EVOLVING MULTI-AGENT SYSTEMS THROUGH SENSE OF AGENCY

Anonymous authors
Paper under double-blind review

ABSTRACT

Multi-agent systems (MAS) tackle complex tasks by coordinating agents with complementary roles and capabilities. Evolving their individual behaviors and interaction structures during inference enables adaptation to task-specific demands, but requires informative feedback. Existing approaches often rely on *external feedback* from task-specific verification environments or separate LLM evaluators, limiting evolution when such feedback is unavailable. We propose ***SOAP***, ***the Sense of Agency Paradigm***, which draws inspiration from people's *Sense of Agency (SoA)*—the feeling that their own actions shape what happens. SOAP turns this intuition into internal feedback: agents anticipate what their actions will achieve and assess whether execution meets those expectations. We formulate *computational SoA* to capture both an agent's expectation of performing well and how closely its post-execution self-assessment matches that expectation. At *individual* (i.e., *intra-agent)* and *interaction* (i.e., *inter-agent)* levels, this feedback guides iterative prompt and topology updates toward higher anticipated achievement and smaller prediction–outcome gaps, without external feedback. Experiments on BrowseComp-Plus, Plancraft, WorkBench, and Cloudcast demonstrate superior performance against evaluated baselines, with gains across backbone models. Further analysis highlights the complementary benefits of individual- and interaction-level feedback, supporting SoA as an effective internal signal for improving task completion during MAS evolution. Supplementary materials are available here, and a demo video can be downloaded here.

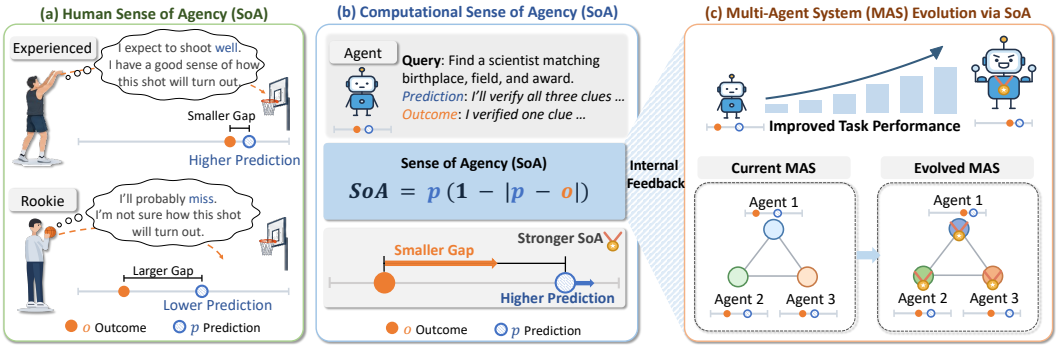

Figure 1: **Sense of Agency (SoA) as internal feedback for Multi-Agent System (MAS) evolution.** We introduce SOAP, a framework that uses agents' own assessments to guide the evolution of individual behaviors and collaboration. In cognitive science, *Sense of Agency (SoA)* is the subjective experience of controlling one's own actions and their consequences in the outside world. Inspired by this concept, we formulate computational SoA through two components: high anticipated performance and close agreement between prediction and actual outcome. **(a) Human intuition:** An experienced basketball player expects to perform well and can more accurately anticipate the outcome, whereas a novice has lower expectations and larger uncertainty. **(b) Computational formulation:** We define $SoA = p(1 - |p - o|)$ combines a prediction score p with its agreement with the post-execution assessment o. **(c) MAS evolution:** SoA-based internal feedback guides successive rounds of adaptation towards higher predictions and smaller prediction–outcome gaps, enabling improved task performance.

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

1 INTRODUCTION

Large language model (LLM)-based Multi-Agent Systems (MAS) address complex tasks by coordinating agents with different roles and tools, with various applications in information retrieval, workplace automation, and software development (Anthropic, 2025; Wu et al., 2023; Hong et al., 2023). Their effectiveness depends on both how individual agents act and how they exchange information and coordinate their work. However, agent prompts and collaboration structures configured before execution may not adequately accommodate the demands of a particular task. An agent may adopt an ineffective strategy, while a collaboration pattern may fail to deliver the information needed to make progress. These limitations motivate *MAS evolution*: adapting agent behaviors and collaboration structures using experience gathered during task execution (Dang et al., 2025; Xu et al., 2026). Such adaptation can take place at inference time, allowing the system to address task-specific difficulties as they emerge. A central question is therefore what feedback should guide this evolution.

Existing approaches to agent and MAS evolution use *external feedback* from sources such as execution-based verification and external LLM judges. For tasks with executable checks, a verification environment can indicate whether a proposed solution successfully solves the task (Hong et al., 2023; Qu et al., 2026). For tasks without straightforward verification, an external LLM evaluator can assess an execution or its outputs against specified criteria (Chen et al., 2023; Yuan et al., 2025). Although useful, these approaches make evolution dependent on the availability and configuration of external evaluation mechanisms. Transferring them to new task settings may require constructing or adapting verification environments, evaluation criteria, or judge configurations. This dependence limits the portability of the evolution process: before a system can improve, an appropriate external source of feedback must be available.

To obtain generally applicable feedback for MAS evolution, we draw inspiration from how people assess their own performance. Consider an experienced basketball player and a first-time player attempting the same shot (Figure 1(a)). The experienced player is confident in their ability to shoot accurately and, upon releasing the ball, can usually tell whether it will go in; their expectation closely matches the actual result. The first-time player, by contrast, has little confidence in making the shot and may remain unsure whether the ball is on target even after releasing it. This example highlights both confidence in achieving an intended outcome and an accurate understanding of how one's actions affect that outcome. It motivates a connection to *sense of agency* (SoA) in cognitive science. SoA refers to the subjective experience of controlling one's own actions and, through them, influencing what happens in the external world (Haggard, 2017; Chambon et al., 2014; Moore & Fletcher, 2012).

Inspired by this concept, we introduce computational SoA as an internal objective for evolving individual agents within a MAS. We operationalize it through two components: high anticipated performance and close agreement between predicted and retrospectively assessed performance. Specifically, an agent's anticipated performance is weighted by this agreement to obtain its computational SoA. Agents generate these assessments from their own execution experience, providing *internal feedback* that guides revisions to their behaviors and collaborative interactions. Evolution thus targets both higher performance expectations and better execution that supports those expectations.

To this end, We propose SOAP (**S**ense **o**f **A**gency **P**aradigm), a framework for Multi-Agent Systems (MAS) evolution guided by Sense of Agency (SoA) based internal feedback. Specifically, SOAP automatically generates individual- and interaction-level rubrics without manual design. Before execution, each agent predicts how well it expects to satisfy these criteria; after execution, it assesses its performance using the same rubrics. At each evolution round, the framework uses these paired assessments to prompt revisions that target higher anticipated performance and smaller prediction-outcome discrepancies, jointly seeking to increase the agent's computational SoA. This internal feedback guides updates to agent prompts and collaboration topology, supporting the evolution of both individual behaviors and inter-agent interactions. The two objectives work together: agents are encouraged to expect better performance and revise their behavior so that execution more closely matches those higher expectations. Here, *internal* refers to feedback formed by the agents themselves from execution observations, rather than evaluation signals supplied by external evaluators or the environment. The framework thus turns self-assessments made before and after execution into a repeated process of MAS adaptation.

SOAP outperforms the evaluated baselines across diverse task scenarios, including information retrieval on BrowseComp-Plus (Chen et al., 2025), planning on Plancraft (Dagan et al., 2024), work-

place tool use on WorkBench (Styles et al., 2024), and system optimization on Cloudcast (Qi et al., 2026). We find that leveraging SoA to guide both individual execution and collaborative interactions contributes to these improvements, Consistent gains across different backbone models further support the applicability of the approach beyond a particular LLM. Empirical evidence on consistent performance improvements over different agent counts and evolution iterations demonstrates the robsustness of our proposed method. Finally, analyses across evolution rounds show that higher prospective assessments and smaller prediction–outcome discrepancies accompany improvements in task performance. These findings suggest that internally guided evolution can improve both what individual agents do and how they work together, with the agents' own Sense of Agency providing useful guidance for that process.

We summarize our contributions as follows:

- To the best of our knowledge, we are the first to enable inference-time MAS evolution driven entirely by agents' own assessments, using SoA-based internal feedback without any external feedback such as a separate evaluator or externally supplied task rewards guiding evolution.
- We formulate computational SoA for individual agents and introduce SOAP, which uses automatically generated rubrics and prompt-guided evolution to jointly target higher anticipated performance and smaller prediction-outcome discrepancies, adapting both agent prompts and collaboration topology.
- We demonstrate improvements over evaluated baselines across four benchmarks and multiple backbone models, with ablations and evolution analyses (*i.e.,* dynamics of SoA, prediction, prediction-outcome gap, and task performance), revealing the complementary effectiveness and robustness of leveraging SoA as internal feedback for MAS evolution.

2 Related Work

LLM-based Multi-Agent Systems. LLM-based Multi-Agent Systems (MAS) coordinate specialized agents through role assignment and structured communication. Early frameworks organize collaboration using predefined workflows (Hong et al., 2023), or programmable conversations, as in AutoGen (Wu et al., 2023). These designs support task decomposition and complementary expertise, particularly when subtasks permit parallel exploration (Kim et al., 2025). Industrial systems, including Anthropic's research agents (Anthropic, 2025) and Kimi's agent swarm (Kimi Team, 2026), further demonstrate the practical value of coordinated execution. However, collaboration introduces coordination costs, and predefined roles and communication structures may not accommodate task-specific requirements or emerging execution difficulties, motivating adaptive MAS designs. MAS^2 (Wang et al., 2026a) addresses runtime adaptation through a generator–implementer–rectifier team that constructs and revises task-specific multi-agent systems.

Evolution of Multi-Agent Systems. MAS evolution adapts agent behaviors and collaboration structures using execution experience. GPTSwarm (Zhuge et al., 2024) represents agents as optimizable graphs, refining node prompts and inter-agent connectivity. Feedback sources vary across approaches. *Execution-based verification* guides MetaGPT (Hong et al., 2023) and CORAL (Qu et al., 2026); EvoMAC (Hu et al., 2025) uses software-execution feedback and textual backpropagation to update agents and their connections at test time. *External LLM judges* support open-ended tasks such as deep research (Câmara et al., 2026), with evaluations targeting agent prompts (Yuan et al., 2025), workflows (Zhang et al., 2024), or topology (Xu et al., 2026). These mechanisms depend on verification procedures or evaluator configurations. Other approaches exploit interaction-derived signals: SelfOrg (Tastan et al., 2025) estimates peer contributions from agent responses, while MetaGen (Wang et al., 2026b) combines self-consistency with runtime and execution checks. CoMAS (Xue et al., 2026) uses an LLM judge to derive rewards from agent discussions and updates agent policies through reinforcement learning. SOAP instead uses agents' own rubric-based predictions and outcome assessments to guide within-query prompt and topology adaptation without updating model parameters.

Sense of Agency. Sense of agency (SoA) denotes the subjective experience of controlling actions and their consequences (Haggard, 2017). The correspondence between predicted and observed outcomes provides an important cue for assessing SoA (Chambon et al., 2014; Moore & Fletcher,

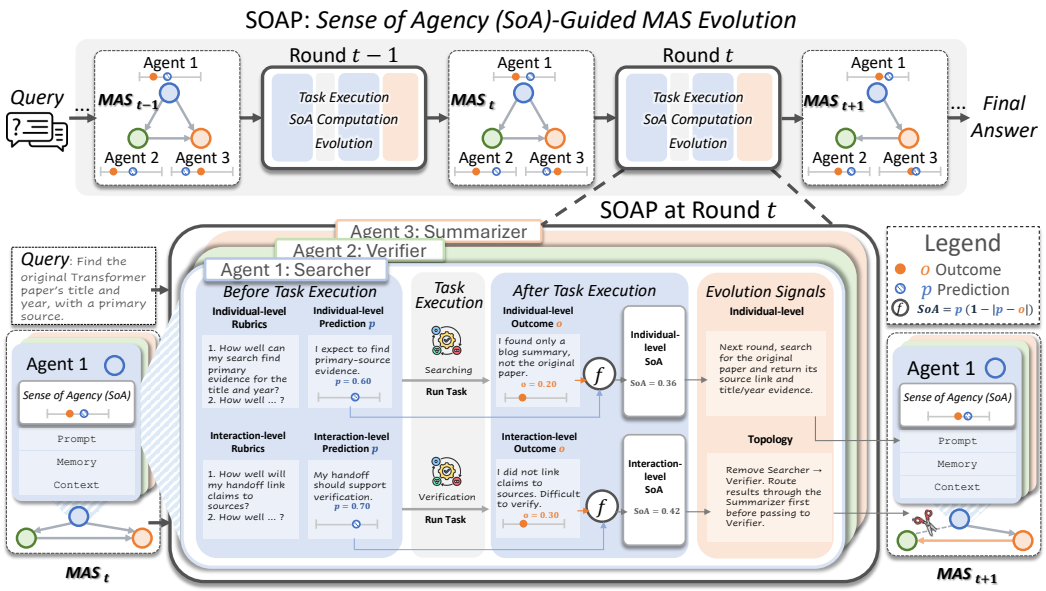

Figure 2: **Sense of Agency Paradigm (SOAP). (1) Upper: Overview.** Agents execute tasks and use SoA as internal feedback signals to evolve the MAS over multiple rounds before producing the final answer. **(2) Lower: Each Round.** Within each round, agents predict performance and assess outcomes under the same rubrics. For each agent, *Individual-level SoA* concerns its own execution, while *Interaction-level SoA* concerns the effects of its actions on downstream execution. These signals guide prompt and topology updates toward higher predictions and smaller prediction–outcome gaps, increasing computational SoA to support more successful task completion. (The example and scores are illustrative; see the Appendix for complete results from executions.)

2012). Cognitive science research also links SoA to learning and behavioral adaptation: agency judgments influence subsequent action selection (Karsh & Eitam, 2015), while stronger dispositional agency is associated with better action–effect learning (Puech et al., 2026). In LLM-related research, studies examine human control in co-creation (Zhang et al., 2025b), perceived human agency in conversational AI use (Beacock et al., 2026), and artificial agency and ownership in the GPT-4-driven humanoid Alter3 (Yoshida et al., 2024). SOAP, in contrast, operationalizes SoA as internal feedback for evolving agent configurations and collaboration topology at test time.

3 Method

We introduce SOAP, a paradigm for test-time MAS evolution guided by sense of agency (SoA) as an internal feedback signal. The evolution procedure includes *SoA computation* and *SoA-guided evolution*. We first introduce the task formulation and notation in Section 3.1, then present the SOAP framework in Section 3.2 and detail its two modules (*i.e.,* SoA computation and SoA-guided evolution) in Sections 3.3 and 3.4, respectively.

3.1 Notation and Task Formulation

Agents with Sense of Agency (SoA). In an MAS consists of agents V connected by a directed collaboration graph $G = (V, E)$, each agent i maintains a configuration θ_i that encompasses its task prompt, execution context, and memory, supporting local task execution, communication with other agents, and interaction with the environment. We additionally equip each agent with a sense-of-agency (SoA) module, which assesses the extent to which it can control the consequences of its actions in accordance with its intentions. We distinguish two types of computational SoA in MAS, including individual-level and interaction-level SoA.

Formally, at the ***individual level***, we define SoA as:

$$A_i = p_i(1 - |p_i - o_i|), \tag{1}$$

where $p \in [0, 1]$ denotes *predicted achievement* before execution and $o \in [0, 1]$ denotes *assessed achievement* after execution under the same rubric (*i.e.,* self-generated rubrics by each agent as detailed in Section 3.3.1).

At the ***interaction level***, we define SoA as:

$$A_{i \to j} = p_{i \to j}(1 - |p_{i \to j} - o_j|), \qquad (i, j) \in E, \tag{2}$$

where $p_{i \to j}$ is the sender agent i's prediction on what a directly connected downstream agent j will achieve upon receiving its output, and o_j is the receiver's assessed outcome.

Multi-Agent System Evolution. MAS evolution aims to improve task execution by adapting agent configurations and collaboration topology based on feedback. We consider test-time evolution over successive execution rounds on the same query. At round t, $\Theta^t = \{\theta_i^t\}_{i \in V^t}$ collects agent configurations, and $G^t = (V^t, E^t)$ specifies their collaboration topology. Evolution proceeds as

$$(\Theta^t, G^t) \xrightarrow{\text{execution on query}} \text{feedback}^t \xrightarrow{\text{evolution}} (\Theta^{t+1}, G^{t+1}). \tag{3}$$

where updating Θ^t improves agents' local task execution, while updating G^t adjusts interaction edges between agents. In this work, we investigate SoA as internal feedback to guide these updates, enabling individual and interaction evolution through agents' own predictions and observed outcomes.

3.2 Overview

Overall Pipeline. As illustrated in Figure 2, SOAP iteratively executes and evolves the MAS over the same query at test time. At each round, before execution, each agent predicts its own outcome and the outcomes of its directly connected downstream agents. The agents then execute their tasks and assess the observed outcomes under the same rubrics used for prediction. Then, each agent compares these predictions and assessments, obtaining individual- and interaction-level SoA (Section 3.3) as internal feedback. The SoA then guides agent refinement, with topology updates adapting the collaboration structure (Section 3.4). The updated system proceeds to the next round until the final answer is generated and the evolution is terminated. Evolution terminates when the mean individual-level SoA across agents remains below ϵ for k consecutive measured rounds. This criterion relies solely on internal feedback, without requiring a gold answer or an external evaluator.

3.3 Sense-of-Agency Computation

Overview. SoA computation consists of prediction, task execution, and SoA calculation. Each agent first predicts its own and its downstream collaborators' outcomes using self-generated, task- and role-conditioned rubrics. The agents then execute their tasks and assess the observed outcomes under the same rubrics. Finally, based on the paired predictions and outcome assessments, we can obtain individual- and interaction-level SoA, providing internal feedback for subsequent evolution.

3.3.1 Rubrics for SoA Assessment

Rubric Generation. An agent's SoA is assessed along dimensions that characterize how well it performs its task and supports downstream collaborators (*e.g.,* a search agent's candidate relevance, evidence sufficiency, and source traceability). For each test-time query, agents generate task- and role-conditioned rubrics, denoted by $\mathcal{R}_\rho$ for role ρ. Each dimension evaluates an aspect of task fulfillment using scoring criteria normalized to $[0, 1]$. The generated rubrics remains fixed across the evolution rounds to ensure consistent assessment. Individual-level SoA uses the agent's own role rubric, while interaction-level SoA uses the downstream receiver's agent rubric to assess how well the sender expects the receiver to perform its task upon receiving its output. Detailed prompts and examples for rubric generation are presented in Appendix A.2.

3.3.2 Individual and Interaction SoA

Prediction Before Task Execution. Before task execution at round t, agent i predicts how well it will perform its task under its own rubric, denoted by p_i^t, *i.e., individual-level prediction*. It also predicts how well each directly connected downstream agent j will perform upon receiving its output (*i.e., interaction-level prediction*), denoted by $p_{i \to j}^t$ and assessed under the receiver's rubric. Both

predictions assign a score in $[0, 1]$ to each corresponding rubric dimension (Section 3.3.1). The prediction prompt and concrete examples are provided in Appendix A.2.

Execution Outcome and SoA Score. After task execution, agent i assesses its execution outcome y_i^t under the same rubric used for prediction, yielding scores $o_{i,d}^t \in [0, 1]$ that reflect task fulfillment on each dimension d. We compute individual-level SoA by averaging the prediction–outcome scores across rubric dimensions:

$$A_i^t = \frac{1}{|\mathcal{R}_{(i)}|} \sum_{d \in \mathcal{R}_{(i)}} p_{i,d}^t (1 - |p_{i,d}^t - o_{i,d}^t|). \tag{4}$$

Interaction-level SoA compares the sender's prediction with the receiver's self-assessed outcome under the receiver's rubric:

$$A_{i \to j}^t = \frac{1}{|\mathcal{R}_{(j)}|} \sum_{d \in \mathcal{R}_{(j)}} p_{i \to j,d}^t (1 - |p_{i \to j,d}^t - o_{j,d}^t|), \qquad (i, j) \in E^t. \tag{5}$$

The receiver's assessment is reused for its own individual-level SoA and its incoming interaction-level scores. Both scores provide internal feedback for subsequent evolution. The prompt used for outcome assessment and SoA score is provided in Appendix A.2.

3.4 Sense-of-Agency-Guided Evolution

Overview. Based on the individual- and interaction-level SoA, SOAP then uses it as internal feedback to guide the MAS evolution defined in Section 3.1. As shown in Figure 2, individual evolution refines agent configurations, such as task prompts, to improve individual execution. On the other hand, interaction evolution adjusts the collaboration graph G^t through edge editing (*e.g.,* removing the edge where the inter-agent interaction has limited contribution to the task).

3.4.1 Individual Evolution

Individual evolution uses individual-level SoA as feedback to improve agents' task execution. Guided by this feedback, each agent reflects on its predictions and outcome to generate an evolution signal that specifies prompt updates:

$$\theta^t \xrightarrow{\text{SoA as feedback}} \text{evolution signal}^t \xrightarrow{\text{prompt update}} \theta^{t+1}. \tag{6}$$

The reflection encourages higher anticipated achievement through concrete execution improvements while reducing prediction–outcome discrepancies. When execution falls short of expectations ($p^t > o^t$), the agent revises its task strategy; when execution exceeds expectations ($p^t < o^t$), it considers more ambitious predictions grounded in observed capabilities. Prompt templates are provided in Appendix A.2.

3.4.2 Topology Evolution

Adapting the topology helps an MAS accommodate changing coordination needs during task execution (Tastan et al., 2025). To achieve this, SOAP uses interaction-level SoA as internal feedback for topology updates. We use accumulated interaction-level SoA and the availability of new task-relevant information to guide three edge operations: 1) *deletion*, when an existing edge exhibits persistently low SoA and carries redundant information (*e.g.,* duplicate evidence sent to a verifier); 2) *addition*, when a new connection can supply missing task-relevant information (*e.g.,* a missing source from another searcher); and 3) *keep*, otherwise. Aggregation and detailed editing implementations are provided in Appendices A.4 and A.5.

4 Experiment

4.1 Experimental Settings

Benchmarks. We evaluate SOAP on BrowseComp-Plus (Chen et al., 2025), WorkBench (Styles et al., 2024), PlanCraft (Dagan et al., 2024), and CloudCast (Qi et al., 2026), covering information

Table 1: **Main Results.** Accuracy (%) on BrowseComp-Plus, WorkBench, and PlanCraft, and transfer cost on CloudCast. Best results are bolded; second-best results are underlined.

Method	BrowseComp-Plus ↑	WorkBench ↑	PlanCraft ↑	CloudCast ↓
Fixed-topology				
MAS-Centralized	55.0	59.7	65.8	766.2
MAS-Decentralized	41.0	61.4	68.0	747.0
Evolution with *external feedback* (train-time)				
MetaAgent	57.8	26.9	26.4	689.4
ADAS	45.8	47.6	58.4	775.2
AFlow	36.5	59.0	66.2	698.4
AgentSquare	40.5	62.8	68.7	717.0
ChatDev-Puppeteer	40.0	54.1	62.8	696.0
MaAS	50.0	61.0	68.1	<u>627.0</u>
ARG-Designer	46.0	53.8	60.7	660.6
SwarmAgentic	<u>58.5</u>	51.4	56.5	660.6
EoM	46.3	40.3	<u>73.6</u>	629.6
Evolution with *external feedback* (test-time)				
EvoAgent	36.0	60.3	65.8	**618.0**
MetaGPT	39.8	54.1	63.6	1035.0
AgentVerse	42.7	<u>63.4</u>	66.4	633.0
CORAL	41.0	33.5	45.9	636.5
Evolution with *internal feedback* (test-time)				
SelfOrg	54.8	48.3	53.0	**618.0**
MetaGen	56.0	30.3	52.3	646.1
SOAP (ours)	**67.0**	**68.3**	**76.5**	**618.0**

retrieval, workplace tool use, planning, and program optimization, respectively. We report accuracy on the first three benchmarks and total transfer cost on CloudCast. Benchmark descriptions and evaluation protocols are provided in Appendix B.1.

Baselines. We compare three groups of baselines: 1) *Static MAS*: MAS-Centralized and MAS-Decentralized, without feedback-driven evolution. 2) *External-feedback methods*: offline search or training methods, including MetaAgent (Zhang et al., 2025d), AgentSquare (Shang et al., 2024), AFlow (Zhang et al., 2024), ADAS (Hu et al., 2024), MaAS (Zhang et al., 2025a), SwarmAgentic (Zhang et al., 2025c), ARG-Designer (Li et al., 2025), and ChatDev-Puppeteer (Dang et al., 2025), use task-performance evaluations or supervision; EoM (Qi et al., 2026) uses environmental rewards; inference-time methods use LLM evaluations (AgentVerse (Chen et al., 2023) and EvoAgent (Yuan et al., 2025)) or execution-based feedback (MetaGPT (Hong et al., 2023) and CORAL (Qu et al., 2026)). 3) *Internal-feedback methods*: SelfOrg (Tastan et al., 2025) uses response-based peer contribution estimates. We additionally include MetaGen (Wang et al., 2026b), which combines internal consistency signals with execution checks.

Implementation Details. Unless otherwise specified, all results are obtained using Gemini-2.5-Flash, with the same task-specific tools provided to all methods within each benchmark. We use GPT-5.5 as the external judge for final rubric-based evaluation on BrowseComp-Plus, WorkBench, and PlanCraft; these scores are not supplied to methods during execution. CloudCast uses a programmatic simulator for evaluation. We use the same initial number of agents and the same initial workflow for all methods within each benchmark. Details are provided in Appendix B.2.

4.2 Overall Performance

Table 1 shows that SOAP achieves the best performance on BrowseComp-Plus, WorkBench, and PlanCraft, exceeding the strongest baselines, while achieving the lowest cost on CloudCast. Existing test-time evolution methods outperform offline-evolved baselines on WorkBench and CloudCast but fall behind on the other two benchmarks, suggesting that adaptation alone is insufficient without informative feedback. SelfOrg demonstrates the potential of internal feedback, matching the best

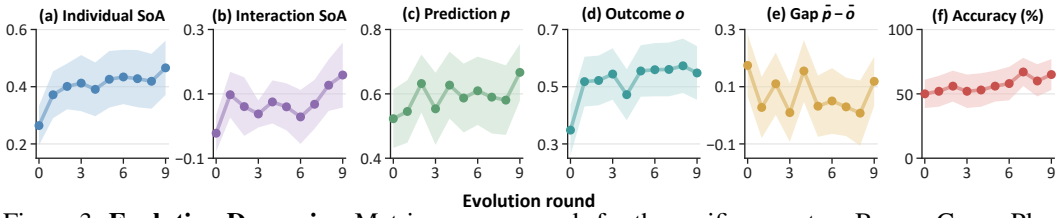

Figure 3: **Evolution Dynamics.** Metrics across rounds for the verifier agent on BrowseComp-Plus. Shading indicates the supplied mean ± error; panel (e) shows the signed prediction–outcome gap.

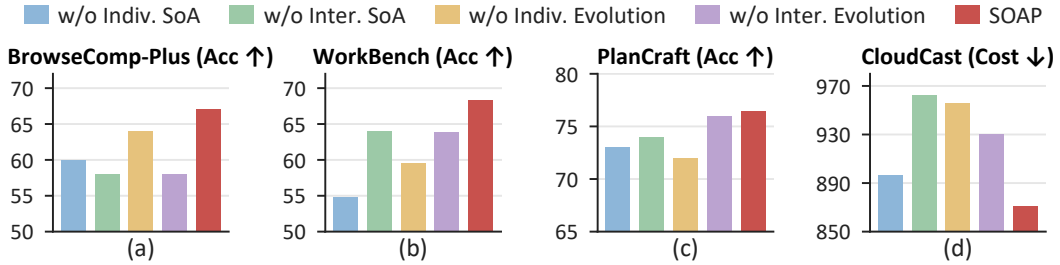

Figure 4: **Ablation Study.** "Indiv." and "Inter." denote "Individual" and "Interaction", respectively.

CloudCast cost, yet remains less competitive on other scenarios. SOAP's consistent performance suggests that rubric-based predictions and self-observed outcomes provide useful feedback across tasks. By distinguishing individual execution from downstream interactions, these signals may better identify actionable improvements than aggregate task feedback alone.

4.3 In-depth Analysis of SOAP

Dynamics of SoA During Evolution. To investigate whether agents develop a stronger sense of agency as the MAS evolves, we analyze task performance and agent SoA dynamics on BrowseComp-Plus. As shown in Figure 3, we observe that 1) *task performance improves* overall with evolution; 2) both *individual- and interaction-level SoA* show overall increases, with interaction SoA exhibiting greater relative fluctuations, potentially reflecting the additional uncertainty in anticipating another agent's behavior; 3) *predictions and assessed outcomes rise overall*, but predictions fluctuate more while outcomes improve relatively steadily. Although both use fixed rubrics, predictions are made before execution without direct evidence of the forthcoming outcome, whereas outcome assessments are grounded in completed execution. The sustained improvement in assessed outcomes is consistent with effective guidance from the evolution signals; and 4) the *prediction–outcome gap is generally smaller* than initially but fluctuates across rounds, potentially because expectations and execution improve at different rates. These observations support the usefulness of SoA as an internal signal for self-evolution and suggest that stronger agency accompanies better task performance.

Ablation Study. We remove each component of SOAP to analyze the effectiveness of two-level SoA and evolution as shown in Figure 4.[1] We can find that 1) removing any component degrades performance, supporting the value of both levels of SoA feedback and the evolution mechanisms that translate this feedback into system updates. 2) The importance of each evolution mechanism varies by task: BrowseComp-Plus benefits more from topology adaptation, whereas WorkBench and PlanCraft rely more on individual evolution, suggesting different demands for information exchange and local execution. 3) Individual SoA contributes more than interaction SoA on two of the three benchmarks. One possible explanation is that agents can better anticipate their own execution, whereas downstream outcomes introduce additional uncertainty from other agents' behavior.

Performance Across Task Types. Table 2 shows that SOAP performs best on the short and medium BrowseComp-Plus groups, while MAS-Centralized leads on the long group. In contrast, on WorkBench, SOAP slightly trails the baselines on single-domain tasks but outperforms both on

[1]Detailed settings of the ablation study are provided in Appendix B.2.

Table 2: **Performance Across Task Groups.** Accuracy (%) of SOAP and baselines on BrowseComp-Plus and WorkBench, grouped by length and number of domains, respectively. Best scores are bolded.

BrowseComp-Plus

Method	Short	Medium	Long
MAS-Centralized	50.0	54.2	**75.0**
ADAS	47.1	43.8	44.4
SOAP (Ours)	**70.6**	**64.6**	66.7

WorkBench

Method	1 Domain	2 Domains	≥3 Domains
AgentVerse	71.7	64.2	21.7
AgentSquare	**72.0**	60.4	30.0
SOAP (Ours)	69.3	**81.1**	**38.3**

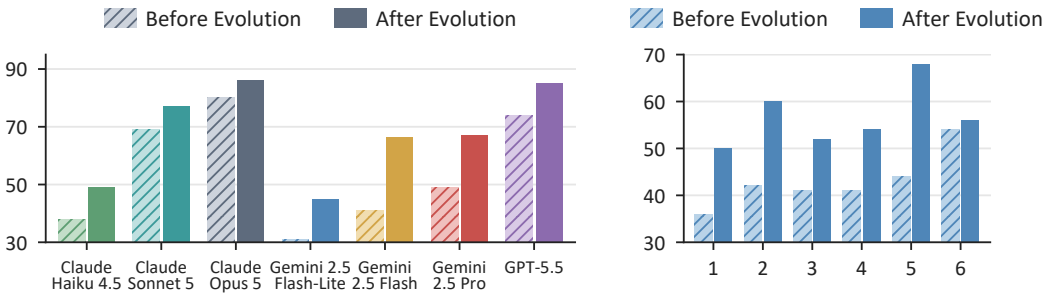

Figure 5: **Performance across Backbones and MAS Agent Count.** BrowseComp-Plus accuracy across (a) various LLM backbones and (b) initial team sizes, before (one round) and after evolution (ten rounds). Both y-axes start at 30%.

multi-domain tasks. One possible reason is that cross-domain requests create more opportunities for evolution to improve task decomposition and information exchange, whereas single-domain tasks offer less scope for such adaptation. These results suggest that SOAP's benefits depend on the coordination demands of a task, which length-based grouping alone may not capture. More results on PlanCraft are in Appendix B.3.

4.4 Robustness of SOAP

Robustness Across LLM Backbones. SOAP consistently improves over MAS without evolution across the evaluated backbones, suggesting that its benefits extend beyond a particular model family. Absolute performance nevertheless depends on the backbone: stronger variants within the Claude and Gemini families achieve higher accuracy after evolution. These results suggest that SoA-guided adaptation complements the underlying model's capabilities, while the backbone remains an important determinant of final performance. Results on other benchmarks are in Appendix B.3.

Robustness on Team Size. As shown in Figure 5(b), evolution improves accuracy at every evaluated team size, with the five-agent configuration achieving the highest accuracy and the largest improvement. Performance varies non-monotonically with team size: increasing the team from five to six agents reduces accuracy and yields a smaller gain from evolution. These results support the effectiveness of SOAP across different initial team sizes, while suggesting that additional agents can introduce coordination overhead rather than consistently improve performance.

5 Conclusion

We introduced SOAP, a framework that uses computational *Sense of Agency* as internal feedback for test-time MAS evolution. Agents compare predictions with self-assessed outcomes to guide prompt refinement and topology adaptation. Experiments across four benchmarks support its utility, while ablations highlight the complementary roles of individual and interaction feedback. Benefits vary across tasks and configurations, and additional evolution does not guarantee continued improvement. Our formulation is one feasible implementation of SoA, rather than a unique choice, and does not guarantee calibrated predictions. Future work will explore alternative SoA formulations, internalize SoA as a learned capability, and extend it to debate and mixed-motive interactions.

AI USE STATEMENT

In this work, we have not used generative AI tools for generating synthetic datasets, formulating mathematical claims, proposing or refining hypotheses, designing or providing feedback on research methodology or experiments, assisting with translation, cleaning and reformatting datasets, supporting qualitative and thematic data analysis, interpreting results. Providing critical ingredients for proving mathematical claims and assisting in the writing of proofs are not applicable to this work. Additionally, we used generative AI tools for creating or editing software code, writing refinement to improve readability, and identifying relevant literature. We have reviewed all AI-assisted work. In particular, all literature identified or listed with the assistance of generative AI tools was independently checked by the authors for accuracy and relevance. LLM-generated code was reviewed, tested, and modified by the authors to verify its correctness and ensure that it implemented the intended functionality. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REFERENCES

Anthropic. How we built our multi-agent research system, June 2025. URL https://www.anthropic.com/engineering/multi-agent-research-system. Engineering blog, published June 13, 2025.

Ian Beacock, Rachel Xu, Laura Murray, Patrick Anson, Beth Goldberg, Devika Kumar, Jun Lee, Rebekah Park, and Anoop Sinha. AI usage patterns are shaped by perceived gains in human agency, 2026. URL https://arxiv.org/abs/2607.02313.

Arthur Câmara, Vincent Slot, and Jakub Zavrel. Self-optimizing multi-agent systems for deep research, 2026. URL https://arxiv.org/abs/2604.02988.

Valérian Chambon, Nura Sidarus, and Patrick Haggard. From action intentions to action effects: how does the sense of agency come about? *Frontiers in Human Neuroscience*, 8:320, 2014. doi: 10.3389/fnhum.2014.00320. URL https://www.frontiersin.org/journals/human-neuroscience/articles/10.3389/fnhum.2014.00320/full.

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

Zijian Chen, Xueguang Ma, Shengyao Zhuang, Ping Nie, Kai Zou, Andrew Liu, Joshua Green, Kshama Patel, Ruoxi Meng, Mingyi Su, Sahel Sharifymoghaddam, Yanxi Li, Haoran Hong, Xinyu Shi, Xuye Liu, Nandan Thakur, Crystina Zhang, Luyu Gao, Wenhu Chen, and Jimmy Lin. BrowseComp-Plus: A More Fair and Transparent Evaluation Benchmark of Deep-Research Agent, 2025. URL https://arxiv.org/abs/2508.06600.

Gautier Dagan, Frank Keller, and Alex Lascarides. Plancraft: an evaluation dataset for planning with LLM agents, 2024. URL https://arxiv.org/abs/2412.21033.

Yufan Dang, Chen Qian, Xueheng Luo, Jingru Fan, Zihao Xie, Ruijie Shi, Weize Chen, Cheng Yang, Xiaoyin Che, Ye Tian, Xuantang Xiong, Lei Han, Zhiyuan Liu, and Maosong Sun. Multi-Agent Collaboration via Evolving Orchestration, 2025. URL https://arxiv.org/abs/2505.19591.

Patrick Haggard. Sense of agency in the human brain. *Nature Reviews Neuroscience*, 18:196–207, 2017. doi: 10.1038/nrn.2017.14. URL https://www.nature.com/articles/nrn.2017.14.

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, 2023. URL https://arxiv.org/abs/2308.00352.

Shengran Hu, Cong Lu, and Jeff Clune. Automated Design of Agentic Systems, 2024. URL https://arxiv.org/abs/2408.08435.

Yue Hu, Yuzhu Cai, Yaxin Du, Xinyu Zhu, Xiangrui Liu, Zijie Yu, Yuchen Hou, Shuo Tang, and Siheng Chen. Self-evolving multi-agent collaboration networks for software development. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/39af4f2f9399122a14ccf95e2d2e7122-Abstract-Conference.html.

Noam Karsh and Baruch Eitam. I control therefore i do: Judgments of agency influence action selection. *Cognition*, 138:122–131, 2015. doi: 10.1016/j.cognition.2015.02.002. URL https://www.sciencedirect.com/science/article/pii/S0010027715000268.

Yubin Kim, Ken Gu, Chanwoo Park, Chunjong Park, Samuel Schmidgall, A. Ali Heydari, Yao Yan, Zhihan Zhang, Yuchen Zhuang, Yun Liu, Mark Malhotra, Paul Pu Liang, Hae Won Park, Yuzhe Yang, Xuhai Xu, Yilun Du, Shwetak Patel, Tim Althoff, Daniel McDuff, and Xin Liu. Towards a science of scaling agent systems, 2025. URL https://arxiv.org/abs/2512.08296.

Kimi Team. Kimi K2.5: Visual agentic intelligence, 2026. URL https://arxiv.org/abs/2602.02276. Technical report.

Shiyuan Li, Yixin Liu, Qingsong Wen, Chengqi Zhang, and Shirui Pan. Assemble Your Crew: Automatic Multi-agent Communication Topology Design via Autoregressive Graph Generation, 2025. URL https://arxiv.org/abs/2507.18224.

J. W. Moore and P. C. Fletcher. Sense of agency in health and disease: A review of cue integration approaches. *Consciousness and Cognition*, 21(1):59–68, 2012. doi: 10.1016/j.concog.2011.08.010. URL https://pmc.ncbi.nlm.nih.gov/articles/PMC3315009/.

Lorina Puech, Karolina Moutsopoulou, and Lionel Brunel. Sense of agency and ideomotor learning: High dispositional sense of agency is linked to better action-effect learning. *Quarterly Journal of Experimental Psychology*, 79(3):769–794, 2026. doi: 10.1177/17470218251365225. URL https://journals.sagepub.com/doi/10.1177/17470218251365225.

Zhenting Qi, Huangyuan Su, Ao Qu, Chenyu Wang, Yu Yao, Han Zheng, Kushal Chattopadhyay, Guowei Xu, Zihan Wang, Weirui Ye, Vijay Janapa Reddi, Ju Li, Paul Pu Liang, Himabindu Lakkaraju, Sham Kakade, and Yilun Du. Economy of Minds: Emerging Multi-Agent Intelligence with Economic Interactions, 2026. URL https://arxiv.org/abs/2606.02859.

Ao Qu, Han Zheng, Zijian Zhou, Yihao Yan, Yihong Tang, Shao Yong Ong, Fenglu Hong, Kaichen Zhou, Chonghe Jiang, Minwei Kong, Jiacheng Zhu, Xuan Jiang, Sirui Li, Cathy Wu, Bryan Kian Hsiang Low, Jinhua Zhao, and Paul Pu Liang. CORAL: Towards autonomous multi-agent evolution for open-ended discovery, 2026. URL https://arxiv.org/abs/2604.01658.

Yu Shang, Yu Li, Keyu Zhao, Likai Ma, Jiahe Liu, Fengli Xu, and Yong Li. AgentSquare: Automatic LLM Agent Search in Modular Design Space, 2024. URL https://arxiv.org/abs/2410.06153.

Olly Styles, Sam Miller, Patricio Cerda-Mardini, Tanaya Guha, Victor Sanchez, and Bertie Vidgen. WorkBench: a Benchmark Dataset for Agents in a Realistic Workplace Setting, 2024. URL https://arxiv.org/abs/2405.00823.

Nurbek Tastan, Samuel Horvath, and Karthik Nandakumar. Stochastic Self-Organization in Multi-Agent Systems, 2025. URL https://arxiv.org/abs/2510.00685.

Kun Wang, Guibin Zhang, ManKit Ye, Xinyu Deng, Dongxia Wang, Xiaobin Hu, Jinyang Guo, Yang Liu, and Yufei Guo. MAS^2: Self-generative, self-configuring, self-rectifying multi-agent systems. In *The Fourteenth International Conference on Learning Representations*, 2026a. URL https://openreview.net/forum?id=qumy27hMDY.

Yimeng Wang, Jiaxing Zhao, Hongbin Xie, Hexing Ma, Yuzhen Lei, Shuangxue Liu, Xuan Song, Zichen Zhang, and Haoran Zhang. MetaGen: Self-Evolving Roles and Topologies for Multi-Agent LLM Reasoning, 2026b. URL https://arxiv.org/abs/2601.19290.

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.

Chen Xu, Yicheng Hu, Ruizi Wang, Xinyu Lin, Wenjie Wang, Dongrui Liu, and Fuli Feng. TacoMAS: Test-time co-evolution of topology and capability in LLM-based multi-agent systems, 2026. URL https://arxiv.org/abs/2605.09539.

Xiangyuan Xue, Yifan Zhou, Guibin Zhang, Zaibin Zhang, Yijiang Li, Chen Zhang, Zhenfei Yin, Philip Torr, Wanli Ouyang, and Lei Bai. CoMAS: Co-evolving multi-agent systems via interaction rewards. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=ihwAzktmWc.

Takahide Yoshida, Suzune Baba, Atsushi Masumori, and Takashi Ikegami. Minimal self in humanoid robot "Alter3" driven by large language model. In *ALIFE 2024: Proceedings of the 2024 Artificial Life Conference*, 2024. doi: 10.1162/isal_a_00778. URL https://doi.org/10.1162/isal_a_00778.

Siyu Yuan, Kaitao Song, Jiangjie Chen, Xu Tan, Dongsheng Li, and Deqing Yang. EvoAgent: Towards automatic multi-agent generation via evolutionary algorithms. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pp. 6192–6217, 2025. doi: 10.18653/v1/2025.naacl-long.315. URL https://aclanthology.org/2025.naacl-long.315/.

Guibin Zhang, Luyang Niu, Junfeng Fang, Kun Wang, Lei Bai, and Xiang Wang. Multi-agent Architecture Search via Agentic Supernet, 2025a. URL https://arxiv.org/abs/2502.04180.

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, 2024. URL https://arxiv.org/abs/2410.10762.

Shuning Zhang, Hui Wang, and Xin Yi. Exploring collaboration patterns and strategies in human-AI co-creation through the lens of agency: A scoping review of the top-tier HCI literature. *Proceedings of the ACM on Human-Computer Interaction*, 9(7):1–43, 2025b. doi: 10.1145/3757594. URL https://doi.org/10.1145/3757594.

Yao Zhang, Chenyang Lin, Shijie Tang, Haokun Chen, Shijie Zhou, Yunpu Ma, and Volker Tresp. SwarmAgentic: Towards Fully Automated Agentic System Generation via Swarm Intelligence, 2025c. URL https://arxiv.org/abs/2506.15672.

Yaolun Zhang, Xiaogeng Liu, and Chaowei Xiao. MetaAgent: Automatically Constructing Multi-Agent Systems Based on Finite State Machines, 2025d. URL https://arxiv.org/abs/2507.22606.

Mingchen Zhuge, Wenyi Wang, Louis Kirsch, Francesco Faccio, Dmitrii Khizbullin, and Jürgen Schmidhuber. GPTSwarm: Language agents as optimizable graphs. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 62743–62767. PMLR, 2024. URL https://proceedings.mlr.press/v235/zhuge24a.html.

A Method Details

A.1 SOAP Algorithm

A.2 LLM Prompts

We present prompt templates for rubric generation, pre-execution prediction, outcome assessment, and evolution signal generation. Placeholders in braces are filled with the query, agent configuration,

Algorithm 1 SOAP: SoA-Guided MAS Evolution

Input: Query query, initial configuration (Θ^0, G^0), interval K, stopping parameters ϵ, k

Generate fixed role-conditioned rubrics and assign $\mathcal{R}_{(i)}$ to each agent i
for $t = 0, 1, \ldots$ **do**
  // SoA Computation
  $(\text{query}, \Theta^t, G^t, \{\mathcal{R}_{(i)}\}) \xrightarrow{\text{predict}} \{p_i^t, p_{i\to j}^t\}$
  $(\text{query}, \Theta^t, G^t) \xrightarrow{\text{execute}} (\{y_i^t\}, \text{answer}^t)$
  $(\{y_i^t\}, \{\mathcal{R}_{(i)}\}) \xrightarrow{\text{assess}} \{o_i^t\}$
  $(p_i^t, o_i^t) \mapsto A_i^t, \quad (p_{i\to j}^t, o_j^t) \mapsto A_{i\to j}^t$ ▷ Eq.(4), Eq.(5)
  Break if mean individual SoA remains below ϵ for k consecutive measured rounds
  // SoA-Guided Evolution
  $(\theta_i^t, p_i^t, o_i^t, A_i^t, y_i^t) \xrightarrow{\text{reflect}} \text{evolution signal}_i^t \xrightarrow{\text{update}} \theta_i^{t+1}$ ▷ Eq.(6)
  Accumulate interaction-level SoA ▷ Appendix A.4
  $G^{t+1} \leftarrow G^t$
  if $(t+1) \bmod K = 0$ **then**
    $G^t \xrightarrow{\text{accumulated SoA and information availability}} G^{t+1}$ ▷ Section 3.4.2
  $\Theta^{t+1} \leftarrow \{\theta_i^{t+1}\}_{i \in V^{t+1}}$
return $\text{answer}^t, (\Theta^t, G^t)$

available messages, and execution records. These templates describe the prediction–outcome formulation of SOAP; the case study in Appendix A.3 traces prediction, execution, assessment, SoA computation, and the resulting prompt update. Numerical SoA scores are computed deterministically from predictions and assessed outcomes, rather than generated by an LLM.

Rubric Generation. Rubrics specify what successful task execution looks like for each role on the current query. They are generated once and remain fixed across rounds. An agent uses its own rubric for individual-level assessment; a sender uses the downstream receiver's rubric for interaction-level assessment. Thus, the two levels share the rubric-generation procedure rather than requiring independent, potentially inconsistent criteria. Binary outcomes are scored as 0 or 1; when graded outcomes use a 1–5 scale, they are normalized as $(v-1)/4$. The template below expresses the scoring anchors directly on $[0, 1]$ and imposes no fixed number of dimensions.

Template 1: Task- and role-conditioned rubric generation

```
A multi-agent team is working on this query:
{query}

AGENTS, ROLES, AND ASSIGNED TASKS:
{team_configuration}

For each role, define concrete rubric dimensions describing how
well that role fulfills its task on this query. Each dimension
must be assessable from execution outputs and available evidence.

Make the dimensions task-specific. For a searcher, consider the
relevance of candidates, supporting evidence, and source
traceability. For a verifier, identify the query constraints
that must be checked.

Give each dimension a unique ID, a checkable question, and
scoring anchors in [0,1]. Use binary scoring for clearly
checkable conditions and graded scoring for partial fulfillment.
Do not require access to a reference answer or external grader.
Use the same rubric for agents with the same role.
```

```
Return JSON only:
{
  "<role>": [
    {
      "id": "<dimension_id>",
      "question": "<aspect of task fulfillment>",
      "scale": "binary or graded",
      "anchors": {"<score>": "<criterion>"}
    }
  ]
}
```

Prediction Before Task Execution. At round t, agent i forecasts its own achievement p_i^t and each downstream receiver's achievement $p_{i\to j}^t$ before executing its task. Individual predictions use $\mathcal{R}_i$, while interaction predictions use $\mathcal{R}_j$. Forecasts are based on information available before execution and are held fixed when the resulting outcomes are assessed. The output schema is instantiated with the actual dimension and receiver IDs.

Template 2: Individual and interaction predictions

```
You are {agent_id}, about to execute your task on:
{query}

YOUR CURRENT CONFIGURATION:
{agent_configuration}

YOUR RUBRIC:
{own_rubric}

DIRECT DOWNSTREAM AGENTS, THEIR TASKS, AND THEIR RUBRICS:
{downstream_agents_and_rubrics}

Before acting, predict the achievement you expect at the end of
this round. Do not execute the task in this response.

1. Individual prediction: for each dimension of YOUR rubric,
   forecast how well you will fulfill it after your execution.
2. Interaction prediction: for each downstream agent, forecast
   how well it will fulfill EACH dimension of ITS rubric after
   receiving your planned output and executing its task.

Use the rubric anchors and return normalized scores in [0,1].
Base forecasts on your current context, history, and planned
work. Do not use post-execution results from this round or access
a reference answer. An interaction prediction concerns the
    receiver's achievement, not your
exclusive causal contribution to it.

Return JSON using the supplied dimension IDs:
{
  "individual_prediction": {"<own_dimension_id>": <score>},
  "interaction_prediction": {
    "<receiver_id>": {"<receiver_dimension_id>": <score>}
  }
}
If there are no downstream agents, return an empty object for
"interaction_prediction".
```

SoA Computation. After execution, each agent assesses its output y_i^t under its unchanged rubric, producing o_i^t. The assessment should be grounded in the execution record rather than adjusted to match the earlier prediction. The same procedure is applied to every receiver j; its assessment o_j^t is

reused for both its individual score and incoming interaction scores. No counterfactual assessment with the sender's message removed is required.

Template 3: Evidence-grounded outcome assessment

```
You are {agent_id}. Assess your completed execution on:
{query}

YOUR RUBRIC AND SCORING ANCHORS:
{own_rubric}

INFORMATION AVAILABLE DURING EXECUTION:
{execution_context}

YOUR EXECUTION OUTPUT AND TOOL OBSERVATIONS:
{execution_record}

For every rubric dimension, assess how well your completed
execution fulfills its criterion. Use normalized scores in
[0,1] and identify the evidence supporting each assessment.
Distinguish a supported conclusion from an unresolved claim.
Do not infer success merely from the effort spent or the length
of your response. Do not change rubric criteria or adjust scores
to agree with an earlier prediction. If the evidence does not
establish fulfillment, use the corresponding rubric anchor.

Return JSON using exactly the supplied dimension IDs:
{
  "outcome": {"<dimension_id>": <score>},
  "evidence": {"<dimension_id>": "<support or unresolved issue>"}
}
```

Given the recorded predictions and assessments, the system computes

$$A_i^t = \frac{1}{|\mathcal{R}_i|} \sum_{d \in \mathcal{R}_i} p_{i,d}^t \big(1 - |p_{i,d}^t - o_{i,d}^t|\big), \tag{7}$$

$$A_{i \to j}^t = \frac{1}{|\mathcal{R}_j|} \sum_{d \in \mathcal{R}_j} p_{i \to j,d}^t \big(1 - |p_{i \to j,d}^t - o_{j,d}^t|\big). \tag{8}$$

All rubric dimensions are included in the average, including those with zero predictions. The numerical scores are internal feedback; the evolution signal is the subsequent instruction for changing the agent's behavior.

Evolution Signal Generation. For individual evolution, the agent reflects on the query, configuration, execution record, predictions, assessed outcomes, and computed SoA. It produces an actionable instruction for the next-round task prompt. Reflection targets execution improvements and better-grounded expectations; changing predictions alone is not an execution improvement.

Template 4: SoA-guided individual evolution

```
You are {agent_id}, reflecting on your completed round on:
{query}

YOUR CURRENT CONFIGURATION AND RUBRIC:
{agent_configuration}
{own_rubric}

YOUR EXECUTION OUTPUT AND TOOL OBSERVATIONS:
{execution_record}

YOUR PRE-EXECUTION PREDICTIONS, ASSESSED OUTCOMES, AND
COMPUTED INDIVIDUAL SoA:
```

```
{individual_prediction}
{assessed_outcome}
{individual_soa}

Use this feedback to write a concise instruction for your next
round. Name the rubric dimension or task requirement it targets.

- Where prediction exceeded outcome, identify the execution
  shortcoming and a concrete change that could address it.
- Where outcome exceeded prediction, consider whether the
  observed evidence supports a more ambitious expectation.
- Preserve successful behavior and address unresolved needs.
  Do not invent a failure when the evidence shows success.
- Specify what to start, stop, or change in task execution.
  Do not merely restate the task or inflate predicted scores.

Return JSON only:
{"evolution_signal": "<one or two actionable imperative sentences>"}
```

For topology evolution, the system uses accumulated interaction SoA and the availability of new task-relevant information to edit edges every K rounds. It removes an eligible edge when low SoA coincides with redundant information, adds a connection that can supply missing information, and otherwise retains the topology. These decisions are rule-based in the described procedure, so there is no additional LLM edge-editing prompt. The aggregation and editing routines are specified in the corresponding implementation subsections.

A.3 An Example SoA Computation and SoA-guided Evolution.

We use BrowseComp-Plus instance 39 and verifier `agent_3` at round 8 of an execution log. The query asks for an individual whose publishing career began in 1999 and who founded two energy-focused magazines between 2000 and 2010, with further constraints concerning an award, a university publication, and government consulting. This is an abbreviated task description, not the full benchmark query. The verifier receives evidence from searchers and checks candidate Dianne R. Sutherland against five binary dimensions, summarized in Table 3.

Table 3: Recorded verifier predictions and assessments at round 8. Rubric descriptions are abbreviated; all five dimensions are binary.

ID	Criterion	p	o	$p(1-\lvert p-o\rvert)$
v1	Publishing career began in 1999.	1	1	1
v2	Founded exactly two energy-focused magazines between 2000 and 2010.	1	1	1
v3	Was an award finalist in 2019.	1	1	1
v4	Contributed a 2018 article on the relevant state to the Faculty Journal Articles of a university founded in 1919.	0	0	0
v5	Consulted for government agencies and gave an update to a foreign affairs subcommittee in 2014.	0	1	0

Prediction and execution. Before execution, the verifier forecasts $(1, 1, 1, 0, 0)$. Its subsequent output cites document 89533 for the 1999 publishing start and document 64485 for the magazine launches in 2002 and 2008. Its recorded self-assessment is $(1, 1, 1, 0, 1)$, while the university-publication condition remains unresolved.

Individual SoA. Applying Eq. 7 gives

$$A_3^8 = \frac{1+1+1+0+0}{5} = 0.60. \tag{9}$$

Although the agent assesses v5 as fulfilled, its prediction for that dimension was zero, so the dimension contributes zero to SoA. The unresolved v4 also contributes zero. This distinguishes anticipated achievement from outcome assessment alone: mean assessed achievement is 0.80, whereas SoA is 0.60.

Evolution signal and prompt update. The verifier uses the discrepancy on `v5` and the unresolved requirement `v4` to revise its next-round task instruction. The update calls for consolidating existing evidence before forecasting and directing further retrieval toward the missing requirement:

Next-round task instruction

```
Build a criterion-by-criterion evidence checklist from available
messages and prior findings before forecasting. Reconcile the
evidence for v5, direct further retrieval to the unresolved
university-publication condition v4, and report confirmed and
unresolved criteria separately to the summary agent.
```

Interaction-level assessment. For edge `agent_3→agent_4`, the verifier predicts the achievement of the downstream summary agent under the receiver's rubric before execution. After execution, the summary agent assesses its outcome under the same rubric. The three dimensions concern claim support, coverage of query constraints, and explicit reporting of unresolved evidence. Table 4 presents the paired predictions and assessments used to compute interaction-level SoA in Eq. 8.

Table 4: Interaction-level predictions, receiver assessments, and SoA scores at round 8. The sender predicts under the summary agent's rubric.

Receiver criterion	$p^{8}_{3\to4,d}$	$o^{8}_{4,d}$	Score
Evidence supports the candidate claims.	0.75	0.50	0.5625
Query constraints are covered in the summary.	0.75	0.50	0.5625
Unresolved evidence is explicitly identified.	1.00	1.00	1.0000

The resulting interaction SoA is $(0.5625 + 0.5625 + 1)/3 \approx 0.7083$. This measures agreement between the verifier's anticipated downstream achievement and the receiver's assessment; it does not isolate the verifier's causal contribution. The edge is then evaluated using its accumulated history and whether its messages supply new task-relevant information. Neither deletion nor addition follows from this single score alone. For example, duplicate verification messages combined with persistently low SoA could support deletion, whereas a verifier holding a missing source could motivate a new connection to a searcher.

A.4 Temporal Aggregation.

The purpose of this component is to prevent an agent whose contribution is only exploited in a later round from being reported as unproductive. Per-round scores are therefore accumulated into discounted returns $\bar{A}_i$ and $\bar{A}_{i\to j}$, together with observation counts $n_{i\to j}$, and the averages are taken over the rounds in which a claim was actually made, so that "never measured" is not confused with "measured and found unproductive". These three tables, together with the per-round history behind them, constitute the agency sequence consumed by Section 3.4;

A.5 Topology Evolution

Every K rounds, we review the collaboration graph using accumulated interaction-level SoA and marginal information coverage. Marginal coverage measures how much a sender's output covers the receiver's unmet rubric dimensions beyond information available from other sources. We prioritize deletion over addition and apply at most one edge edit per review.

Edge Deletion. An existing edge is eligible for deletion when its accumulated interaction-level SoA is at or below the weak-edge threshold τ_{SoA} and its marginal coverage is at most 0.02. To avoid premature removal, the edge must have existed for at least four rounds and have at least two SoA observations. Deletion must also preserve at least one outgoing connection for the sender. These conditions identify connections that are both persistently weak under SoA feedback and informationally redundant.

Edge Addition. If no deletion is selected, we consider absent edges and select the admissible candidate with the largest marginal coverage, provided that coverage is at least 0.15. The receiver's resulting in-degree must not exceed three, and additions are subject to the configured limit of two. Since an absent edge has no interaction history, its initial selection relies on potential information benefit; subsequent retention is informed by the interaction-level SoA accumulated after the connection is established.

Retention. If neither operation is admissible, the graph remains unchanged. In particular, low SoA alone does not justify removal when the sender still provides information needed by the receiver.

B Experimental Details

B.1 Benchmarks and Evaluation Metrics

BrowseComp-Plus. BrowseComp-Plus (Chen et al., 2025) evaluates information retrieval and reasoning over a fixed document corpus. Queries require identifying relevant evidence, resolving ambiguous candidates, and combining information across sources. We report accuracy (%), computed by averaging per-query correctness scores from a GPT-5.5 rubric evaluator with two items (correctness and contradiction), yielding scores in $\{0, 0.5, 1\}$.

WorkBench. WorkBench (Styles et al., 2024) evaluates workplace task execution through tools for applications such as email, calendars, and project management. Tasks require selecting appropriate tools and coordinating multiple actions to achieve a requested outcome. We report accuracy (%) over a 345-instance test split randomly sampled from 690 converted instances with seed 42. using GPT-5.5 rubric evaluation for the former and exact-match evaluation for the latter, with the two subsets reported separately, to determine whether each request is completed correctly.

PlanCraft. PlanCraft (Dagan et al., 2024) evaluates planning in a Minecraft-style crafting environment. Given an initial inventory and a target item, agents must coordinate crafting actions under resource constraints or recognize that the target is infeasible. We follow the PlanCraft experimental and evaluation setup of TacoMAS (Xu et al., 2026). We report accuracy (%) using a GPT-5.5 evaluator with two rubric items yielding scores in $\{0, 0.5, 1\}$.

CloudCast. Following EoM (Qi et al., 2026), CloudCast evaluates iterative optimization of a broadcast-routing program in a multi-region, multi-cloud environment. The program constructs routing topologies from network information, including transfer costs and throughput. We report the final valid program's total data-transfer cost across the evaluation scenarios, where lower values indicate better performance. We use five fixed scenarios (intra-AWS, intra-Azure, intra-GCP, inter-cloud AGZ, and inter-cloud GAZ2) evaluated by the bundled network simulator and aggregate results using the arithmetic mean of per-trial total costs.

Evaluation and Evolution Feedback. Benchmark evaluation is separate from SOAP's evolution loop. For SOAP, reference answers and benchmark scores are used only to evaluate final outputs; they are not provided to SOAP for agent updates, topology editing, or termination.

B.2 Implementation Details

Agent Initialization. Each run starts with five agents: a planner, two searchers, a verifier, and a summarizer. The initial graph contains six directed edges: planner-to-searcher connections for both searchers, both searchers to the verifier, and connections from one searcher and the verifier to the summarizer. Role instructions are adapted to each benchmark. Each query starts from the same initial configuration and evolves independently.

Backbones and Inference Budgets. We use Gemini-2.5-Flash with temperature 0 on BrowseComp-Plus, WorkBench, and PlanCraft. Each run allows up to 10 evolution rounds and six tool calls per agent per round. The maximum number of iterations per agent is 20 on BrowseComp-Plus and PlanCraft and 10 on WorkBench. Rubric-generation and assessment calls use the provider-default temperature,



Figure 6: Performance change across different evolution rounds on four benchmarks.

with token budgets of 8,000 and 12,000, respectively. GPT-5.5 performs final rubric-based evaluation where applicable. Baselines use the same execution backbone and grading infrastructure on each benchmark.

Evolution Settings. Topology reviews occur every $K = 2$ rounds. Edge addition requires marginal information coverage of at least 0.15, subject to a maximum receiver in-degree of three and at most two additions per instance. Edge deletion requires low accumulated interaction-level SoA, marginal coverage of at most 0.02, an edge age of at least four rounds, and at least two observations. Deletion must preserve at least one outgoing connection for the sender. The topology tie-breaking seed is 1. SOAP's external meta-controller is disabled, and agent updates are generated through SoA-guided self-reflection.

CloudCast Settings. CloudCast consists of a single program-optimization task evaluated over independent trials. Each trial starts from the same seed program in a fresh workspace shared by all agents. For the main CloudCast comparison in Table 1, we use Claude Sonnet 5 as the execution backbone for all methods, with a temperature of 1, and Claude Opus 5 as the meta model. The choice of execution backbone is motivated by preliminary experiments in which Gemini-2.5-Flash produced programs with higher costs than the seed program. Each trial allows up to 10 rounds and 25 tool calls per agent per round. Available tools are `read_file`, `write_file`, read-only `shell`, `request_eval`, and `done`. Verifier and shell timeouts are 900 and 60 seconds, respectively. The simulator validates routing topologies and sums data-transfer costs over five fixed scenarios: intra-AWS, intra-Azure, intra-GCP, inter-cloud AGZ, and inter-cloud GAZ2. Valid programs receive the normalized score $1 - \text{cost}/1035$, where 1035 is the seed-program cost. Final evaluation considers both the workspace program and any program in the final response, retaining the better result. We report the mean total cost across independent trials.

Ablation Settings. The CloudCast ablation in Figure 4 uses Claude Haiku as the execution backbone, whereas the main comparison in Table 1 uses Claude Sonnet 5. Its absolute costs should therefore be interpreted within the ablation setting rather than directly compared with the main-table costs.

Feedback Access and Evaluation. On BrowseComp-Plus, WorkBench, and PlanCraft, reference answers and external grading scores are reserved for final evaluation and are not supplied to SOAP's update loop. Baselines likewise receive no reference-based judge scores during execution and use their configured self-reflection or internal candidate-evaluation mechanisms. On CloudCast, all methods can access programmatic simulator feedback through `request_eval`; no LLM judge is used. Dataset subsets and final scoring protocols are described in Appendix B.1.

B.3 Additional Results

Effect of evolution rounds. We examine whether allowing more evolution rounds improves MAS performance. We vary the maximum round budget from 2 to 10 and compare against performance before evolution. A two-round budget improves performance on all four benchmarks, but larger

Table 5: Accuracy (%) on PlanCraft grouped by the number of crafting steps. Best results within each group are bolded.

Method	1 Step	2 Steps	≥3 Steps
AgentVerse	77.5	**70.0**	**28.6**
MAS-Decentralized	70.0	43.3	14.3
SOAP	**92.5**	**70.0**	**28.6**

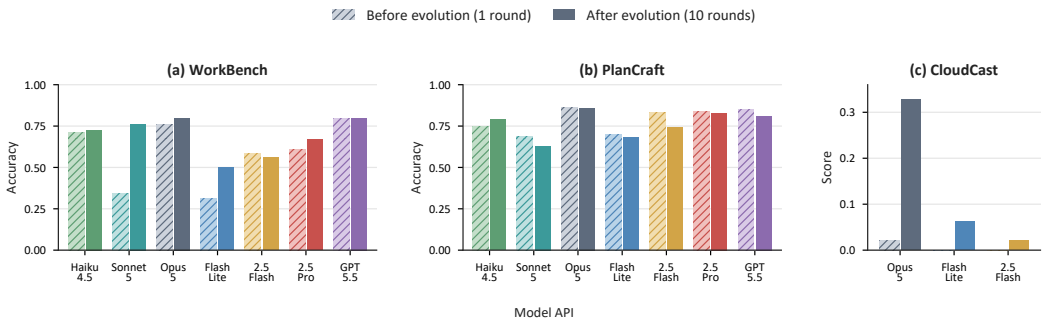

Figure 7: Performance of SOAP across seven LLM backbones before and after evolution. Workbench and PlanCraft report accuracy; CloudCast reports the normalized score $1 - \text{cost}/1035$, where 1035 is the seed-program cost. Higher values indicate better performance.

budgets do not consistently yield further gains, with some settings on Workbench and PlanCraft falling below their pre-evolution baselines. One possible explanation is that imperfect internal feedback can lead subsequent updates to disrupt effective agent strategies or coordination, so additional evolution does not guarantee better task performance.

Results across crafting steps on PlanCraft. Table 5 shows that SOAP outperforms both baselines on single-step tasks and matches AgentVerse on tasks requiring two or more steps, while consistently exceeding MAS-Decentralized. Unlike the gains on multi-domain WorkBench tasks, SOAP's advantage over the strongest baseline is concentrated in simpler crafting tasks. A possible explanation is that short tasks allow local execution feedback to translate more directly into actionable corrections, whereas longer crafting sequences require resolving dependencies and managing intermediate resources that local SoA assessments may not fully capture. These results suggest that improving long-horizon planning remains a challenge for SoA-guided evolution.

Robustness across LLM Backbones. We evaluate SOAP across seven LLM backbones to examine whether its benefits extend beyond a single model. On Workbench, evolution improves accuracy for five backbones, preserves GPT-5.5's performance, and slightly reduces Gemini 2.5 Flash's accuracy. The largest gains occur with Claude Sonnet 5 and Gemini Flash Lite, at 41.7 and 19.0 percentage points, respectively, indicating that the benefits of evolution extend across multiple backbones but vary in magnitude. CloudCast exhibits both floor and saturation effects: some backbones remain near the seed-program score despite evolution, whereas others reach the best observed performance within one round. We therefore display Claude Opus 5, Gemini Flash Lite, and Gemini 2.5 Flash in Figure 7(c), focusing on backbones that show measurable gains from additional evolution.