

000 001 002 003 004 005 006 007 008 009 010 011 012 013 014 015 016 017 018 019 020 021 022 023 024 025 026 027 028 029 030 031 032 033 034 035 036 037 038 039 040 041 042 043 044 045 046 047 048 049 050 051 052 053 AGENT*: OPTIMIZING TEST-TIME COMPUTE FOR MULTI-AGENT SYSTEMS WITH MODULARIZED COLLABORATION

Anonymous authors

Paper under double-blind review

ABSTRACT

Scaling test-time computation improves large language model performance without additional training. Recent work demonstrates that techniques such as repeated sampling, self-verification, and self-reflection can significantly enhance task success by allocating more inference-time compute. However, applying these techniques across multiple agents in a multi-agent system is difficult: there does not exist principled mechanisms to allocate compute to foster collaboration among agents, to extend test-time scaling to collaborative interactions, or to distribute compute across agents under explicit budget constraints. To address this gap, we propose AGENT*, a framework for *optimizing* test-time compute allocation in multi-agent systems under fixed budgets. AGENT* introduces *modularized collaboration*, formalized as callable functions that encapsulate reusable multi-agent workflows. These modules are automatically derived through self-play reflection by abstracting recurring interaction patterns from past trajectories. Building on these modules, AGENT* employs a *dual-level planning architecture* that optimizes compute allocation by reasoning over the current task state while also *speculating* on future steps. Experiments on complex agent benchmarks demonstrate that AGENT* consistently outperforms baselines across diverse budget settings, validating its effectiveness for multi-agent collaboration in inference-time optimization.

1 INTRODUCTION

Increasing inference-time computation (Snell et al., 2024; Muennighoff et al., 2025; Balachandran et al., 2025; Zhang et al., 2025b) has emerged as a powerful strategy for improving the performance of large language model (LLM) without additional training. Recent models, such as OpenAI’s o1 and o3 and Anthropic’s Claude Sonnet 3.7, have demonstrated strong reasoning capabilities through techniques such as self-correction (Madaan et al., 2023; Chen et al., 2025a), iterative verification (Lee et al., 2025), and best-of-N sampling (Brown et al., 2024). However, extending test-time scaling to multi-agent systems (OpenAI, 2025; Google, 2025; Anthropic, 2024; OpenAI, 2024) introduces unique challenges.

First, current test-time scaling techniques do not extend effectively to multi-agent systems. In single-agent settings, extra compute can be spent directly on repeated sampling, verification, or reflection. In contrast, multi-agent systems face the challenge of deciding how additional compute should be *allocated* across agents and interactions. Existing approaches, such as the widely adopted orchestrator-worker paradigm (Hadfield et al., 2025; Tran et al., 2025), decompose tasks and invokes agents sequentially, but this setup neither facilitates genuine collaboration nor encourages the system to invest extra compute in coordination. As a result, synergies between complementary agents remain underexploited, and execution is biased toward fixed patterns, limiting adaptability to varying task demands.

Second, there is no principled mechanism for maximizing performance under a fixed compute budget. Existing strategies, such as budget enforcement (Muennighoff et al., 2025), typically rely on static rules or heuristics. Such approaches cannot adjust compute allocation across inference steps to account for varying complexities, nor can they anticipate future compute demands. Anticipation

054 is essential for scaling, because deciding whether to scale compute for the current step depends on
 055 estimating how much is likely to be needed later. Without this foresight, systems risk overspend-
 056 ing early or under-investing when tasks become more challenging. This limitation is particularly
 057 acute in multi-agent systems, where compute must be strategically distributed across agents with
 058 heterogeneous capabilities and costs.

059 In this paper, we address the core question: *How can multi-agent systems maximize performance*
 060 *under fixed compute budgets while leveraging complementary agents capable of collaboration?* To
 061 answer this, we propose AGENT*, a general framework for budget-constrained optimization of test-
 062 time compute in multi-agent systems. At its core, AGENT* introduces the abstraction of *collab-*
 063 *oration modules* — modular, callable functions that encapsulate high-level coordination strategies
 064 among agents. Each module defines a structured multi-agent workflow with a standardized input
 065 schema and functionality, making agent interactions composable and reusable, similar to tools. Col-
 066 laboration modules provide a principled way to allocate and scale compute effectively, since the
 067 orchestrator can decide when and how to invoke more *compute-intensive forms* of collaboration
 068 depending on task state and budget. In this view, multi-agent collaboration reduces to a function-
 069 calling problem, with a meta-agent dynamically choosing which module to invoke. The modules
 070 themselves are automatically induced through self-play reflection on past trajectories, ensuring they
 071 capture reusable coordination patterns without manual design.

072 Having established reusable collaboration modules, the central challenge becomes how to allocate
 073 computation across them under budget constraints. To address this, AGENT* employs a *dual-level*
 074 *planning architecture* that integrates short-horizon and long-horizon planning. The short-horizon
 075 planning proposes candidate next actions, either invoking a collaboration module or an individual
 076 agent, conditioned on the current task state. In parallel, the long-horizon planning, inspired by spec-
 077 ulative decoding (Leviathan et al., 2023), performs high-level speculation by reasoning abstractly
 078 over potential sequences of collaboration modules instead of concretely executing them. This yields
 079 low-cost estimates of budget feasibility, enabling the system to reason about when additional com-
 080 pute should be saved or spent. Together, the two levels operate in an A*-like fashion, where the
 081 short-horizon planner prioritizes promising near-term actions and the long-horizon planner supplies
 082 a lookahead signal that aligns decisions with budget-aware trajectories.

083 To summarize, this paper makes the following contributions:

- 084 • We formulate the problem of optimizing test-time compute allocation for multi-agent systems
 085 under fixed budget constraints with a set of agents capable of collaboration.
- 086 • We propose AGENT*, a general framework for budget-aware multi-agent collaboration. AGENT*
 087 introduces *collaboration modules* to facilitate effective multi-agent collaboration, and employs
 088 a *dual-level planning architecture* that balances short-horizon action selection with long-horizon
 089 speculation to allocate compute effectively under budget constraints.
- 090 • We develop *self-play reflection* to collect cost estimates of agent and collaboration module execu-
 091 tions and automatically induce collaboration modules from recurring interaction patterns.
- 092 • We demonstrate that AGENT* consistently outperforms baselines across challenging multi-agent
 093 benchmarks, achieving higher task success rates and more effective budget utilization.

095 2 ORCHESTRATOR–WORKER FRAMEWORK UNDER BUDGET CONSTRAINTS

098 In this section, we introduce the general setting of orchestrator–worker framework (Hadfield et al.,
 099 2025; Tran et al., 2025) in multi-agent systems that operate under a fixed test-time compute budget.
 100 The goal is to solve an input task while ensuring total computation stays within budget.

101 Let $\mathcal{A} = \{a_1, \dots, a_N\}$ denote the set of available *worker agents*, each initialized with a large
 102 language model (LLM) with distinct capabilities and an associated cost of invocation. A task is
 103 solved through a sequence of intermediate states $\{s_t\}_{t=0}^H$, where s_0 is the initial task description and
 104 s_H is the final output returned by the system. At each step t , the orchestrator selects an *action* α_t

$$105 \quad \alpha_t = (a_t, v_t), \quad a_t \in \mathcal{A},$$

107 where v_t specifies the subtask or input to the chosen worker. Executing α_t produces an output o_t ,
 and the new state is defined as $s_t = (v_t, o_t)$. Each action incurs a cost $\text{cost}(\alpha_t)$, and the cumulative

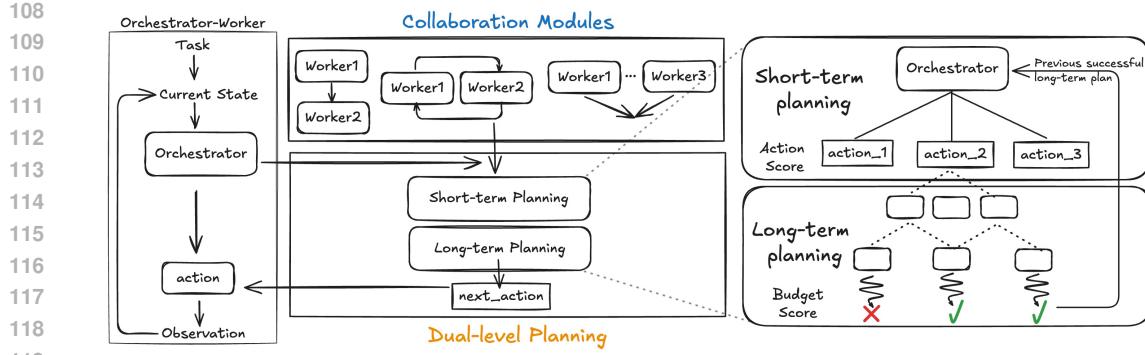


Figure 1: Overview of AGENT*. The framework extends the standard orchestrator–worker paradigm by introducing *collaboration modules*, which encapsulate reusable multi-agent workflows, and a *dual-level planning architecture*, which integrates short-term and long-term planning to select the most promising next action.

cost is constrained by the budget

$$\sum_{t=1}^H \text{cost}(\alpha_t) \leq B.$$

The system’s behavior at step t is governed by a policy π that selects the next action based on the execution history $\mathcal{H}_t = \{s_r\}_{r=0}^{t-1}$.

We instantiate the policy π with a LLM (Yao et al., 2020; Huang et al., 2022; Yao et al., 2023), which serves as the backbone for the orchestrator agent. Following the ReAct framework (Yao et al., 2023), the orchestrator generates subtasks, assigns them to worker agents, evaluates their outputs, and determines subsequent actions until a final solution is produced or the budget is exhausted.

3 AGENT*

Multi-agent systems built on the orchestrator–worker paradigm (Hadfield et al., 2025; Tran et al., 2025) suffer from limited coordination and inefficient use of additional compute. Synergies between agents are underexploited, and budget-aware optimization has not been well studied. To address these issues, we propose AGENT*, which extends the orchestrator–worker framework with two key components.

At the core of AGENT* are collaboration modules (§3.1), that are modular functions encapsulating reusable coordination strategies and transforming multi-agent interaction into a structured function-calling problem. Building on this abstraction, AGENT* employs a dual-level planning architecture (§3.2) that balances short-horizon action selection with long-horizon speculation, enabling adaptive and forward-looking compute allocation under strict budget limits. An overview is shown in Figure 1.

3.1 COLLABORATION MODULES

The traditional orchestrator–worker framework is constrained by sequential, one-agent-at-a-time execution. This design becomes inefficient on complex, open-ended tasks, as it is difficult to capture synergies between agents with complementary capabilities. In particular, the sequential setup limits coordination, since intermediate results are not jointly integrated across agents, leaving sub-tasks only partially addressed. Moreover, this rigid design restricts how additional compute can be used: allocating more resources to a single agent, through repeated sampling or iterative verification, merely amplifies that agent’s behavior without fostering cross-agent collaboration. Unlike test-time scaling in single-agent LLMs, such additional compute does not provide a principled path to improving performance in multi-agent systems.

To address these limitations, we introduce *collaboration modules* — modular, callable functions that encapsulate high-level coordination strategies among multiple agents. Each module specifies

a standardized and reusable workflow, such as combining outputs from multiple agents or chaining agents in a pipeline, and can be invoked with a single function call. This abstraction reframes multi-agent collaboration as a function-calling problem, where the orchestrator decides which collaboration module to invoke at each step. By structuring interactions in this way, collaboration modules enable richer coordination and provides a scalable and principled mechanism for utilizing test-time compute more effectively, as additional budget is allocated to cross-agent collaboration rather than simply amplifying the behavior of individual agents.

Formally, we define a collaboration module as

$$m = (\mathcal{S}, \kappa), \quad \mathcal{S} \subseteq \mathcal{A},$$

where \mathcal{S} denotes a subset of worker agents and κ is a coordination strategy that specifies how these agents interact. The action space available to the orchestrator is therefore expanded to include not only the individual base agents but also collaboration modules:

$$\mathcal{A}' = \mathcal{A} \cup \mathcal{M},$$

where \mathcal{A} is the set of base worker agents and \mathcal{M} is the set of collaboration modules. From this point onward, we assume that a_t in action α_t is drawn from \mathcal{A}' , enabling the orchestrator to invoke either a single agent or a collaboration module at each step.

3.2 DUAL-LEVEL PLANNING ARCHITECTURE

While collaboration modules provide a structured abstraction for multi-agent coordination, they also introduce the challenge of how to allocate computation effectively under budget constraints. Distributing test-time resources across modules is inherently uncertain, since the system cannot know in advance which modules will be invoked or how their interactions will unfold. This uncertainty motivates the need for a planning mechanism that adaptively determines which modules to invoke and how to allocate computation across them, ensuring resources are used efficiently to maximize task performance under strict budget constraints.

Inspired by traditional A* search (Hart et al., 1968; Meng et al., 2024), AGENT* frames planning as the exploration of a search tree, where each node represents a decision point at step t for invoking an action $\alpha_t = (a_t, v_t)$, evaluated by the cumulative gain $g(\alpha_t)$ and the future gain $h(\alpha_t)$ ¹. Analogously, in AGENT*, short-term planning plays the role of $g(\alpha_t)$: it expands candidate next actions based on the policy $\pi(\cdot | \mathcal{H}_t)$ and assigns each a score reflecting the short-term gain. Long-term planning corresponds to $h(\alpha_t)$: it speculates over possible future trajectories to estimate whether subsequent steps will remain feasible under the budget. By combining these two levels, the orchestrator effectively selects the candidate with the highest overall utility score $f(\alpha_t) = g(\alpha_t) + h(\alpha_t)$, ensuring that immediate actions are consistent with budget-aware long-term feasibility.

3.2.1 SHORT-TERM PLANNING

In short-term planning, AGENT* evaluates the immediate utility of a candidate action $\alpha_t = (a_t, v_t)$ proposed by the policy π . At each step t , the policy generates K candidate actions by drawing from the distribution

$$\mathcal{C}_t = \{\alpha_t^{(1)}, \dots, \alpha_t^{(K)}\} \sim \pi(\cdot | \mathcal{H}_t, \mathcal{T}_{\text{feasible}, t-1}),$$

where $\mathcal{T}_{\text{feasible}, t-1}$ denotes the set of budget-feasible speculative trajectories carried over from the previous step. This conditioning ensures that short-term proposals are informed by speculated budget-feasible high-level plans rather than sampled solely from \mathcal{H}_t . As a result, the candidate set reflects not only the current task context but also long-term budget feasibility, yielding proposals that are both contextually grounded and more likely to lead to successful completions within budget.

To assess these candidates, we compute a self-consistency score (Wang et al., 2022), which serves as a proxy for their effectiveness. Formally, let $\phi((a, v)) = a$ extract the agent or module from an action. The short-term gain is defined as

$$g(\alpha_t^{(i)}) = \frac{1}{K} \sum_{k=1}^K \mathbf{1}[\phi(\alpha_t^{(k)}) = \phi(\alpha_t^{(i)})],$$

¹Here, cumulative and future gains are interpreted as the inverse of cumulative and future costs in A*.

216 where $\mathbf{1}[\cdot]$ is the indicator function. Actions with higher self-consistency receive larger $g(\alpha_t^{(i)})$ values,
 217 indicating stronger evidence that they will contribute effectively to task progress. This mech-
 218 anism provides a lightweight yet reliable signal for prioritizing near-term actions before long-term
 219 feasibility is considered.
 220

221 3.2.2 LONG-TERM PLANNING

223 While short-term planning evaluates the immediate promise of an action, long-term planning specu-
 224 lates on its feasibility under the remaining budget. The goal is to compute a budget feasibility score
 225 $h(\alpha_t^{(i)})$ that captures whether choosing $\alpha_t^{(i)}$ as the next action is likely to keep future trajectories
 226 within budget, without actually executing $\alpha_t^{(i)}$ and following actions.
 227

228 Concretely, the orchestrator agent expands each candidate action into *speculative trajectories*, which
 229 are abstract rollouts representing possible continuations of agents or collaboration modules. These
 230 trajectories are generated symbolically at the module level, so no agents are invoked, keeping the
 231 procedural lightweight. Each trajectory is associated with an estimated cumulative cost, and any that
 232 exceed the remaining budget are filtered out. Formally, let $\mathcal{T}_t(\alpha_t^{(i)})$ denote the set of speculative tra-
 233 jectories beginning with $\alpha_t^{(i)}$, and let $\text{cost}(\tau)$ represent the estimated cumulative cost of a trajectory
 234 $\tau \in \mathcal{T}_t(\alpha_t^{(i)})$. Given the remaining budget B_t , we define the set of feasible trajectories as
 235

$$\mathcal{T}_{\text{feasible},t}(\alpha_t^{(i)}) = \{\tau \in \mathcal{T}_t(\alpha_t^{(i)}) \mid \text{cost}(\tau) \leq B_t\},$$

236 so that trajectories whose costs exceed B_t are filtered out. Among the feasible trajectories, we then
 237 normalize across the K candidate actions sampled at step t . This defines the budget feasibility score:
 238

$$h(\alpha_t^{(i)}) = \frac{|\mathcal{T}_{\text{feasible},t}(\alpha_t^{(i)})|}{\sum_{r=1}^K |\mathcal{T}_{\text{feasible},t}(\alpha_t^{(r)})|}.$$

242 Intuitively, $h(\alpha_t^{(i)})$ reflects the likelihood that an action can be extended into a successful budget-
 243 compliant plan. Actions with higher $h(\alpha_t^{(i)})$ values are favored, since they not only appear promising
 244 in the short term but are also more likely to sustain progress without violating budget constraints.
 245 This speculative lookahead ensures that immediate choices remain consistent with long-term feasi-
 246 bility, complementing the short-term gain $g(\alpha_t^{(i)})$ in the overall utility $f(\alpha_t^{(i)}) = g(\alpha_t^{(i)}) + h(\alpha_t^{(i)})$.
 247 Moreover, the resulting feasible speculative set $\mathcal{T}_{\text{feasible},t}$ is carried forward to guide candidate gen-
 248 eration in the next step’s short-term planning.
 249

250 Finally, we select the candidate action with the highest utility score as the next action. The dual-level
 251 planning procedure is formally presented in Alg. 1.
 252

253 3.3 SELF-PLAY REFLECTION

254 Dual-level planning requires estimates of the execution cost for both agents and collaboration mod-
 255 ules in order to conduct long-term planning. Also, for the collaboration modules, we need effective
 256 ways to structure multi-agent collaborations. To address both needs, AGENT* employs *self-play re-
 257 flection*, an *iterative* process that builds experience by generating execution trajectories on a subset
 258 of validation tasks and uses this experience both to compute the average cost of invoking each agent
 259 or module and to automatically construct collaboration modules from recurring interaction patterns.
 260

261 Formally, given the set of actions \mathcal{A}' , the system executes multiple trajectories and logs the agents
 262 invoked, subtasks addressed, outputs produced, and costs incurred. The average execution cost is
 263 then estimated as
 264

$$\widehat{\text{cost}}(a) = \mathbb{E}_{\text{traj},v} [\text{cost}(a, v)], \quad a \in \mathcal{A}',$$

265 providing the statistics required for long-term planning. While in theory the cost of invoking an
 266 agent or module depends on both a and the specific subtask v , enumerating all possible subtasks
 267 is infeasible. We therefore relax this assumption and approximate the cost by averaging across
 268 observed subtasks in self-play trajectories, treating the resulting estimate as transferable across tasks.
 269

In parallel, successful trajectories are reflected upon by an LLM to identify recurring sequences of
 agent interactions that consistently contribute to successful task completion. These sequences are

270 abstracted into reusable collaboration modules, which are added to the action space \mathcal{A}' and refined
 271 through further rounds of self-play. Over successive iterations, this process yields a growing library
 272 of collaboration modules as well as reliable cost profiles for both modules and base agents.

273 Through self-play reflection, AGENT* unifies cost estimation and collaboration module discovery
 274 in a single data-driven procedure, eliminating the need for manual engineering and enabling more
 275 effective budget-optimal planning.

278 4 EXPERIMENTS

280 To evaluate the effectiveness of AGENT* in multi-agent settings, we conduct experiments on the
 281 GAIA and BrowseComp-Plus benchmarks, comparing its performance against baselines under strict
 282 budget constraints.

284 4.1 EXPERIMENT SETTINGS

286 **Dataset.** We evaluate AGENT* on two agent benchmarks: (1) GAIA (Mialon et al., 2023): a real-
 287 world QA benchmark that evaluates web browsing and general tool-use ability of language models;
 288 and (2) BrowseComp-Plus (Chen et al., 2025b): a benchmark derived from BrowseComp (Wei et al.,
 289 2025) that measures the ability for agents to browse the web using a fixed, curated corpus.

291 **Agents and Collaboration Modules.** For each benchmark, we first instantiate a set of task-
 292 specialized agents. In GAIA, we utilize four agents: Search Agent, Browser Agent, Reasoning
 293 Agent, and Media Inspector Agent. In BrowseComp-Plus, we use three agents: Retriever Agent,
 294 Document Reader Agent, and Critic Agent.

295 To construct collaboration modules, we conduct five rounds of self-play reflection for each bench-
 296 mark, with each round producing one candidate module. We utilize the first 30 questions in the
 297 benchmark as a validation set to collect cost estimates and collaboration modules. This process
 298 yields mainly five distinct modules for GAIA: *interactive search and browse*, *search then browse*,
 299 *ensemble search*, *two ensemble reasoning*, and *three ensemble reasoning*. For BrowseComp-Plus, it
 300 yields four unique modules: *interactive search*, *ensemble interactive search*, *interactive search then*
 301 *critic*, and *ensemble interactive search then critic*, with the final round producing a duplicate².

303 **Models.** We employ two model families: Claude (Anthropic, 2025) and Qwen3 (Yang et al.,
 304 2025). Within the Claude family, we use Claude-3.7-Sonnet for the orchestrator, reasoning,
 305 critic, and media inspector agents, and Claude-3.5-Haiku for the rest. For the Qwen family,
 306 we use Qwen3-32B for all the agents.

308 **Budget Constraints.** For the unit of cost and budget, we compute the monetary cost of each action
 309 by multiplying its input and output token usage with the official token pricing from AWS Bedrock³.
 310 We evaluate performance under four budget settings. To determine suitable constraints for each
 311 benchmark and model, we first measure the trajectory costs from self-play reflection and take their
 312 average as the minimum budget. Larger budgets are then defined either by fixed increments or by
 313 exponential scaling. For Claude experiments, we allocate per-question budgets of \$0.2, \$0.3, \$0.4,
 314 and \$0.5 across both benchmarks. For Qwen experiments, we set budgets of \$0.05, \$0.1, \$0.2, and
 315 \$0.3 for GAIA, and \$0.025, \$0.05, \$0.1, and \$0.2 for BrowseComp-Plus.

316 **Metrics.** We report results using $\text{Acc}@B$, which denotes the accuracy achieved under a budget
 317 B . This metric reflects the proportion of questions answered correctly while ensuring that the total
 318 token-based compute cost does not exceed B for each query. $\text{Acc}@B$ enables fair comparison across
 319 different compute regimes and highlights how effectively each method converts its allotted budget
 320 into correct answers.

322 ²We provide the prompt for self-play reflection in Figure 3 and detailed descriptions of agents and collabora-
 323 tion modules in Appx. §A

324 ³We referred to <https://aws.amazon.com/bedrock/pricing/>. The actual token price we used is in Appx. §B

Claude Models								
Method	GAIA				BrowseComp-Plus			
	Acc@0.2	Acc@0.3	Acc@0.4	Acc@0.5	Acc@0.2	Acc@0.3	Acc@0.4	Acc@0.5
<i>Fixed Agent Workflow</i>								
ADAS	11.72	11.72	11.72	11.72	6.20	6.20	6.20	6.20
AFlow	12.96	12.96	12.96	12.96	5.42	5.42	5.42	5.42
<i>ReAct with test-time scaling</i>								
ReAct	35.80	35.80	35.80	35.80	24.33	24.66	24.66	24.66
w/ Best-of-N	35.80	35.80	37.03	37.65	24.33	24.80	24.80	24.80
w/ Iterative Verification	35.80	36.41	38.27	37.03	24.54	25.06	25.58	24.80
<i>ReAct with Collaboration Modules</i>								
Budget-Unaware	35.80	37.65	38.27	38.27	25.32	25.50	27.08	26.20
Budget-Aware Prompt	36.41	41.97	43.20	43.20	24.07	26.34	27.37	28.07
AGENT*	38.89	44.44	46.91	48.15	25.53	27.50	28.07	29.00
Qwen3-32B								
Method	GAIA				BrowseComp-Plus			
	Acc@0.05	Acc@0.1	Acc@0.2	Acc@0.3	Acc@0.025	Acc@0.05	Acc@0.1	Acc@0.2
<i>Fixed Agent Workflow</i>								
ADAS	4.93	4.93	4.93	4.93	2.84	2.84	2.84	2.84
AFlow	3.70	3.70	3.70	3.70	3.35	3.35	3.35	3.35
<i>ReAct with test-time scaling</i>								
ReAct	12.66	12.66	12.66	12.66	7.10	7.71	8.07	8.07
w/ Best-of-N	12.66	13.58	13.58	14.19	7.10	7.71	7.71	8.07
w/ Iterative Verification	12.66	14.19	16.00	16.67	8.07	14.72	15.24	16.14
<i>ReAct with Collaboration Modules</i>								
Budget-Unaware	15.33	16.00	16.00	16.00	10.36	16.14	17.46	17.46
Budget-Aware Prompt	11.33	13.33	14.00	14.00	16.98	16.74	16.74	16.74
AGENT*	16.00	16.67	20.00	21.33	18.07	18.92	19.42	20.72

Table 1: Evaluation Results of the baselines using Accuracy under budget (Acc@ B) on GAIA and BrowseComp-Plus for Claude and Qwen3 model families.

Baselines. We consider the following categories of baselines. (1) **Fixed Agent Workflow**: Baselines with optimized agent workflow that are fixed during the test time. We employ the most representative works, ADAS (Hu et al., 2024) and AFlow (Zhang et al., 2024a). (2) **ReAct with test-time scaling**: We evaluate on the standard ReAct framework (Yao et al., 2023) along with two well-established test-time scaling methods, Best-of-N (Brown et al., 2024) and Iterative Verification (Madaan et al., 2023). (3) **ReAct with Collaboration Modules**: We incorporate collaboration modules into the ReAct loop and evaluate two settings: **Budget-Unaware**, where the orchestrator does not receive explicit budget information, and **Budget-Aware Prompt**, where the orchestrator is given explicit budget information in the prompt⁴.

4.2 BENCHMARK RESULTS

Table 1 presents the experiment results of the baselines on the GAIA and BrowseComp-Plus benchmarks across different budget constraints.

First, fixed agent workflow baselines underperform on both GAIA and BrowseComp-Plus because they cannot adapt agent execution dynamically during inference. These benchmarks contain questions that require highly diverse workflows, causing the optimized workflow in these methods to collapse into a simple, generic pipeline, which makes it difficult for a single, static workflow to address these tasks effectively. Moreover, since these approaches have no mechanism for dynamically allocating compute, their performance remains largely unchanged even as the available budget increases.

Next, ReAct performs noticeably better than fixed agent workflow baselines but exhibits limited utilization of larger budgets. Although adding Best-of-N or Iterative Verification partially mitigates

⁴We provide detailed implementation details in Appx. §C

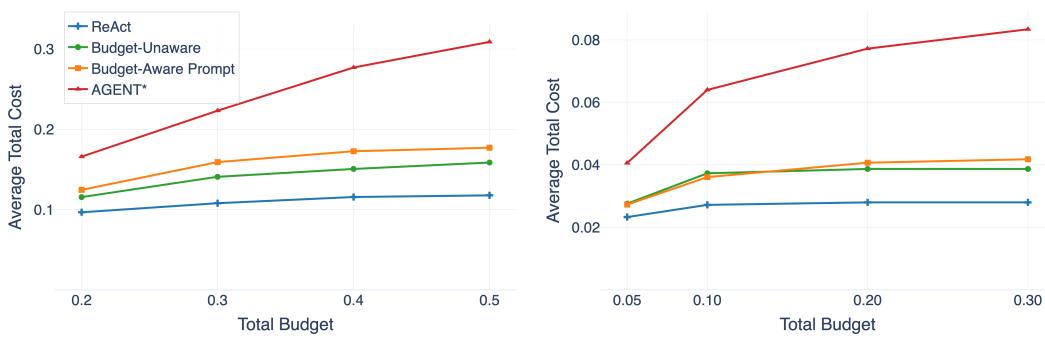


Figure 2: Average total cost vs total budget on GAIA using Claude (left) and Qwen3 (right).

this issue, these methods consume significantly more compute while yielding only minimal and inconsistent accuracy gains, which quickly plateau as the budget increases. This suggests that simply allocating more budget through standard test-time scaling does not guarantee improved performance.

Lastly, collaboration modules lead to clear and consistent improvements. The Budget-Unaware variant already surpasses ReAct by a significant margin, confirming its effectiveness in using test-time compute. Moreover, while Budget-Aware Prompt method occasionally improves accuracy relative to non-budget-aware settings, its effect is inconsistent and in some cases plateaus at higher budgets. AGENT* further achieves the strongest results across all budgets, outperforming both the Budget-Unaware and Budget-Aware Prompt baselines. These gains highlight the value of dual-level planning, which enables the orchestrator to allocate test-time compute more effectively by balancing short-term decisions with long-horizon feasibility.

4.3 COST UTILIZATION UNDER BUDGET CONSTRAINTS

We further examine the average total cost incurred when running the GAIA benchmark under different budget constraints, as reported in Figure 2. Across all three baselines, the available budget is consistently underutilized, with the total cost plateauing well below the specified constraint. In contrast, AGENT* achieves substantially higher cost utilization, which transfers into a performance enhancement. This finding highlights a fundamental challenge in multi-agent systems: even when sufficient budget is available, baseline strategies struggle to convert it into effective computation that drives performance gains. By leveraging dual-level planning, AGENT* allocates resources more aggressively when necessary, ensuring that budget headroom is effectively translated into higher performance.

4.4 IMPACT OF SHORT-TERM AND LONG-TERM PLANNING

To assess the contribution of each planning component in AGENT*, we evaluate them by isolating each component under \$0.2 and \$0.5 budgets using Claude models on GAIA benchmark. The results are shown in Table 2. Short-term planning alone provides a meaningful boost over vanilla ReAct, as it guides the orchestrator toward more informed local decisions and encourages the use of valuable collaboration modules. Long-term planning alone performs slightly better, particularly at higher budgets, since it enables the model to anticipate downstream computation and avoid prematurely committing to suboptimal trajectories. When combined, these complementary behaviors yield the strongest performance, underscoring the value of integrating both local and global planning signals for budget-optimal multi-agent coordination.

Method	Acc@0.2	Acc@0.5
Only Short-term	36.41	44.44
Only Long-term	37.65	46.91
Dual-level (Both)	38.89	48.15

Table 2: Evaluation results of short-term and long-term planning on GAIA under \$0.2 and \$0.5 budget.

432

5 RELATED WORKS

433

5.1 TEST-TIME SCALING FOR LLMs

434 Existing approaches to increasing test-time compute for LLMs can be broadly categorized into two
 435 paradigms (Muennighoff et al., 2025), including parallel scaling and sequential scaling. Parallel
 436 scaling methods, such as best-of-N sampling (Brown et al., 2024; Snell et al., 2024), generate mul-
 437 tiple candidate solutions in parallel and select the best one using voting, confidence heuristics, or
 438 reward models (Wang et al., 2022; Irvine et al., 2023). In contrast, sequential scaling encourages iter-
 439 ative refinement via methods such as chain-of-thought prompting (Wei et al., 2022), self-refinement
 440 (Madaan et al., 2023; Chen et al., 2023; Min et al., 2024; Lee et al., 2025), and verifier-guided re-
 441 vision (Gou et al., 2023; Zhang et al., 2024d). **While these methods offer promising directions for**
 442 **test-time scaling, they do not consider optimization under a strict compute budget.** Recent works
 443 have explored budget-constrained settings (Muennighoff et al., 2025; Han et al., 2025; Zheng et al.,
 444 2024; Qiu et al., 2025), but they either focus on single LLM setting or target system-level efficiency
 445 rather than scaling test-time computation to improve the performance. In contrast, our work focuses
 446 on budget-optimal compute allocation in a multi-agent setting, which is not addressed by these prior
 447 approaches

448

5.2 DESIGNING AND OPTIMIZING MULTI-AGENT SYSTEM

449 Multi-agent systems (MAS) have recently gained traction as a way to structure complex tasks
 450 through the collaboration of specialized LLM-based agents. Early systems such as CAMEL (Li
 451 et al., 2023), AutoGen (Wu et al., 2024), and MetaGPT (Hong et al., 2023) demonstrated the value
 452 of explicit role assignment and agent interaction, but relied heavily on manual configurations, in-
 453 cluding prompt engineering, agent profiling, and fixed communication protocols (Qian et al., 2024).
 454 These limitations hinder their adaptability across domains and tasks. In response, recent work has
 455 focused on automating various components of MAS design. Some methods treat agent functions
 456 as learnable policies (Zhang et al., 2024b;c) or synthesize trajectories for offline agent optimiza-
 457 tion (Qiao et al., 2024). Others expand the MAS search space to include prompts (Khattab et al.,
 458 2023), tools (Zhou et al., 2024), workflows (Li et al., 2024), and reasoning strategies (Shang et al.,
 459 2024). DyLAN (Liu et al., 2024) supports dynamic agent composition, while Archon (Saad-Falcon
 460 et al., 2024) treats MAS construction as a hyperparameter optimization problem. GPTSwarm (Zhuge
 461 et al., 2024) optimizes agent communication using policy gradients, and state-of-the-art systems like
 462 ADAS (Hu et al., 2024) and AFlow (Zhang et al., 2024a) perform full workflow optimization using
 463 search algorithms or LLM controllers.

464 **More recently, several methods generate workflows on a per-query basis. MaAS (Zhang et al.,**
 465 **2025a) learns distributions over architectures to trade off performance and cost; FlowReasoner (Gao**
 466 **et al., 2025), ScoreFlow (Wang et al., 2025), and Flow (Niu et al., 2025) synthesize or refine work-
 467 **flows using execution feedback, DPO training, or graph-based reasoning.** However, these systems
 468 aim to produce a single optimized configuration per task and do not support dynamic, inference-time
 469 planning under compute budgets, nor do they introduce reusable collaboration abstractions that en-
 470 able adaptive, budget-efficient multi-agent coordination without retraining or static system design.**

471

6 CONCLUSION

472 We presented AGENT*, a general framework for budget-constrained optimization of test-time com-
 473 pute in multi-agent systems. AGENT* leverages collaboration modules as reusable abstractions of
 474 multi-agent coordination and employs a dual-level planning architecture that balances short-horizon
 475 execution with long-horizon speculation. Extensive experiments demonstrate that AGENT* consis-
 476 tently outperforms both standard baselines and those augmented with test-time scaling, achieving
 477 higher accuracy while utilizing resources more efficiently. These findings underscore the impor-
 478 tance of structured collaboration and forward-looking planning for budget-constrained inference,
 479 and point toward a promising direction for building more adaptable and compute-efficient multi-
 480 agent systems.

486 REFERENCES
487

488 Anthropic. Claude code: Deep coding at terminal velocity. Anthropic Blog / Product Page, September
489 2024. URL: <https://www.anthropic.com/clause-code>.

490 Anthropic. Claude: Product overview. [https://www.anthropic.com/product/](https://www.anthropic.com/product/overview)
491 overview, 2025. Accessed: YYYY-MM-DD.

492 Vidhisha Balachandran, Jingya Chen, Lingjiao Chen, Shivam Garg, Neel Joshi, Yash Lara, John
493 Langford, Besmira Nushi, Vibhav Vineet, Yue Wu, et al. Inference-time scaling for complex
494 tasks: Where we stand and what lies ahead. *arXiv preprint arXiv:2504.00294*, 2025.

495 Bradley Brown, Jordan Juravsky, Ryan Ehrlich, Ronald Clark, Quoc V Le, Christopher Ré, and
496 Azalia Mirhoseini. Large language monkeys: Scaling inference compute with repeated sampling.
497 *arXiv preprint arXiv:2407.21787*, 2024.

498 Jiefeng Chen, Jie Ren, Xinyun Chen, Chengrun Yang, Ruoxi Sun, Jinsung Yoon, and Sercan Ö
499 Arik. Sets: Leveraging self-verification and self-correction for improved test-time scaling. *arXiv*
500 *preprint arXiv:2501.19306*, 2025a.

501 Xinyun Chen, Maxwell Lin, Nathanael Schärlí, and Denny Zhou. Teaching large language models
502 to self-debug. *arXiv preprint arXiv:2304.05128*, 2023.

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

506 Hongcheng Gao, Yue Liu, Yufei He, Longxu Dou, Chao Du, Zhijie Deng, Bryan Hooi, Min
507 Lin, and Tianyu Pang. Flowreasoner: Reinforcing query-level meta-agents. *arXiv preprint*
508 *arXiv:2504.15257*, 2025.

509 Google. Gemini deep research. Gemini Overview, February 2025. URL: <https://gemini.google/overview/deep-research/>.

510 Zhibin Gou, Zhihong Shao, Yeyun Gong, Yelong Shen, Yujiu Yang, Nan Duan, and Weizhu Chen.
511 Critic: Large language models can self-correct with tool-interactive critiquing. *arXiv preprint*
512 *arXiv:2305.11738*, 2023.

513 Jeremy Hadfield, Barry Zhang, Kenneth Lien, Florian Scholz, Jeremy Fox, and Daniel Ford. How we
514 built our multi-agent research system. <https://www.anthropic.com/engineering/multi-agent-research-system>, June 2025. Blog post, *Engineering at Anthropic*.

515 Tingxu Han, Zhenting Wang, Chunrong Fang, Shiyu Zhao, Shiqing Ma, and Zhenyu Chen. Token-
516 budget-aware llm reasoning. In *Findings of the Association for Computational Linguistics: ACL*
517 2025, pp. 24842–24855, 2025.

518 Peter E Hart, Nils J Nilsson, and Bertram Raphael. A formal basis for the heuristic determination
519 of minimum cost paths. *IEEE transactions on Systems Science and Cybernetics*, 4(2):100–107,
520 1968.

521 Sirui Hong, Mingchen Zhuge, Jonathan Chen, Xiawu Zheng, Yuheng Cheng, Jinlin Wang, Ceyao
522 Zhang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, et al. Metagpt: Meta programming for
523 a multi-agent collaborative framework. In *The Twelfth International Conference on Learning*
524 *Representations*, 2023.

525 Shengran Hu, Cong Lu, and Jeff Clune. Automated design of agentic systems. *arXiv preprint*
526 *arXiv:2408.08435*, 2024.

527 Wenlong Huang, Pieter Abbeel, Deepak Pathak, and Igor Mordatch. Language models as zero-shot
528 planners: Extracting actionable knowledge for embodied agents. In *International conference on*
529 *machine learning*, pp. 9118–9147. PMLR, 2022.

530 Robert Irvine, Douglas Boubert, Vyas Raina, Adian Liusie, Ziyi Zhu, Vineet Mudupalli, Aliaksei
531 Korshuk, Zongyi Liu, Fritz Cremer, Valentin Assassi, et al. Rewarding chatbots for real-world
532 engagement with millions of users. *arXiv preprint arXiv:2303.06135*, 2023.

540 Omar Khattab, Arnav Singhvi, Paridhi Maheshwari, Zhiyuan Zhang, Keshav Santhanam, Sri
 541 Vardhamanan, Saiful Haq, Ashutosh Sharma, Thomas T Joshi, Hanna Moazam, et al. Dspy:
 542 Compiling declarative language model calls into self-improving pipelines. *arXiv preprint*
 543 *arXiv:2310.03714*, 2023.

544 Kuang-Huei Lee, Ian Fischer, Yueh-Hua Wu, Dave Marwood, Shumeet Baluja, Dale Schuurmans,
 545 and Xinyun Chen. Evolving deeper llm thinking. *arXiv preprint arXiv:2501.09891*, 2025.

546 Yaniv Leviathan, Matan Kalman, and Yossi Matias. Fast inference from transformers via speculative
 547 decoding. In *International Conference on Machine Learning*, pp. 19274–19286. PMLR, 2023.

548 Guohao Li, Hasan Hammoud, Hani Itani, Dmitrii Khizbulin, and Bernard Ghanem. Camel: Com-
 549 municative agents for “mind” exploration of large language model society. *Advances in Neural*
 550 *Information Processing Systems*, 36:51991–52008, 2023.

551 Zelong Li, Shuyuan Xu, Kai Mei, Wenyue Hua, Balaji Rama, Om Raheja, Hao Wang, He Zhu, and
 552 Yongfeng Zhang. Autoflow: Automated workflow generation for large language model agents.
 553 *arXiv preprint arXiv:2407.12821*, 2024.

554 Zijun Liu, Yanzhe Zhang, Peng Li, Yang Liu, and Diyin Yang. A dynamic llm-powered agent network
 555 for task-oriented agent collaboration. In *First Conference on Language Modeling*, 2024.

556 Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri
 557 Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, et al. Self-refine: Iterative refinement
 558 with self-feedback. *Advances in Neural Information Processing Systems*, 36:46534–46594, 2023.

559 Silin Meng, Yiwei Wang, Cheng-Fu Yang, Nanyun Peng, and Kai-Wei Chang. Llm-a*: Large
 560 language model enhanced incremental heuristic search on path planning. In *Findings of the As-
 561 sociation for Computational Linguistics: EMNLP 2024*, pp. 1087–1102, 2024.

562 Grégoire Mialon, Clémentine Fourrier, Thomas Wolf, Yann LeCun, and Thomas Scialom. Gaia:
 563 a benchmark for general ai assistants. In *The Twelfth International Conference on Learning
 564 Representations*, 2023.

565 Yingqian Min, Zhipeng Chen, Jinhao Jiang, Jie Chen, Jia Deng, Yiwen Hu, Yiru Tang, Jiapeng
 566 Wang, Xiaoxue Cheng, Huatong Song, et al. Imitate, explore, and self-improve: A reproduction
 567 report on slow-thinking reasoning systems. *arXiv preprint arXiv:2412.09413*, 2024.

568 Niklas Muennighoff, Zitong Yang, Weijia Shi, Xiang Lisa Li, Li Fei-Fei, Hannaneh Hajishirzi, Luke
 569 Zettlemoyer, Percy Liang, Emmanuel Candès, and Tatsunori Hashimoto. s1: Simple test-time
 570 scaling. *arXiv preprint arXiv:2501.19393*, 2025.

571 Boye Niu, Yiliao Song, Kai Lian, Yifan Shen, Yu Yao, Kun Zhang, and Tongliang Liu. Flow:
 572 Modularized agentic workflow automation. *arXiv preprint arXiv:2501.07834*, 2025.

573 OpenAI. Codex. OpenAI Product Page, 2024. URL: <https://openai.com/codex/>.

574 OpenAI. Introducing deep research. OpenAI Blog, February 2 2025. URL: <https://openai.com/index/introducing-deep-research/>.

575 Chen Qian, Zihao Xie, Yifei Wang, Wei Liu, Kunlun Zhu, Hanchen Xia, Yufan Dang, Zhuoyun Du,
 576 Weize Chen, Cheng Yang, et al. Scaling large language model-based multi-agent collaboration.
 577 *arXiv preprint arXiv:2406.07155*, 2024.

578 Shuofei Qiao, Ningyu Zhang, Runnan Fang, Yujie Luo, Wangchunshu Zhou, Yuchen Jiang,
 579 Chengfei Lv, and Huajun Chen. AutoAct: Automatic agent learning from scratch for QA via
 580 self-planning. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *Proceedings of the*
 581 *62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Pa-
 582 pers)*, pp. 3003–3021, Bangkok, Thailand, August 2024. Association for Computational Linguis-
 583 tics. doi: 10.18653/v1/2024.acl-long.165. URL <https://aclanthology.org/2024.acl-long.165/>.

594 Rennai Qiu, Chen Qian, Ran Li, Yufan Dang, Weize Chen, Cheng Yang, Yingli Zhang, Ye Tian, Xu-
 595 antang Xiong, Lei Han, et al. Co-saving: Resource aware multi-agent collaboration for software
 596 development. *arXiv preprint arXiv:2505.21898*, 2025.

597

598 Jon Saad-Falcon, Adrian Gamarra Lafuente, Shlok Natarajan, Nahum Maru, Hristo Todorov, Etash
 599 Guha, E Kelly Buchanan, Mayee Chen, Neel Guha, Christopher Ré, et al. Archon: An architecture
 600 search framework for inference-time techniques. *arXiv preprint arXiv:2409.15254*, 2024.

601

602 Yu Shang, Yu Li, Keyu Zhao, Likai Ma, Jiahe Liu, Fengli Xu, and Yong Li. Agentsquare: Automatic
 603 llm agent search in modular design space. *arXiv preprint arXiv:2410.06153*, 2024.

604

605 Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. Scaling llm test-time compute optimally
 606 can be more effective than scaling model parameters. *arXiv preprint arXiv:2408.03314*, 2024.

607

608 Khanh-Tung Tran, Dung Dao, Minh-Duong Nguyen, Quoc-Viet Pham, Barry O’Sullivan, and
 609 Hoang D Nguyen. Multi-agent collaboration mechanisms: A survey of llms. *arXiv preprint
 610 arXiv:2501.06322*, 2025.

611

612 Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdh-
 613 ery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models.
 614 *arXiv preprint arXiv:2203.11171*, 2022.

615

616 Yinjie Wang, Ling Yang, Guohao Li, Mengdi Wang, and Bryon Aragam. Scoreflow: Mastering
 617 llm agent workflows via score-based preference optimization. *arXiv preprint arXiv:2502.04306*,
 618 2025.

619

620 Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny
 621 Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in
 622 neural information processing systems*, 35:24824–24837, 2022.

623

624 Jason Wei, Zhiqing Sun, Spencer Papay, Scott McKinney, Jeffrey Han, Isa Fulford, Hyung Won
 625 Chung, Alex Tachard Passos, William Fedus, and Amelia Glaese. Browsecmp: A simple yet
 626 challenging benchmark for browsing agents. *arXiv preprint arXiv:2504.12516*, 2025.

627

628 Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun
 629 Zhang, Shaokun Zhang, Jiale Liu, et al. Autogen: Enabling next-gen llm applications via multi-
 630 agent conversations. In *First Conference on Language Modeling*, 2024.

631

632 An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang
 633 Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu,
 634 Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin
 635 Yang, Jiaxi Yang, Jing Zhou, Jingren Zhou, Junyang Lin, Kai Dang, Keqin Bao, Kexin Yang,
 636 Le Yu, Lianghao Deng, Mei Li, Mingfeng Xue, Mingze Li, Pei Zhang, Peng Wang, Qin Zhu, Rui
 637 Men, Ruize Gao, Shixuan Liu, Shuang Luo, Tianhao Li, Tianyi Tang, Wenbiao Yin, Xingzhang
 638 Ren, Xinyu Wang, Xinyu Zhang, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yinger
 639 Zhang, Yu Wan, Yuqiong Liu, Zekun Wang, Zeyu Cui, Zhenru Zhang, Zhipeng Zhou, and Zihan
 640 Qiu. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

641

642 Shunyu Yao, Rohan Rao, Matthew Hausknecht, and Karthik Narasimhan. Keep calm and explore:
 643 Language models for action generation in text-based games. *arXiv preprint arXiv:2010.02903*,
 644 2020.

645

646 Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao.
 647 React: Synergizing reasoning and acting in language models. In *International Conference on
 648 Learning Representations (ICLR)*, 2023.

649

650 Guibin Zhang, Luyang Niu, Junfeng Fang, Kun Wang, Lei Bai, and Xiang Wang. Multi-agent
 651 architecture search via agentic supernet. *arXiv preprint arXiv:2502.04180*, 2025a.

652

653 Jiayi Zhang, Jinyu Xiang, Zhaoyang Yu, Fengwei Teng, Xionghui Chen, Jiaqi Chen, Mingchen
 654 Zhuge, Xin Cheng, Sirui Hong, Jinlin Wang, et al. Aflow: Automating agentic workflow genera-
 655 tion. *arXiv preprint arXiv:2410.10762*, 2024a.

648 Qiyuan Zhang, Fuyuan Lyu, Zexu Sun, Lei Wang, Weixu Zhang, Wenyue Hua, Haolun Wu, Zhihan
 649 Guo, Yufei Wang, Niklas Muennighoff, et al. A survey on test-time scaling in large language
 650 models: What, how, where, and how well? *arXiv preprint arXiv:2503.24235*, 2025b.

651 Shaokun Zhang, Jieyu Zhang, Jiale Liu, Linxin Song, Chi Wang, Ranjay Krishna, and Qingyun
 652 Wu. Offline training of language model agents with functions as learnable weights. In *Forty-first*
 653 *International Conference on Machine Learning*, 2024b.

654 Wenqi Zhang, Ke Tang, Hai Wu, Mengna Wang, Yongliang Shen, Guiyang Hou, Zeqi Tan, Peng Li,
 655 Yuetong Zhuang, and Weiming Lu. Agent-pro: Learning to evolve via policy-level reflection and
 656 optimization. *arXiv preprint arXiv:2402.17574*, 2024c.

657 Yunxiang Zhang, Muhammad Khalifa, Lajanugen Logeswaran, Jaekyeom Kim, Moontae Lee,
 658 Honglak Lee, and Lu Wang. Small language models need strong verifiers to self-correct rea-
 659 soning. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *Findings of the Association*
 660 *for Computational Linguistics: ACL 2024*, pp. 15637–15653, Bangkok, Thailand, August
 661 2024d. Association for Computational Linguistics. doi: 10.18653/v1/2024.findings-acl.924. URL
 662 <https://aclanthology.org/2024.findings-acl.924/>.

663 Yuanhang Zheng, Peng Li, Ming Yan, Ji Zhang, Fei Huang, and Yang Liu. Budget-constrained tool
 664 learning with planning. *arXiv preprint arXiv:2402.15960*, 2024.

665 Wangchunshu Zhou, Yixin Ou, Shengwei Ding, Long Li, Jialong Wu, Tiannan Wang, Jiamin Chen,
 666 Shuai Wang, Xiaohua Xu, Ningyu Zhang, et al. Symbolic learning enables self-evolving agents.
 667 *arXiv preprint arXiv:2406.18532*, 2024.

668 Mingchen Zhuge, Wenyi Wang, Louis Kirsch, Francesco Faccio, Dmitrii Khizbulin, and Jürgen
 669 Schmidhuber. Gptswarm: Language agents as optimizable graphs. In *Forty-first International*
 670 *Conference on Machine Learning*, 2024.

671 A SUPPLEMENTARY DETAILS ON AGENTS AND COLLABORATION MODULES

672 For the GAIA benchmark, we employ the following agents:

- 673 • *Search Agent*: Given a search query, outputs the google search results
- 674 • *Browser Agent*: Given one or more URLs, visits the pages and returns the content of webpages.
- 675 • *Reasoning Agent*: Given a problem, performs multi-step reasoning and outputs a final solution.
- 676 • *Media Inspector Agent*: Given an image or video link and a question, analyzes the media and
 677 answers the question.

678 In addition, we define the following collaboration modules:

- 679 • *interactive_search_and_browse*: A search agent and a browser agent share context and
 680 invoke each other interactively.
- 681 • *search_then_browse*: First searches for URLs, then browses the retrieved pages for detailed
 682 content.
- 683 • *ensemble_search*: Generates three distinct queries, spawns three search agents in parallel,
 684 and aggregates their results.
- 685 • *two_ensemble_reasoning*: Two reasoning agents independently produce reasoning paths,
 686 which are aggregated into a final answer.
- 687 • *three_ensemble_reasoning*: Three reasoning agents independently produce reasoning
 688 paths, which are aggregated into a final answer.

689 For the BrowseComp-Plus benchmark, we employ the following agents:

- 690 • *Retriever Agent*: Given a search query, retrieves the top-5 documents by semantic similarity and
 691 returns (*doc_id*, *title*, *snippet*) for each.

702 • *Document Reader Agent*: Given a `doc_id`, fetches and returns the full document content.
 703 • *Critic Agent*: Given the current task state and available information, identifies missing information
 704 and recommends what to search for next.
 705

706 We define the following collaboration modules:
 707

708 • `interactive_search`: A search agent and a document reader agent share context and invoke
 709 each other interactively.
 710 • `ensemble_interactive_search`: Spawns three `interactive_search` modules in par-
 711 allel and aggregates their results.
 712 • `interactive_search_then_critic`: First calls the `interactive_search` module,
 713 then the critic agent evaluates the gathered information.
 714 • `ensemble_interactive_search_then_critic`: Spawns three
 715 `interactive_search_then_critic` modules in parallel and aggregates their results.
 716

717 B TOKEN PRICING FOR COST COMPUTATION

720 We compute the monetary cost of each action by multiplying its input and output token usage with
 721 the official token pricing provided by AWS Bedrock. Table 3 lists the exact token prices used
 722 for all models in our experiments. These values correspond to the pricing available at the time of
 723 experimentation.

724 Model Name	725 Input Price (\$/1K tok)	725 Output Price (\$/1K tok)
726 <code>claude-3-5-haiku-latest</code>	\$0.0008	\$0.004
727 <code>claude-3-7-sonnet-latest</code>	\$0.003	\$0.015
728 <code>qwen3-32b</code>	\$0.0007	\$0.0028

729 Table 3: Token prices used for calculating costs and budgets.
 730

732 C IMPLEMENTATION DETAILS

735 Here, we provide detailed implementation details for the baselines. For the best-of-N baseline, we
 736 set $N = 3$, but if the budget is exhausted before completing all three attempts, we use the available
 737 attempts and apply self-consistency on the answers from the attempts to produce the final answer.
 738 For the iterative verification baseline, once an initial answer is generated, any remaining budget is
 739 used to prompt the orchestrator to re-examine the trajectory and refine the solution until the budget
 740 is fully consumed.

741 D MAIN EXPERIMENT SCORES

744 We have additionally included the specific performance scores presented in §4.2 in Table 4 and
 745 Table 5.

747 E DUAL-LEVEL PLANNING ALGORITHM

749 We provide algorithm for dual-level planning in Alg. 1.
 750

751
 752
 753
 754
 755

756 You are given a set of successful execution trajectories produced by
 757 a multi-agent system.
 758 Each trajectory contains the sequence of agents invoked with a
 759 subtask, and their intermediate outputs
 760
 761 Your goal is to analyze these trajectories to identify a recurring
 762 agent interactions that frequently appear in successful
 763 trajectories.
 764 Such patterns may include:
 765 - Sequential workflows
 766 - Parallel workflows
 767 - Verification or Refinement
 768
 769 For each recurring pattern you identify:
 770 1. Describe the workflow in plain language.
 771 2. Specify the agents involved and their roles.
 772 3. Implement a class that executes this workflow using these agents
 773 4. Explain why this pattern is effective, referring to the trajectory
 774 evidence.
 775
 776 Here are some examples of the extracted pattern:
 777 {few_shot_demonstrations}
 778
 779 Here are the patterns that are already observed. Avoid outputting
 780 duplicated patterns.
 781 {collected_collaboration_modules}

779
 780 Figure 3: Prompt used for self-play reflection to induce collaboration modules from successful
 781 trajectories.

Method	GAIA		BrowseComp-Plus	
	Budget (\$)	Accuracy (%)	Budget (\$)	Accuracy (%)
No Module, No Budget Aware		35.80		24.33
With Module, No Budget Aware	0.2	35.80	0.2	25.32
With Module, Budget-Aware		36.41		24.07
AGENT*		38.89		25.53
No Module, No Budget Aware		35.80		24.66
With Module, No Budget Aware	0.3	37.65	0.3	25.50
With Module, Budget-Aware		41.97		26.34
AGENT*		44.44		27.50
No Module, No Budget Aware		35.80		24.66
With Module, No Budget Aware	0.4	38.27	0.4	27.08
With Module, Budget-Aware		43.20		27.37
AGENT*		46.91		28.07
No Module, No Budget Aware		35.80		24.66
With Module, No Budget Aware	0.5	38.27	0.5	26.2
With Module, Budget-Aware		43.20		28.07
AGENT*		48.15		29.00

802 Table 4: Performance of different methods across different budget on GAIA and BrowseComp-Plus
 803 using Claude model family.

804
 805
 806
 807
 808
 809

810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825

Method	GAIA		BrowseComp-Plus	
	Budget (\$)	Accuracy (%)	Budget (\$)	Accuracy (%)
No Module, No Budget Aware		12.66		7.10
With Module, No Budget Aware	0.05	15.33	0.025	10.36
With Module, Budget-Aware		11.33		16.98
AGENT*		16.00		18.07
No Module, No Budget Aware		12.66		7.71
With Module, No Budget Aware	0.1	16.0	0.05	16.14
With Module, Budget-Aware		13.33		16.74
AGENT*		16.67		18.92
No Module, No Budget Aware		12.66		8.07
With Module, No Budget Aware	0.2	16.0	0.1	17.46
With Module, Budget-Aware		14.00		16.74
AGENT*		20.00		19.42
No Module, No Budget Aware		12.66		8.07
With Module, No Budget Aware	0.3	16.0	0.2	17.46
With Module, Budget-Aware		14.00		16.74
AGENT*		21.33		20.72

846 Table 5: Performance of different methods across different budget on GAIA and BrowseComp-Plus
847 using Qwen3 model family.

848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863

864

865

866

867

868

869

870

871

872

873

Algorithm 1: Dual-Level Planning

Input: Policy π , set of agents/modules \mathcal{A}' , initial history \mathcal{H}_0 , total budget B , max steps T_{\max} **Output:** Final output and execution trace

```

1  $t \leftarrow 0;$ 
2  $\mathcal{H}_t \leftarrow \mathcal{H}_0;$ 
3  $B_t \leftarrow B;$ 
4  $\mathcal{T}_{\text{feasible},0} \leftarrow \emptyset;$ 
5 while  $t < T_{\max}$  and  $B_t > 0$  and not SOLVED( $\mathcal{H}_t$ ) do
6   Sample candidate actions
872
873    $\mathcal{C}_t = \{\alpha_t^{(1)}, \dots, \alpha_t^{(K)}\} \sim \pi(\cdot \mid \mathcal{H}_t, \mathcal{T}_{\text{feasible},t-1})$ 
874   // Let  $\phi((a, v)) = a$  extract the agent/module from an action
875   for  $i = 1$  to  $K$  do
876
877
878    $g(\alpha_t^{(i)}) = \frac{1}{K} \sum_{k=1}^K \mathbf{1}[\phi(\alpha_t^{(k)}) = \phi(\alpha_t^{(i)})]$ 
879
880
881   for  $i = 1$  to  $K$  do
882     Generate speculative trajectories  $\mathcal{T}_t(\alpha_t^{(i)})$  beginning with  $\alpha_t^{(i)}$ ;
883     Define feasible set:
884
885      $\mathcal{T}_{\text{feasible},t}(\alpha_t^{(i)}) = \{\tau \in \mathcal{T}_t(\alpha_t^{(i)}) \mid \text{cost}(\tau) \leq B_t\}$ 
886
887      $h(\alpha_t^{(i)}) = \frac{|\mathcal{T}_{\text{feasible},t}(\alpha_t^{(i)})|}{\sum_{r=1}^K |\mathcal{T}_{\text{feasible},t}(\alpha_t^{(r)})|}$ 
888
889
890   Compute  $f(\alpha_t^{(i)}) = g(\alpha_t^{(i)}) + h(\alpha_t^{(i)})$  for  $i = 1, \dots, K$ ;
891
892    $\alpha_t^* = \arg \max_{\alpha \in \mathcal{C}_t} f(\alpha)$ 
893
894   Execute  $\alpha_t^* = (a_t^*, v_t^*)$  to obtain output  $o_t$ ;
895   Update state:  $\mathcal{H}_{t+1} \leftarrow \mathcal{H}_t \cup \{(v_t^*, o_t)\}$ ;
896   Update budget:  $B_{t+1} \leftarrow B_t - \text{cost}(\alpha_t^*)$ ;
897    $t \leftarrow t + 1$ ;
898
899   return Final output derived from  $\mathcal{H}_t$ 

```

909

910

911

912

913

914

915

916

917

Method	Budget (\$)	Total Cost (\$)	Accuracy (%)
With Modules			
+ No Budget Aware		0.1156	35.80
+ Budget-Aware Prompting	0.2	0.1246	36.41
+ Best-of- N Sampling		0.1698	35.80
+ Iterative Verification		0.1928	36.41
AGENT*		0.1660	38.89
With Modules			
+ No Budget Aware		0.1409	37.65
+ Budget-Aware Prompting	0.3	0.1592	41.97
+ Best-of- N Sampling		0.2583	37.65
+ Iterative Verification		0.2903	42.59
AGENT*		0.2233	44.44
With Modules			
+ No Budget Aware		0.1507	38.27
+ Budget-Aware Prompting	0.4	0.1727	43.20
+ Best-of- N Sampling		0.3464	43.20
+ Iterative Verification		0.3885	43.20
AGENT*		0.2771	46.91
With Modules			
+ No Budget Aware		0.1587	38.27
+ Budget-Aware Prompting	0.5	0.1771	43.20
+ Best-of- N Sampling		0.3825	44.44
+ Iterative Verification		0.4803	43.20
AGENT*		0.3090	48.15

Table 6: Performance comparison among different test-time scaling methods under budget constraints on GAIA with Claude.