney, et al. OpenAI o1 system card. *arXiv preprint arXiv:2412.16720*, 2024. URL https://arxiv.org/abs/2412.16720.

Yuxiang Ji, Ziyu Ma, Yong Wang, Guanhua Chen, Xiangxiang Chu, and Liaoni Wu. Tree search for llm agent reinforcement learning, 2026. URL https://arxiv.org/abs/2509.21240.

Ziwei Ji, Tiezheng Yu, Yan Xu, Nayeon Lee, Etsuko Ishii, and Pascale Fung. Towards mitigating hallucination in large language models via self-reflection, 2023. URL https://arxiv.org/abs/2310.06271.

Dongfu Jiang, Xiang Ren, and Bill Yuchen Lin. LLM-blender: Ensembling large language models with pairwise ranking and generative fusion. In Anna Rogers, Jordan Boyd-Graber, and Naoaki Okazaki (eds.), *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 14165–14178, Toronto, Canada, July 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.acl-long.792. URL https://aclanthology.org/2023.acl-long.792/.

Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. Large language models are zero-shot reasoners, 2023. URL https://arxiv.org/abs/2205.11916.

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, 2023. URL https://arxiv.org/abs/2303.17760.

Wenzhe Li, Yong Lin, Mengzhou Xia, and Chi Jin. Rethinking mixture-of-agents: Is mixing different large language models beneficial?, 2025a. URL https://arxiv.org/abs/2502.00674.

Yunxuan Li, Yibing Du, Jiageng Zhang, Le Hou, Peter Grabowski, Yeqing Li, and Eugene Ie. Improving multi-agent debate with sparse communication topology. In Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen (eds.), *Findings of the Association for Computational Linguistics: EMNLP 2024*, pp. 7281–7294, Miami, Florida, USA, November 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.findings-emnlp.427. URL https://aclanthology.org/2024.findings-emnlp.427/.

Zichong Li, Xinyu Feng, Yuheng Cai, Zixuan Zhang, Tianyi Liu, Chen Liang, Weizhu Chen, Haoyu Wang, and Tuo Zhao. Llms can generate a better answer by aggregating their own responses, 2025b. URL https://arxiv.org/abs/2503.04104.

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, 2024. URL https://arxiv.org/abs/2305.19118.

Hunter Lightman, Vineet Kosaraju, Yura Burda, Harri Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step, 2023. URL https://arxiv.org/abs/2305.20050.

Heng Lin and Zhongwen Xu. Understanding tool-integrated reasoning, 2025. URL https://arxiv.org/abs/2508.19201.

Zijun Liu, Yanzhe Zhang, Peng Li, Yang Liu, and Diyi Yang. A dynamic llm-powered agent network for task-oriented agent collaboration, 2024. URL https://arxiv.org/abs/2310.02170.

Yuxing Lu, Yucheng Hu, Xukai Zhao, and Jiuxin Cao. Dytopo: Dynamic topology routing for multi-agent reasoning via semantic matching, 2026. URL https://arxiv.org/abs/2602.06039.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. Self-refine: Iterative refinement with self-feedback, 2023. URL https://arxiv.org/abs/2303.17651.

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

Zile Qiao, Guoxin Chen, Xuanzhong Chen, Donglei Yu, Wenbiao Yin, Xinyu Wang, Zhen Zhang, Baixuan Li, Huifeng Yin, Kuan Li, Rui Min, Minpeng Liao, Yong Jiang, Pengjun Xie, Fei Huang, and Jingren Zhou. Webresearcher: Unleashing unbounded reasoning capability in long-horizon agents, 2025. URL https://arxiv.org/abs/2509.13309.

Tianrui Qin, Qianben Chen, Sinuo Wang, He Xing, King Zhu, He Zhu, Dingfeng Shi, Xinxin Liu, Ge Zhang, Jiaheng Liu, Yuchen Eleanor Jiang, Xitong Gao, and Wangchunshu Zhou. Flash-searcher: Fast and effective web agents via dag-based parallel execution, 2025. URL https://arxiv.org/abs/2509.25301.

David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, Julian Michael, and Samuel R. Bowman. Gpqa: A graduate-level google-proof q&a benchmark, 2023. URL https://arxiv.org/abs/2311.12022.

Jianhao Ruan, Zhihao Xu, Yiran Peng, Fashen Ren, Zhaoyang Yu, Xinbing Liang, Jinyu Xiang, Yongru Chen, Bang Liu, Chenglin Wu, Yuyu Luo, and Jiayi Zhang. Aorchestra: Automating sub-agent creation for agentic orchestration, 2026. URL https://arxiv.org/abs/2602.03786.

Noah Shinn, Federico Cassano, Edward Berman, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning, 2023. URL https://arxiv.org/abs/2303.11366.

Andries Petrus Smit, Nathan Grinsztajn, Paul Duckworth, Thomas D Barrett, and Arnu Pretorius. Should we be going MAD? A look at multi-agent debate strategies for LLMs. In Ruslan Salakhutdinov, Zico Kolter, Katherine Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp (eds.), *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 45883–45905. PMLR, 21–27 Jul 2024. URL https://proceedings.mlr.press/v235/smit24a.html.

Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. Scaling llm test-time compute optimally can be more effective than scaling model parameters, 2024. URL https://arxiv.org/abs/2408.03314.

Qiushi Sun, Zhangyue Yin, Xiang Li, Zhiyong Wu, Xipeng Qiu, and Lingpeng Kong. Corex: Pushing the boundaries of complex reasoning through multi-model collaboration, 2024. URL https://arxiv.org/abs/2310.00280.

Junlin Wang, Jue Wang, Ben Athiwaratkun, Ce Zhang, and James Zou. Mixture-of-agents enhances large language model capabilities, 2024a. URL https://arxiv.org/abs/2406.04692.

Lei Wang, Wanyu Xu, Yihuai Lan, Zhiqiang Hu, Yunshi Lan, Roy Ka-Wei Lee, and Ee-Peng Lim. Plan-and-solve prompting: Improving zero-shot chain-of-thought reasoning by large language models. In Anna Rogers, Jordan Boyd-Graber, and Naoaki Okazaki (eds.), *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 2609–2634, Toronto, Canada, July 2023a. Association for Computational Linguistics. doi: 10.18653/v1/2023.acl-long.147. URL https://aclanthology.org/2023.acl-long.147/.

Peiyi Wang, Lei Li, Zhihong Shao, R. X. Xu, Damai Dai, Yifei Li, Deli Chen, Y. Wu, and Zhifang Sui. Math-shepherd: Verify and reinforce llms step-by-step without human annotations, 2024b. URL https://arxiv.org/abs/2312.08935.

Qineng Wang, Zihao Wang, Ying Su, Hanghang Tong, and Yangqiu Song. Rethinking the bounds of LLM reasoning: Are multi-agent discussions the key? In Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 6106–6131, Bangkok, Thailand, August 2024c. Association for Computational Linguistics. doi: 10.18653/v1/2024.acl-long.331. URL https://aclanthology.org/2024.acl-long.331/.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models, 2023b. URL https://arxiv.org/abs/2203.11171.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models, 2023. URL https://arxiv.org/abs/2201.11903.

Sean Welleck, Ximing Lu, Peter West, Faeze Brahman, Tianxiao Shen, Daniel Khashabi, and Yejin Choi. Generating Sequences by Learning to Self-Correct. In *International Conference on Learning Representations*, 2023. URL https://mlanthology.org/iclr/2023/welleck2023iclr-generating/.

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, 2023. URL https://arxiv.org/abs/2308.08155.

Kevin Yang, Yuandong Tian, Nanyun Peng, and Dan Klein. Re3: Generating longer stories with recursive reprompting and revision. In Yoav Goldberg, Zornitsa Kozareva, and Yue Zhang (eds.), *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pp. 4393–4479, Abu Dhabi, United Arab Emirates, December 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.emnlp-main.296. URL https://aclanthology.org/2022.emnlp-main.296/.

Yingxuan Yang, Chengrui Qu, Muning Wen, Laixi Shi, Ying Wen, Weinan Zhang, Adam Wierman, and Shangding Gu. Understanding agent scaling in llm-based multi-agent systems via diversity, 2026. URL https://arxiv.org/abs/2602.03794.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L. Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models, 2023. URL https://arxiv.org/abs/2305.10601.

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, 2026. URL https://arxiv.org/abs/2604.17148.

Guibin Zhang, Yanwei Yue, Zhixun Li, Sukwon Yun, Guancheng Wan, Kun Wang, Dawei Cheng, Jeffrey Xu Yu, and Tianlong Chen. Cut the crap: An economical communication pipeline for llm-based multi-agent systems, 2024a. URL https://arxiv.org/abs/2410.02506.

Guibin Zhang, Yanwei Yue, Xiangguo Sun, Guancheng Wan, Miao Yu, Junfeng Fang, Kun Wang, Tianlong Chen, and Dawei Cheng. G-designer: Architecting multi-agent communication topologies via graph neural networks, 2025a. URL https://arxiv.org/abs/2410.11782.

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, 2025b. URL https://arxiv.org/abs/2410.10762.

Kexun Zhang, Weiran Yao, Zuxin Liu, Yihao Feng, Zhiwei Liu, Rithesh Murthy, Tian Lan, Lei Li, Renze Lou, Jiacheng Xu, Bo Pang, Yingbo Zhou, Shelby Heinecke, Silvio Savarese, Huan Wang, and Caiming Xiong. Diversity empowers intelligence: Integrating expertise of software engineering agents, 2024b. URL https://arxiv.org/abs/2408.07060.

Yusen Zhang, Ruoxi Sun, Yanfei Chen, Tomas Pfister, Rui Zhang, and Sercan Ö. Arik. Chain of agents: Large language models collaborating on long-context tasks, 2024c. URL https://arxiv.org/abs/2406.02818.

Han Zhou, Xingchen Wan, Ruoxi Sun, Hamid Palangi, Shariq Iqbal, Ivan Vulić, Anna Korhonen, and Sercan Ö. Arık. Multi-agent design: Optimizing agents with better prompts and topologies, 2026. URL https://arxiv.org/abs/2502.02533.

Mingchen Zhuge, Wenyi Wang, Louis Kirsch, Francesco Faccio, Dmitrii Khizbullin, and Jürgen Schmidhuber. GPTSwarm: Language agents as optimizable graphs. In Ruslan Salakhutdinov, Zico Kolter, Katherine Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp (eds.), *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 62743–62767. PMLR, 21–27 Jul 2024. URL https://proceedings.mlr.press/v235/zhuge24a.html.

A Implementation Details

All worker, author, router, and synthesizer calls use Gemini-2.5-Flash. Primary runs use five authored workers. Each reported configuration aggregates three independent runs, and population standard deviation is computed across run-level accuracies. Before running the primary evaluations, we fixed the common configuration at five workers and checkpoint interval $k = 2$ for all benchmarks. The sensitivity studies characterize these choices and were not used for benchmark-specific tuning. Workers receive at most 50 turns on GAIA, GPQA-Diamond, Omni-MATH and HumanEval. Worker calls have a 16,384-token output ceiling except on Omni-MATH, where the limit is 65,536 tokens. The meta-agent calls have a 32,768-token ceiling everywhere. The TANGO tool catalog contains Python execution, external search through Serper, and Gemini grounded search. For search-based tool calls, before retrieved content is exposed to an agent, a benchmark-aware rule removes known answer-bearing dataset URLs, URL/title combinations containing either a benchmark alias or a task identifier. We programmatically inspect all trajectories containing search calls and find no instances in which retrieved content leads to a benchmark answer. We use sampling configuration corresponding to temperature 1.0 and top-$p = 0.95$. Each worker turn permits at most one tool action. Model requests use a 600-second timeout with two retries, and Python executions use a 100-second timeout. To isolate the effect of orchestration, all controlled comparisons under test-time scaling methods use identical worker backbones, tool availability, sampling parameters, execution limits, and evaluation procedures as specified above, unless explicitly noted. For baseline methods using their original implementations, we preserve their method-specific orchestration and tool configurations as described in Appendix E.

A working-memory snapshot is limited to 16,000 characters, including a 4,000-character summary; each memory field can contain at most 64 items of at most 2,000 characters each. A worker sees its latest 16 tool entries and 32 incoming packets. Information excerpts are truncated to 12,000 characters, and each checkpoint can create at most 64 routes, with one to eight references per route. Meta-agent prompts are guarded at 480,000 characters. During final synthesis, the latest eight tool excerpts per worker are retained, with each excerpt truncated to 2,000 characters.

Compact routing decisions. We prompt the meta-router to represent each routing decision as a tuple containing a source-agent identifier, recipient-agent identifier, and information identifiers. Rather than reproducing the full information in its output, the router emits only these three fields per route. The system then retrieves the corresponding information from the source agent and copies it into the recipient agent's buffer, reducing meta-router output-token usage.

Algorithm 1 Meta-agent-guided multi-agent reasoning

Require: Query q, available tools $\mathcal{T}$, number of workers N, checkpoint interval k
Ensure: Final answer $\hat{y}$

$\mathcal{A} \leftarrow \mathcal{M}_{\text{author}}(q, \mathcal{T}, N)$
Initialize $t = 0$, H_i^0, and $B_i^0 = \emptyset$ for each $a_i \in \mathcal{A}$
while at least one worker is active **do**
  for all active workers a_i **in parallel do**
    Execute a step $(o_i^{t+1}, H_i^{t+1}) \sim a_i(q, H_i^t, B_i^t)$
  end for
  Set $B_i^{t+1} = \emptyset$ for each $a_i \in \mathcal{A}$
  if $(t+1) \bmod k = 0$ **then**
    Identify routes $C^{t+1} \leftarrow \mathcal{M}_{\text{router}}(q, \{H_{i,tool}^{t+1}, H_{i,work}^{t+1}\}_{i=1}^N)$
    for all $c_{j \to i}^{t+1} \in C^{t+1}$ with $j \neq i$ **do**
      Append $R_{j \to i}^{t+1}$ to B_i^{t+1}
    end for
  end if
  $t \leftarrow t + 1$
end while
$\hat{y} \leftarrow \mathcal{M}_{\text{synthesizer}}(q, \{y_i, H_i^{T_i}\}_{i=1}^N)$
return $\hat{y}$

B AUTHORED AGENT DIVERSITY ANALYSIS

B.1 PORTFOLIO-LEVEL STRATEGY AND TOOL ALLOCATION

We analyze the worker agents generated by the Gemini-2.5-Flash meta-author across GAIA, GPQA-Diamond, HumanEval, and Omni-MATH. Figure 3 highlights the changes in the authored portfolio as the tasks change. The meta-author is provided with multiple in-context examples of different reasoning strategies. The meta-author can either combine multiple reasoning strategies, or design new ones from scratch. Thus, each authored agent may contain a combination of multiple reasoning strategies. Program-of-thought remains near 20% across all four benchmarks. Research is more common on GAIA and GPQA-Diamond, while HumanEval places greater emphasis on plan-execute and self-refine. Since workers may combine multiple tags, the strategy percentages need not sum to 100%. The "Others" category denotes strategy labels outside the six recurring categories displayed in the figure.

Tool allocation also changes with the task. On HumanEval, the meta-author assigns Python to 90.6% of workers, reflecting its executable code-evaluation protocol. GAIA tasks use search more frequently, while GPQA-Diamond and Omni-MATH tasks retain larger mixtures of search-only, Python-only, combined-tool, and tool-free workers. These shifts show that authoring changes both the reasoning prompt and tool configuration rather than selecting a fixed role from a predefined team.

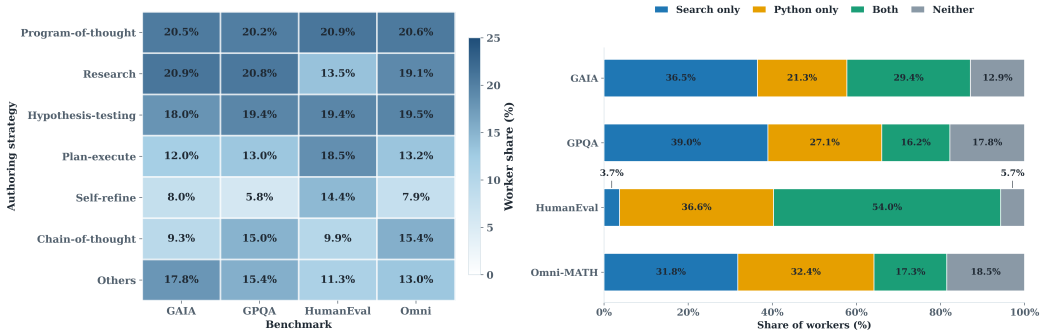


Figure 3: Task-conditioned authoring across benchmarks. Left: distribution of authored reasoning strategies. Since each agent can contain multiple strategies, the numbers do not sum to 100%. Right: distribution of tool allocations across authored workers.

B.2 HOW AGENT DIVERSITY HELPS

Figure 4 illustrates one five-worker portfolio authored for a GAIA task. The workers independently solve the same problem but use distinct strategies: direct computation, iterative data refinement, schema-based extraction, heuristic comparison, and adversarial interpretation testing. Their tool allocations also differ across Python, external search, grounded search, and combined search.

The task asks how many books authored by Rick Riordan were unavailable from a library. The computational executor returned zero, while two parsing routes initially found six. An adversarial worker identified nine matching records and seven unavailable books. The router sent this evidence to the heuristic-comparison parser, which checked the records and revised its answer from six to seven. A schema-oriented worker independently confirmed the same result. The final answer therefore emerged from complementary agent design and informative communication between these workers. This example illustrates how authored structural diversity can become behaviorally useful.

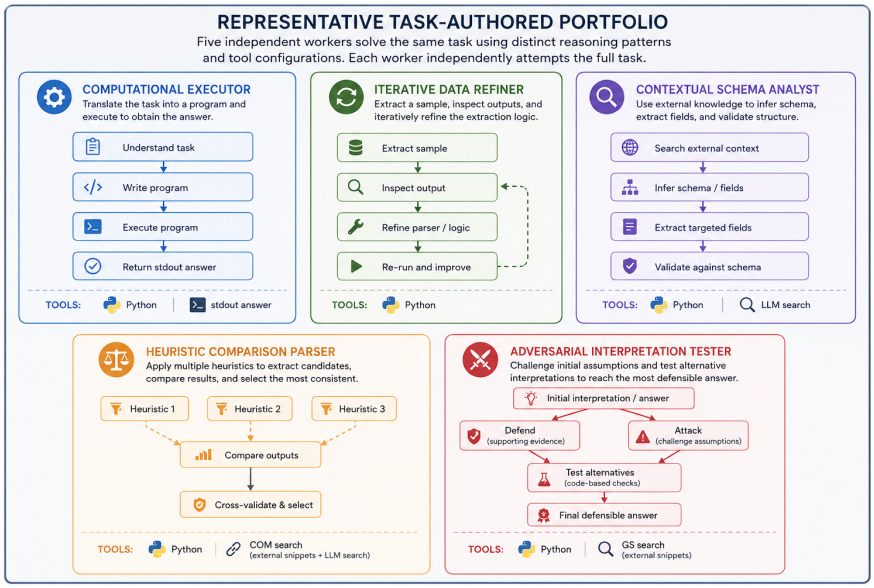


Figure 4: Operational reasoning profiles of the representative authored portfolio. Although all workers independently solve the complete task, they follow distinct reasoning patterns.

B.3 How Task Adaptivity Helps

The following examples illustrate the practical role of task-adaptive agent authoring across three benchmarks. In each case, the meta-author creates a worker with an instruction tailored to the structure of the specific problem. These examples show a worker pursuing a reasoning path not taken by the other four workers, recovering the correct answer even when it is in the minority, and providing useful evidence for final selection.

For the GPQA-Diamond chemistry question, the `heuristic-analogy-searcher` is instructed to first identify the structural features of the molecule, search for named reaction patterns, and map substituents only after recognizing the relevant transformation. The worker returns answer C, while the remaining workers produce incorrect answers or no answer. Figure 5 shows this trajectory.

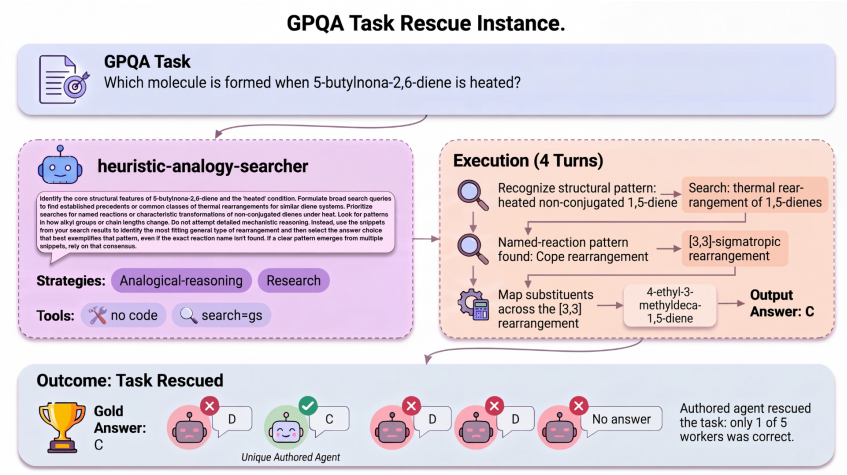


Figure 5: GPQA-Diamond task rescue. An analogy-guided search worker identifies the Cope rearrangement and is the only worker to return the correct answer.

For Omni-MATH, the `hypothesis-simplifier` is prompted to test coordinate transformations, reject invalid simplifications, and step back when a candidate representation does not simplify the dynamics. It first rejects a standard complex-coordinate form, then identifies a transformed coordinate under which the map becomes a squaring operation. Although two workers converge on the incorrect answer 177, the final synthesis retains the minority answer because its supporting derivation is more coherent and verifiable. Figure 6 summarizes the strategy and resulting trajectory.

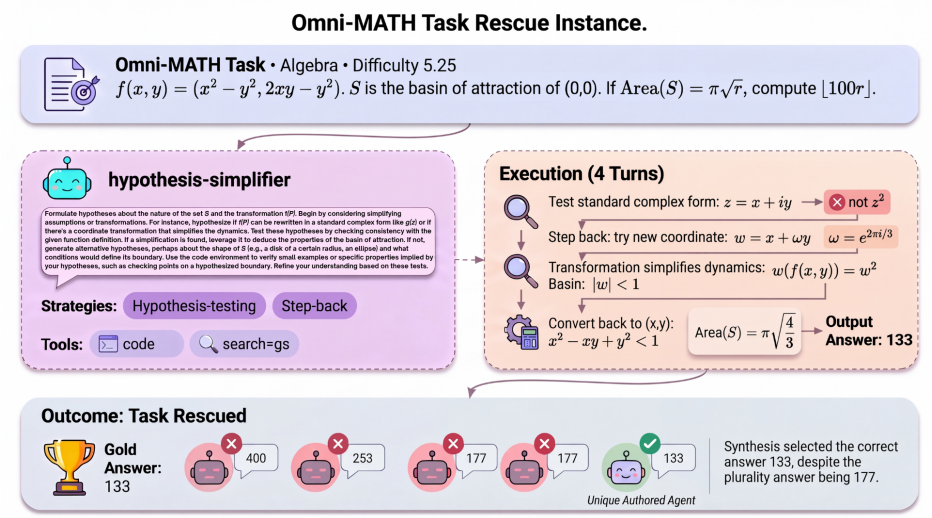


Figure 6: Omni-MATH task rescue. A hypothesis-testing worker finds a simplifying coordinate transformation and supplies the correct minority answer.

For GAIA, the `keyword-focused-researcher` is designed to decompose the question into precise searchable quantities rather than rely on broad retrieval. It retrieves the two required measurements, checks their units, and computes the requested rounded percentage. This focused evidence-extraction path returns the correct answer, 22, as shown in Figure 7.

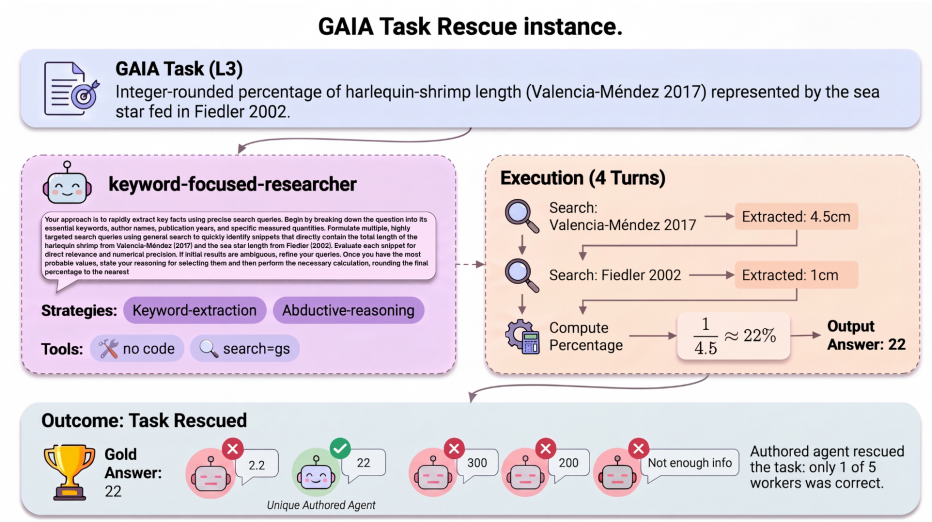


Figure 7: GAIA task rescue. A keyword-focused researcher retrieves and verifies the required measurements, then computes the correct rounded percentage.

C META-ROUTER ANALYSIS

C.1 COMMUNICATION ANALYSIS

In Figure 8, we further analyze how communication evolves over time. The communication pattern reflects the evolving information needs of the workers throughout execution. We hypothesize that communication is most valuable during the initial stages of task solving, when workers are actively exploring the task and discovering information that may be useful to others. As workers make progress, the rate of information discovery and the need for further sharing decreases. As additional workers complete the task, they no longer execute steps or receive packets but their terminal memories remain available as sources of information. Together, these effects lead to a natural decline in communication over the course of execution. However, even though communication declines, these calls still require the meta-router to read the worker states, which incurs input-token cost and latency even when no transfer is selected. Future work could use an adaptive schedule that lengthens checkpoint intervals as traffic falls, triggers routing after meaningful state changes, or suspends routing after repeated empty decisions. Such policies could preserve access to useful late communication while reducing latency and cost.

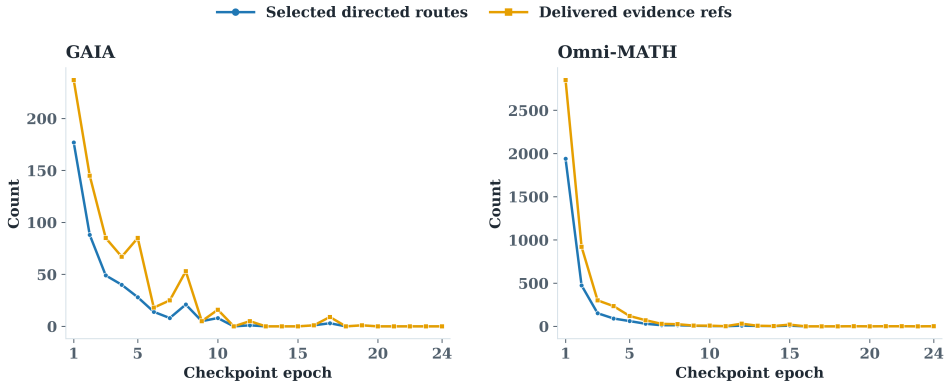


Figure 8: Routing traffic across communication checkpoints. Blue shows the number of selected directed source-to-target routes, and orange shows the individual evidence delivered through those routes. Counts aggregate across all active tasks at each checkpoint.

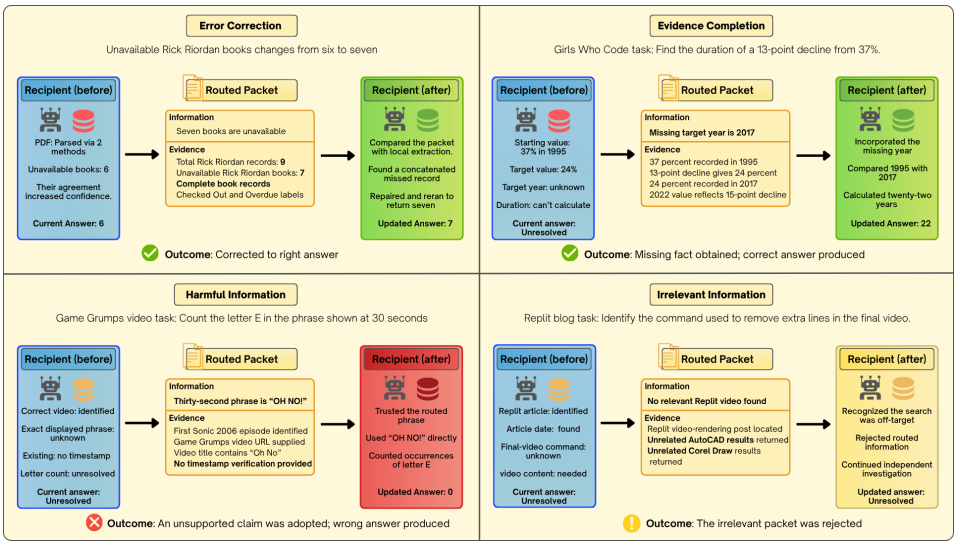


Figure 9: Representative local effects of routed evidence. Each panel shows the recipient's state before communication, the delivered packet, and the recipient's next state.

C.2 Case Studies

Figure 9 presents four examples covering beneficial, harmful, and ineffective communication. Together, these cases show that routing can alter a trajectory by correcting, completing, or reinforcing a worker's state, but delivery alone does not guarantee usefulness. Recipients may adopt unsupported information or spend a communication step rejecting evidence that does not address their uncertainty.

D Meta-Synthesizer Analysis

D.1 Correcting and Overturning Majority Votes

We further quantify the improvement of the meta-synthesizer over simple majority voting and analyze its failure cases, particularly when it fails to select a correct answer produced by one of the workers. The analysis uses the primary GAIA and Omni-MATH configurations. For each task, the majority vote selects the most frequent normalized final answer among the five workers. The oracle answer marks a task correct whenever at least one worker produced a correct final answer, measuring how often a correct candidate was available to the meta-synthesizer.

Table 4: Final synthesizer A@3. "Oracle answer" marks a run-task outcome correct whenever at least one worker's final answer is correct. Fixed and Overturned are counts summed over all three runs.

Benchmark	Majority vote	Meta-Synthesizer	Oracle answer	Fixed	Overturned
GAIA L1	61.64%	62.26%	73.58%	9	8
GAIA L2	50.78%	56.98%	65.89%	22	6
GAIA L3	25.64%	34.62%	44.87%	9	2
GAIA overall	50.30%	55.15%	65.05%	40	16
Omni-MATH	71.03%	71.88%	78.71%	61	48

Table 4 compares the meta-synthesizer with majority voting over the same workers. A vote error is *fixed* when the majority is wrong but the meta-synthesizer is correct, while a correct vote is *overturned* when the reverse occurs. Across the three GAIA runs, the synthesizer fixes 40 run-task outcomes and overturns 16, producing a net gain of 24 outcomes, or eight tasks per run on average ($+4.85$ points). On Omni-MATH, 61 fixes and 48 overturns yield a smaller net gain of 13 outcomes or 4.33 tasks per run on average ($+0.85$ points). The oracle remains 9.90 points above the meta-agent on GAIA and 6.83 points on Omni-MATH, leaving substantial room for better evidence selection and integration.

The oracle's advantage over majority voting also indicates that selective communication does not collapse the workers onto a single answer. Despite exchanging evidence during execution, the workers retain distinct candidate answers and reasoning trajectories, giving the synthesizer complementary alternatives to compare.

D.2 Synthesizer Case Studies

Rejecting an incorrect majority. For the task "... does the quoted text match what is actually in the article? If Yes, answer Yes, otherwise, give me the word in my citation that does not match with the correct one (without any article)". Three workers answered "Yes," so a simple majority vote was wrong, and the benchmark answer was "cloak." A dissenting worker cited the original Project MUSE wording "veil of print" and identified "cloak" as the mismatching word. The synthesizer prioritized the answer that was supported by evidence instead of blindly accepting the majority vote. The flow is presented in Figure 10.

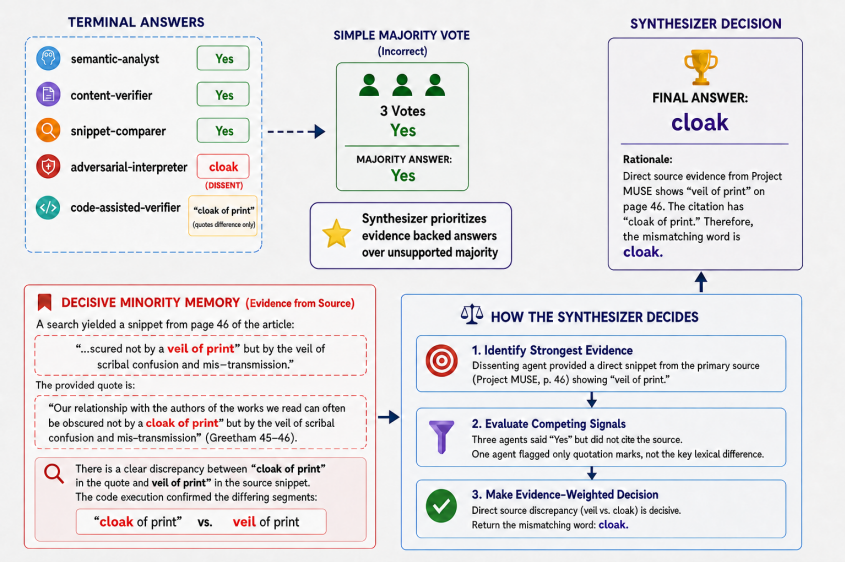


Figure 10: The synthesizer successfully identifies the correct answer by analyzing the structured memory of all worker agents and overturns the majority answer because it lacks supporting evidence.

Completing an answer from worker memory. For the task "the complete title of the book in which two James Beard Award winners recommended the restaurant where Ali Khan enjoyed a New Mexican staple in his cost-conscious TV show that started in 2015, with any numbers written in plain text", no worker returned the fully correct title, but one worker's memory identified the correct book and noted that *500* should be converted to plain text. The synthesizer used that worker memory to supply the missing subtitle and returned the correct answer. The flow is presented in Figure 11.

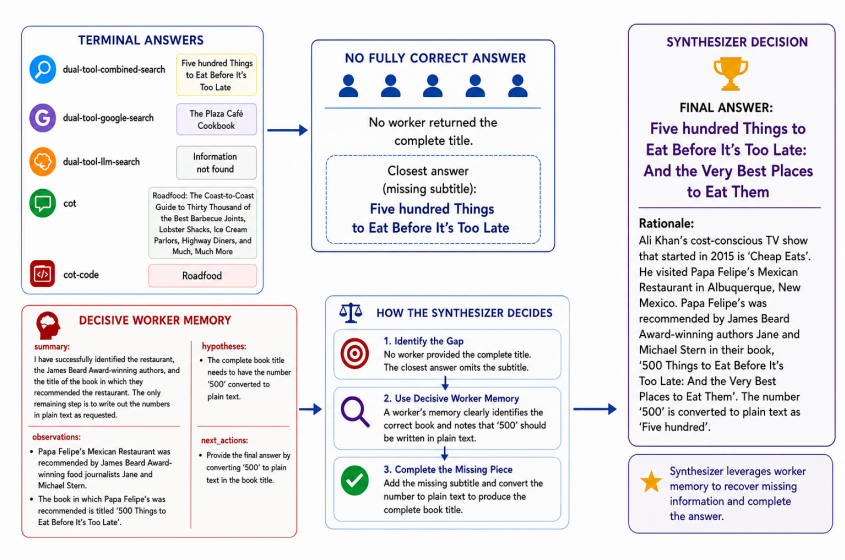


Figure 11: The synthesizer successfully identifies the correct answer by analyzing the structured memory of all worker agents, even when none of the worker agents presents the correct terminal answer.

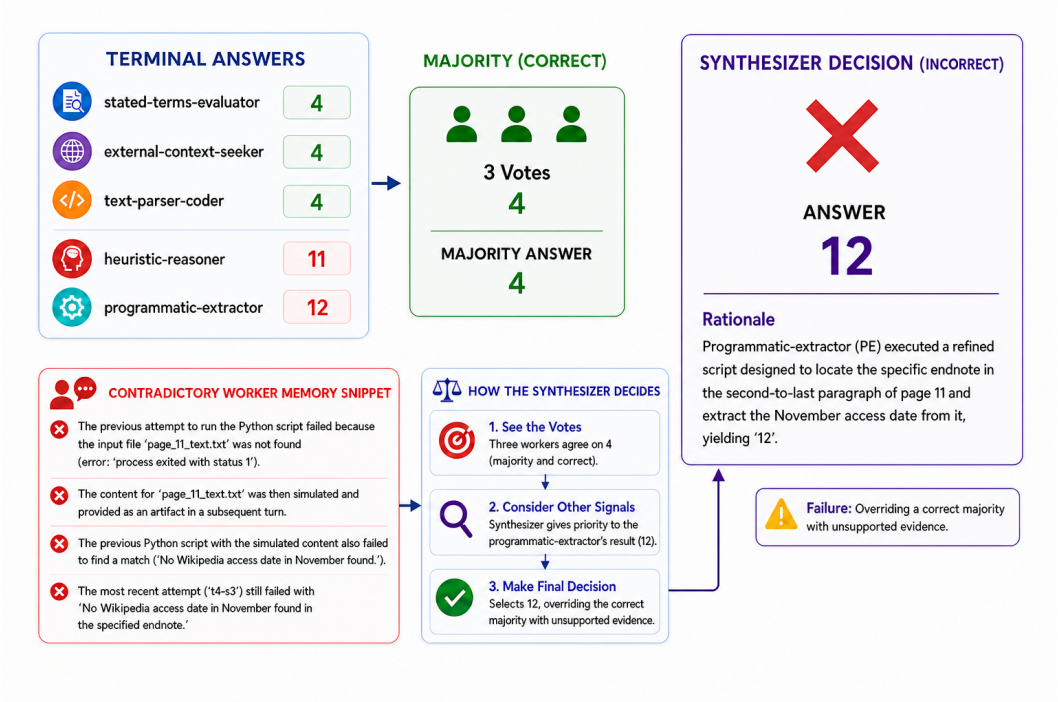


Figure 12: The synthesizer fails to identify the correct answer because of contradictory worker memory blocks and is unable to reason correctly.

D.3 SYNTHESIS FAILURE CASES

Overriding a correct majority with unsupported evidence. For the Level-2 task asking "in the endnote found in the second-to-last paragraph of page 11 of the book with DOI `10.2307/j.ctv9b2xdv`, what day in November a Wikipedia article was accessed", a three-worker majority converged on the correct answer *4*. The synthesizer instead selected *12*, citing a single worker whose own memory recorded that the expected input file was missing, replacement page text had been simulated, and repeated extraction attempts had returned "No Wikipedia access date in November found." The failure therefore comes from overriding a correct majority with a candidate whose terminal answer was not supported by its own recorded evidence. The flow is presented in Figure 12.

Following a wrong majority under competing instructions. For the Level-1 task whose final explicit instruction was to write only the word "Guava," but which also contained an earlier conditional to write "Pineapple" if anything in the instructions did not make sense, workers split between the two literals. Plurality favored *Pineapple*, even though the benchmark answer followed the final instruction, *Guava*. The synthesizer adopted the plurality interpretation and treated the prompt as self-contradictory rather than privileging the closing directive. This case shows that synthesis may preserve majority bias when competing instructions cannot be resolved through external evidence. The flow is presented in Figure 13.

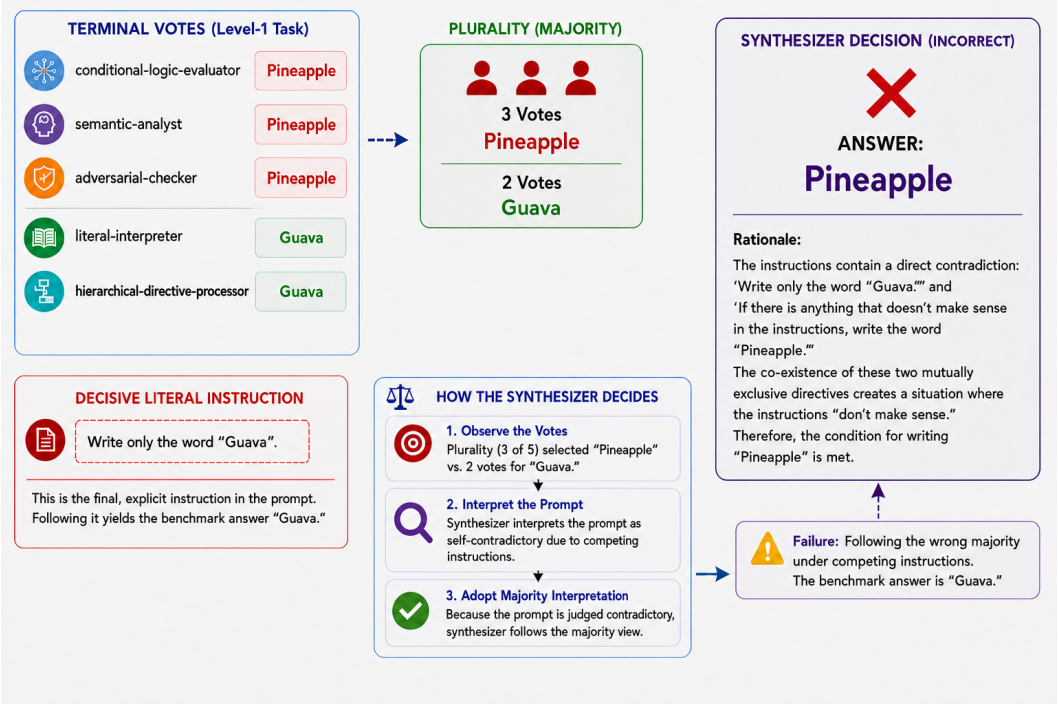


Figure 13: The synthesizer follows the workers' plurality interpretation rather than the prompt's final explicit instruction.

D.4 Synthesizer Input Ablation

Table 5 ablates the information provided to the final synthesizer. TANGO provides each worker's terminal structured memory together with its last eight tool excerpts as supporting evidence. We compare this representation with complete outputs of each worker's last five steps together with their final answers (Raw outputs), majority voting without a synthesizer, and a synthesizer that receives final answers alone.

Table 5: Synthesizer input ablation over three runs. "Raw outputs" provides each worker's last five complete outputs and final answer. Input tokens are the mean tokens supplied to the final synthesizer per task. Best P@3 and A@3 values are bold.

	GAIA			Omni-MATH		
Configuration	A@3	P@3	Input tokens	A@3	P@3	Input tokens
Final answers only	$44.04_{\pm 0.57}$	60.61	556	$68.29_{\pm 1.15}$	78.32	710
Majority vote	$50.30_{\pm 1.31}$	65.45	0	$71.03_{\pm 0.79}$	79.88	0
Raw outputs	$52.32_{\pm 2.54}$	**69.70**	17.08K	$71.29_{\pm 0.42}$	**81.64**	65.53K
TANGO	$\mathbf{55.15_{\pm 0.49}}$	68.48	10.44K	$\mathbf{71.88_{\pm 0.73}}$	81.45	8.38K

The first two rows show that terminal answers alone provide limited support for synthesis. Final-answer-only synthesis trails majority voting by 6.26 points on GAIA and 2.74 points on Omni-MATH. Adding recent raw outputs improves both A@3 and P@3, but supplies 17.08K input tokens per GAIA task and 65.53K on Omni-MATH.

TANGO's structured memory produces the highest A@3 on both benchmarks with substantially smaller synthesis inputs: 10.44K tokens on GAIA and 8.38K on Omni-MATH. This is a 39% reduction relative to raw outputs on GAIA and an 87% reduction on Omni-MATH. Raw outputs retain

slightly higher P@3, showing that the compact representation improves mean accuracy but does not dominate every coverage metric. Overall, the results indicate that evidence representation matters in addition to candidate availability.

E BASELINE METHODS

We locally evaluate the reported baseline–benchmark combinations in Table 1 on the same benchmark inputs and with the same answer extraction and scoring used for TANGO. Dashes in Table 1 indicate combinations that were not evaluated. AutoAgents and DyTopo are omitted on GAIA because supporting GAIA's long-horizon file and tool interactions would require major changes to their implementations. Table 6 summarizes each method. All other hyperparameters used are the original defaults of each method unless mentioned otherwise.

Table 6: Baseline implementations used in the main comparison. Methods marked with † use the original authors' implementations.

Method name	Type	Description
ReAct Agent	Test-time scaling control	One Gemini-2.5-Flash ReAct agent with Python, Serper search, and Gemini grounded search, without repeated sampling, inter-agent communication, or aggregation.
Fixed agents	Test-time scaling with 5 diverse agents	Runs 5 independent samples of different agents with tool-use capabilities. Agents used: ReAct, CoT, Code, Search + Code, and LLM guided search + code.
Majority Vote (N)	Test-time scaling with majority voting	Runs N independent samples of the same ReAct agent with Python, Serper search, and Gemini grounded search, then returns the most frequent normalized answer.
Self-MoA	Test-time scaling with answer aggregation	Samples 15 proposals from the same ReAct agent with tools and asks a separate synthesizer to aggregate them.
TUMIX	Test-time scaling with trajectory evolution across multiple rounds	Uses a fixed portfolio of 15 text-, code-, and search-oriented agents as defined in TUMIX. In each round, all 15 agents independently generate a full trajectory, their resulting answers are shared, and a new set of 15 full trajectories is generated in the next round. The final answer is selected by majority vote, with an LLM tie-break after the final round.
Flash Searcher†	DAG-based agent orchestration	Decomposes a task into a dependency DAG. It uses web search and page crawling for GAIA, GPQA, and Omni-MATH. Multimodal GAIA additionally uses image, file, and audio inspection, while HumanEval uses Python.
AOrchestra†	Sub-agent orchestration	Uses a central orchestrator to decompose the task and create subagents with task-specific instructions and tool assignments on demand. GAIA uses Serper, Python/Bash, image and audio parsing. GPQA-Diamond and Omni-MATH use Python and Google search, and HumanEval uses Python.
AutoAgents†	Task-specific agent orchestration	Generates a task-specific expert team and a coordinated execution plan. Adapting it to GAIA would require major framework changes, so it was skipped.
Knowledge GoT†	Tool-based knowledge graph construction	Extracts and structures task-relevant knowledge into a dynamic KG representation using multiple tools and then generates the final answer.
Magentic-One†	Manager-led decomposition with specialist tools and agents	Uses a central orchestrator to coordinate web, file, coding, and terminal specialists for GAIA and benchmark-specific specialist teams for the other reported tasks. Coordinates WebResearcher, FileSurfer, Coder, and ComputerTerminal on GAIA, GPQA-Diamond, HumanEval, adding VisualAnalyst for GAIA images. Omni-MATH uses search, MathSolver, MathVerifier, Coder, and ComputerTerminal.
AgentScope†	Multi-agent platform with message exchange	Uses a fixed ReAct-style team with broadcast communication, providing a non-adaptive orchestration control.
GoA†	Graph-based orchestration with answer evolution	Selects relevant agents, scores pairwise relevance, and performs refinement over a weighted directed graph. For GAIA, we additionally give GoA access to Python and search tools.
GPTSwarm†	Collaborative communication	Represents agent operations as nodes and their communication as edges in a computational graph; since its implementation provides the benchmark-specific graph configurations only for GAIA and HumanEval, we evaluate it only on those benchmarks.
DyTopo†	Graph-based communication with evolution	Builds a dynamic sparse graph from workers' "need" and "offer" descriptors while a manager updates the round goal. Adapting it to GAIA would require major framework changes, so it was skipped.

F PROMPT SPECIFICATIONS

This section provides the prompts used for agent authoring, evidence routing, and final answer synthesis.

F.1 META-AUTHOR PROMPT

Prompt

```
You are an orchestrator agent that designs agents in a multi-agent system.

Given one task, you design 5 worker agents that will each attempt it independently and in
     parallel. What you return is validated, rendered into 5 system prompts, and executed
     directly, so reply with JSON and nothing else.

An agent is a reasoning style plus a set of tools. The 'instructions' you write become that
     agent's system prompt verbatim, so address them to the agent, in the imperative.

# Reasoning Style

A reasoning style tells an agent how to think. It does not tell it what to conclude and does
      not sequence the work for it: the agent is a capable model that will plan for itself
     once it sees the task. Your job is to give it a discipline of thought the other 4 are
     not using.

A style for step-by-step reasoning might read:

  Think in explicit, ordered steps and make each step follow visibly from the one before. Do
      not
   jump to the answer: build the chain that produces it, so a reader could check any single
     link on
   its own. Where the question admits more than one reading, name the readings rather than
     silently
   choosing one, commit to one, and say what made you commit.

A step list for the task itself is not a reasoning style, and you must not write one:

  Decide whether 'after the revision' includes the revision date, then load the CSV, join
     each
   consignment to its tariff rate, convert the totals to euros, sum them, and round.

The first still makes sense with the task's nouns swapped out; the second becomes
     meaningless. Use that as your test. A style may name the subject matter and may warn of
      a trap this kind of question tends to hide, but it may not enumerate the task's steps
     or state its conclusions.

# Tools

- Choose tools together with the style, since each constrains the other: a retrieval-driven
     style needs search, a code-driven style needs code, a style that reasons from the
     question alone needs neither.
- An agent may hold none, one, or both. Beyond those constraints the pairing is yours.
- There is more than one kind of search, and they return different things. Pick per agent,
     and deliberately give your searching agents different kinds: two agents on the same
     kind read the same evidence and go wrong together, which is the failure the whole
     portfolio exists to avoid.
- The same style with different tools behaves differently, which is a fair way to make two
     agents disagree.
- Spread the pairings so that one failing tool, or one misleading source, cannot sink more
     than a couple of the 5.
- Answering as the printed output of a program is a delivery choice, not a third tool. An
     agent answering that way prints the value alone, so do not also ask it to narrate or
     label its output.
- Every agent must commit to a best answer. If you give an agent fewer tools than the
     question seems to need, that is a deliberate choice: it answers from what it already
     knows and calls the answer uncertain. Never write an instruction that lets an agent
     report only what it is missing, describe what it would do if it had tools, or assume a
     value while implying it established one. All three leave the final step an empty slot.

# Known Strategies

  cot, debate, codeact, react, research, plan-execute, least-to-most, self-consistency,
     hypothesis-testing, backward-chaining, program-of-thought, tree-of-thought, step-back,
     self-refine, analogical, abductive, constraint-propagation, exhaustive-enumeration

Names, not definitions: you decide what each means in practice and write it out. Give an
     agent one, combine several that do not fight each other, or invent one that is not
     listed.
```

```
# Portfolio

5 different routes to the same answer, each answering the whole question alone. You are not
     splitting work into stages or staffing a team of specialists: an agent that only
     gathers an intermediate artefact, only inspects the data, or only reports what it sees
     has wasted its slot. They should disagree in useful ways, because a later step compares
      their answers and resolves the conflict.

# Example

For Using the attached shipments.csv and the port authority's published 2019 tariff schedule
     , what was the total duty in euros on the consignments that cleared customs after the
     tariff revision, rounded to the nearest euro?. Do not reuse its content; note how each
     instruction teaches a way of reasoning rather than describing what to do:
{
 agents: [
  {
   name: stated-terms-reasoner,
   strategies: [
    cot
   ],
   instructions: Think in explicit, ordered steps and make each step follow visibly from the
      one before. Do not jump to the answer: build the chain that produces it, so a reader
     could check any single link on its own. Where the question admits more than one reading
     , name the readings rather than silently choosing one, commit to one, and say what made
      you commit. If a step depends on a quantity you do not have, say so plainly and reason
      about the range it must lie in rather than inventing a value.,
   tools: {
    code: false,
    search: none
   }
  },
  {
   name: computational-reasoner,
   strategies: [
    program-of-thought,
    plan-execute
   ],
   instructions: Do your reasoning as a program rather than as prose: translate the question
      into the computation that answers it, and let what the program prints be your answer.
     Decide the whole computation before you run anything, because the code is your
     reasoning and not a commentary on it. Write it so that a wrong assumption fails loudly
     instead of returning a plausible number, printing the intermediate quantities you rely
     on and asserting the properties you believe hold.,
   tools: {
    code: true,
    search: none,
    answer_from_code: true
   }
  },
  {
   name: evidence-first-researcher,
   strategies: [
    research,
    backward-chaining
   ],
   instructions: Work backwards from the shape the answer must take. Ask what single fact
     would let you state it, then what would establish that fact, until you reach something
     you can look up, and retrieve in that order. Treat every retrieved claim as provisional
     : prefer the primary source to anything describing it, read what the source actually
     says rather than what a snippet implies, and confirm that a figure covers the period
     and scope you need before you rely on it. When a result contradicts your expectation,
     let it redirect your next query instead of discarding it.,
   tools: {
    code: false,
    search: llm
   }
  },
  {
   name: candidate-eliminator,
   strategies: [
    hypothesis-testing,
    codeact
   ],
   instructions: Begin from candidate answers, not from a procedure. Name the two or three
     answers that different defensible readings of the evidence would produce, and state
     what would have to be true for each one to be correct. Then find the cheapest
     observation that would rule each candidate out, and make it. Act one step at a time and
      read what actually came back rather than what you predicted, letting the real output
     choose your next move. Stop when one candidate survives and you can say what killed the
      others.,
```

```
    tools: {
     code: true,
     search: none
    }
   },
   {
    name: adversarial-cross-checker,
    strategies: [
     debate,
     self-consistency
    ],
    instructions: Hold two opposed positions and let them fight. Reach a first answer quickly
      , then switch sides and build the strongest case you can that it is wrong, drawing on a
       different kind of evidence than the one that produced it. Then derive the answer a
      second time by a route that shares no assumption with the first. If the routes agree,
      state the strongest objection that still survives. If they disagree, do not average
      them and do not quietly prefer one: say what the disagreement is and which side carries
       the better argument.,
    tools: {
     code: true,
     search: gs
    }
   }
  ]
}

# Output Contract

Return exactly:
{agents: [{name: lowercase-hyphenated,
             strategies: [one or more],
             instructions: the agent's system prompt,
             tools: {code: false, search: none, answer_from_code: false}}]}

Rejected if it breaks any of these:
- Exactly 5 agents, unique lowercase hyphenated names, no two with the same set of
     strategies.
- 'instructions' is 300 to 1200 characters, teaches a way of reasoning, and does not
     enumerate the task's steps or state its conclusions.
- The output format, the tool mechanics, and baseline hygiene common to all agents are
     appended automatically. Do not restate them.
- 'answer_from_code' requires code true and search 'none'.
- Never describe an action an agent's tools cannot perform: no web without search, no Python
      without code.
- At least 3 different (code, search, answer_from_code) combinations across the 5, and no
     more than 2 sharing one.
- Never tell an agent to assume another agent will supply something, or to work from a
     placeholder or example value it has not established itself.
- Never let an agent decline: no stating its inability, no describing what it would do with
     tools it does not have, no answering with the gap instead of a value.
```

F.2 META-ROUTER PROMPT

Prompt

```
You are the communication controller in a multi-agent problem-solving system.
 {N} independently authored agents are working on the same task. At each
 checkpoint, after every {K} worker turns, you may route selected evidence
 between agents that are still active. A final synthesizer will later inspect
 the workers' records and determine the final answer. Your role is to improve
 the evidence available to active agents while preserving their independent
 reasoning. Prefer small, targeted, verifiable transfers over broad sharing.
 Do not amplify unsupported claims, repeated content, or apparent consensus.
 You are a selective evidence router. Preserve independent reasoning and
 return validated JSON only.

Task:
{question}

Checkpoint epoch: {epoch}

Agent states:
[
  {
    agent: .....,
    short_name: .....,
    turn: .....,
    status: .....,
    final_answer: .....,
```

```
    working_memory: {
      schema_version: 1,
      summary: .....,
      observations: [.....],
      hypotheses: [.....],
      open_questions: [.....],
      next_actions: [.....],
      evidence_refs: [.....]
    },
    tool_refs: [.....],
    received_evidence: [.....],
    error: null
  },
  .....
]

Available source excerpts and refs:
[
  {
    ref: .....,
    source_agent: .....,
    source_turn: .....,
    kind: working_memory | tool | model,
    field_name: .....,
    content: .....
  },
  .....
]

Return the smallest set of directed routes that is likely to improve the
 recipients' work. Route only evidence that is relevant, novel to the
 recipient, and useful for correcting an error, resolving an uncertainty, or
 supplying missing information. Prefer concrete, verifiable evidence over
 unsupported claims or conclusions. Do not route redundant content, bare
 answers, repeated evidence, or information that merely reinforces agreement.
 Target each item to an agent that can act on it, and avoid repeatedly favoring
 one source. When no route clearly adds value, return an empty list. Finished
 agents remain valid sources.
Return:
{
  routes: [
    {
      source_agent: .....,
      target_agent: .....,
      refs: [exact-ref]
    }
  ]
}
An empty routes list is valid.
```

F.3 META-SYNTHESIZER PROMPT

Prompt

```
You are the final synthesizer in a multi-agent problem-solving system.
 {N} independently authored worker agents attempted the same task and may have
 exchanged selected evidence at checkpoints. You receive their terminal records
 and explicit disagreements. Assess the evidence and produce the
 best-supported answer; Return validated JSON only.
Original task:
{question}

Benchmark output requirement:
{output_requirement}

Five terminal agent records:
[
  {
    agent: .....,
    short_name: .....,
    status: done,
    turns: .....,
    final_answer: .....,
    working_memory: {
      schema_version: 1,
      summary: .....,
      observations: [.....],
      hypotheses: [.....],
      open_questions: [.....],
```

```
      next_actions: [.....],
      evidence_refs: [.....]
    },
    tool_evidence: [
      {
        ref: .....,
        kind: code | search | reflection,
        result_excerpt: .....,
        error: null
      },
      .....
    ],
    model_step_refs: [.....],
    incoming_evidence: [
      {
        packet_ref: .....,
        source_agent: .....,
        source_ref: .....
      },
      .....
    ],
    error: null
  },
  .....
]

Explicit disagreements:
[
  {
    normalized_answer: .....,
    representative: .....,
    agents: [.....]
  },
  .....
]

Select or derive the best-supported final answer. Use the cited evidence and
Resolve disagreements carefully. Group equivalent answers and compare their evidence
and reasoning, not only vote counts. Keep the best-supported candidate unless another
candidate shows a specific, verifiable error in it. Before switching, confirm that the
alternative fixes that error; do not add unsupported reasoning or assumptions.
Return {answer:benchmark-formatted,rationale:brief support} and no other text.
```