

Collaborative OS of Multi-Agent Systems

Jiazheng Kang¹, Ruochen Zhao¹, Gezi Li², Dongying Lin²,
Ting Bai^{1*}

¹Beijing University of Posts and Telecommunications

²Huawei Technologies Co., Ltd.

{kjjz,zhaorc,baiting}@bupt.edu.cn

{ligezi,lindongying}@huawei.com

Abstract

Large language model-driven Multi-Agent Systems (MAS) face two key challenges in long-horizon tasks: collaboration workflows struggle to adapt to dynamic task states, and accumulated inter-agent interactions lead to lengthy, noisy contexts that degrade model performance and efficiency. To address these challenges, we propose **MAS-OS**, a collaborative Operating System (OS) for MAS. MAS-OS consists of three core modules: *Memory Manager*, *Agent Scheduler*, and *Context Allocator*. The Memory Manager maintains a structured memory pool and a global summary of agents' interaction history. Based on the evolving memory states, the Agent Scheduler selects the next active agent, while the Context Allocator provides task-relevant memories to support effective context utilization. Experiments on four benchmark datasets demonstrate the effectiveness of MAS-OS. Compared with SOTA baselines, MAS-OS achieves an average improvement of 17.42% under the Llama-3.1-8B backbone. The code is available at <https://github.com/BAI-LAB/MAS-OS>.

1 Introduction

Multi-Agent Systems (MAS) have emerged as a prevailing framework wherein specialized LLM-driven agents collaborate to address complex, long-horizon problems through decomposition into role-specific responsibilities (Qian et al., 2024; Wu et al., 2024). While existing MAS frameworks have shown that such agent collaboration can outperform monolithic models, the effectiveness of these systems is frequently bottlenecked by inefficient information distribution among agents. Specifically, as the volume of inter-agent collaboration traces expands, ensuring that each agent accesses task-critical information while filtering out irrelevant details has emerged as a critical and unresolved challenge.

*Corresponding author.

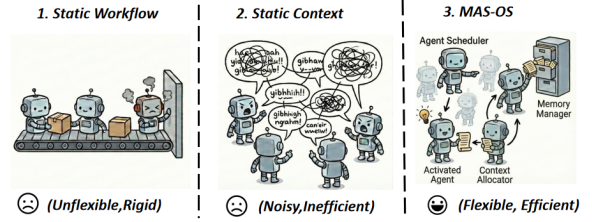


Figure 1: Comparison of collaboration paradigms in different MAS. (1) Rigid static workflows, (2) Noisy static context, (3) MAS-OS is a flexible collaborative operating system for MAS, enabled by contextual memory sharing and automatic agent scheduling.

Most existing multi-agent systems rely heavily on static, hand-crafted, predefined multi-agent workflows, where the sequence of participating agents is predetermined by inflexible rules or predefined graphs (Hong et al., 2024). Two primary message-passing paradigms are commonly adopted. The first is full-context broadcasting, distributing the entire conversation history to all agents (Wu et al., 2024). While ensuring completeness, it inundates agents with redundant content, escalating token costs and cognitive noise while triggering the "lost-in-the-middle" effect that impairs reasoning performance. The second is direct context passing (Qian et al., 2024), which confines each agent to the preceding step's output or a narrow fixed window. While computationally efficient, this induces contextual isolation: agents lose access to global objectives, historical feedback, and long-range dependencies, fragmenting decision-making and undermining consistency. This static-dependent collaboration paradigm in MAS lacks adaptability to dynamic tasks or unexpected errors, preventing timely agent invocation upon plan deviation and decoupling agent scheduling from memory state.

Recent research seeks to mitigate these limitations through dynamic context management and adaptive collaboration. For example, RCR-Router (Liu et al., 2025) dynamically allocates con-

text but retains a static interaction topology, limiting its adaptability to evolving task states. Puppeteer (Dang et al., 2026) dynamically orchestrates agents with a centralized controller, yet lacks fine-grained inter-agent information sharing. Other recent methods improve MAS adaptivity through task routing (Yue et al., 2025), redundant-agent elimination (Wang et al., 2025b), or next-agent prediction with context selection (Wang et al., 2025a). Nevertheless, jointly optimizing dynamic agent scheduling and fine-grained context allocation remains a challenging and underexplored problem.

To address these challenges, we propose **MAS-OS**, a collaborative Operating System (OS) framework in MAS that departs from conventional approaches by integrating scheduling (agent selection) and communication (context management) into a unified architecture. Instead of treating the two as separate tasks, MAS-OS coordinates these core mechanisms based on agents’ memory states, thereby enabling more coordinated and adaptive multi-agent collaboration. Specifically, MAS-OS comprises three integrated components acting as the OS kernel: (1) **Agent Scheduler**: it observes the global summary maintained by Memory Manager and dynamically activates the next agent based on the current system state; (2) **Context Allocator**: it dynamically filters the information from the MemoryPool to deliver a compact, agent-conditioned subset of context, ensuring the active agent receives the relevant information needed for the current operation. (3) **Memory Manager**: it systematically structures raw interactions into a graph-based MemoryPool, while continuously maintaining a rolling global summary to furnish the OS with real-time oversight of task progress. Additionally, reinforcement learning (RL) jointly optimizes the Agent Scheduler and Context Allocator conditioned on agent memory to enhance multi-agent collaboration. Our main contributions are summarized as follows:

- We propose a novel framework MAS-OS, which frames multi-agent collaboration as an OS-level resource management problem within a collaborative framework, featuring unified memory management, agent scheduling, and context routing.
- We make a prior attempt to optimize multi-agent scheduling and allocation within a reinforcement learning (RL) framework, thereby

realizing adaptive resource orchestration and boosting collaboration effectiveness.

- Extensive experiments on four benchmarks, demonstrate that MAS-OS significantly outperforms strong baselines (e.g., MacNet, EvoAgent) and rule-based approaches, achieving an average improvements of 17.42% on Llama-3.1-8B.

2 Related Work

2.1 Multi-Agent Systems (MAS)

LLM-based multi-agent systems (MAS) have become an effective paradigm for solving complex tasks by decomposing problem solving across multiple specialized agents (Ling et al., 2025). Early systems rely on manually designed roles and fixed interaction protocols, such as CAMEL and AutoGen for role-playing or dialogue-based coordination (Li et al., 2023; Wu et al., 2024), and MetaGPT and ChatDev for software-engineering-style workflows (Hong et al., 2024; Qian et al., 2024). Although effective, these predefined workflows are difficult to adapt to evolving task states. Recent work therefore explores adaptive MAS mechanisms, including dynamic orchestration or evolution of agents (Liu et al., 2024; Qian et al., 2025; Yuan et al., 2025; Dang et al., 2026), task and role routing (Yue et al., 2025), redundant-agent elimination (Wang et al., 2025b), and flexible or self-organized collaboration (Wang et al., 2025a; Zhou et al., 2025). However, most existing MAS methods mainly optimize *who* should participate or *how* agents should communicate, while the exchanged context is still handled at a coarse granularity, such as full histories, complete messages, or fixed summaries.

2.2 Multi-Agent Memory Management

Efficient context management is crucial for MAS: unrestricted information sharing may introduce redundancy and noise, while overly limited context can prevent agents from accessing useful global progress. Prior work improves agent memory by externalizing interaction histories into retrievable structures, such as memory streams (Park et al., 2023), hierarchical memory systems (Kang et al., 2025; Packer et al., 2023), and long-term memory mechanisms (Xu et al., 2026; Fang et al., 2025; Chhikara et al., 2025). In multi-agent settings, memory management further requires deciding what information should be exposed to which

agent during collaboration. Existing methods organize memories across agents or interaction levels (Zhang et al., 2026; Xu et al., 2025), or route role-aware memories according to agent needs (Liu et al., 2025). Recent work such as AnyMAC also considers selective access to previous context when predicting the next agent (Wang et al., 2025a). Nevertheless, most memory and context-routing methods depend on fixed schedules, predefined roles, or heuristic retrieval, so context allocation is usually separated from dynamic agent orchestration.

2.3 Reinforcement Learning for Agents

Reinforcement learning (RL) provides a natural framework for improving agents through interaction feedback and has been applied to alignment, tool use, and agent-level self-improvement (Ouyang et al., 2022a; Nakano et al., 2021; Zeng et al., 2024; Shinn et al., 2023). Since MAS collaboration often involves long-horizon decisions and delayed outcomes, recent work applies learning-based optimization to agent collaboration and orchestration. For example, Optima (Chen et al., 2025) improves the effectiveness and efficiency of LLM-based MAS, Puppeteer (Dang et al., 2026) trains a centralized orchestrator for dynamic agent sequencing, and ReSo (Zhou et al., 2025) uses collaborative rewards for self-organized agent selection. Other studies explore RL for specialized agentic reasoning and multi-agent settings (Lan et al., 2025; Zeng et al., 2025; Xue et al., 2025). However, most RL-based methods mainly focus on final task success, agent selection, or high-level orchestration, while joint optimizations of agent scheduling and context routing according to evolving collaboration states remains underexplored.

3 Methodology

3.1 Problem Definition

We formulate multi-agent collaboration as a memory-driven sequential decision-making problem, with its core anchored in system-level agent scheduling and context allocation mechanisms.

Given a task x and a set of available agents $A = \{a_1, a_2, \dots, a_n\}$, where each agent a_i is equipped with specific capabilities (e.g., reasoning, memory retrieval, task execution), the aim of a Multi-Agent System (MAS) is to generate an optimal agent execution sequence that processes the task x and yields a valid solution y while optimizing a predefined performance metric (e.g., accuracy,

latency, cost efficiency). Formally, we define the agent execution sequence as:

$$\pi = [a_{\sigma 1}, a_{\sigma 2}, \dots, a_{\sigma T}], \quad (1)$$

where $\sigma t \in \{1, 2, \dots, n\}$ and $t \in \{1, 2, \dots, T\}$. T denotes the length of the execution sequence, and $a_{\sigma t}$ represents the index of the agent selected at the t -th step.

The optimization objective of the MAS is then formulated as:

$$\arg \max_{\pi} \mathcal{F}(\mathcal{T}(x, \pi, M_{\pi}), y^*), \quad (2)$$

where $\mathcal{T}(x, \pi, M_{\pi})$ denotes the output generated by executing the task x following the agent sequence π , y^* denotes the ground-truth solution of task x . M_{π} denotes the subset of memory resources allocated to the agent sequence π . $\mathcal{F}$ denotes the performance metric function (e.g., accuracy) that quantifies the alignment between the generated output and the ground truth.

3.2 MAS-OS

To enable dynamic collaboration and efficient information sharing among multiple agents, we propose **MAS-OS**, a memory-centric collaborative Operating System (OS) framework designed to manage the lifecycle of agent memory in Multi-Agent Systems (MAS). MAS-OS integrates three core components, each with distinct functionalities:

- **Memory Manager:** Acts as the memory handler, maintaining a structured MemoryPool and a rolling global summary of agents' interaction history to ensure state preservation.
- **Agent Scheduler:** Orchestrates agents in a memory-driven manner, selecting the next active agent to advance the task based on the memory output by the Memory Manager.
- **Context Allocator:** Ensures context utilization by allocating an agent-relevant memory subset to the active agent.

3.2.1 Memory Manager Component

The Memory Manager maintains a shared MemoryPool to manage the memory state for agent scheduling (Agent Scheduler) and context allocation (Context Allocator).

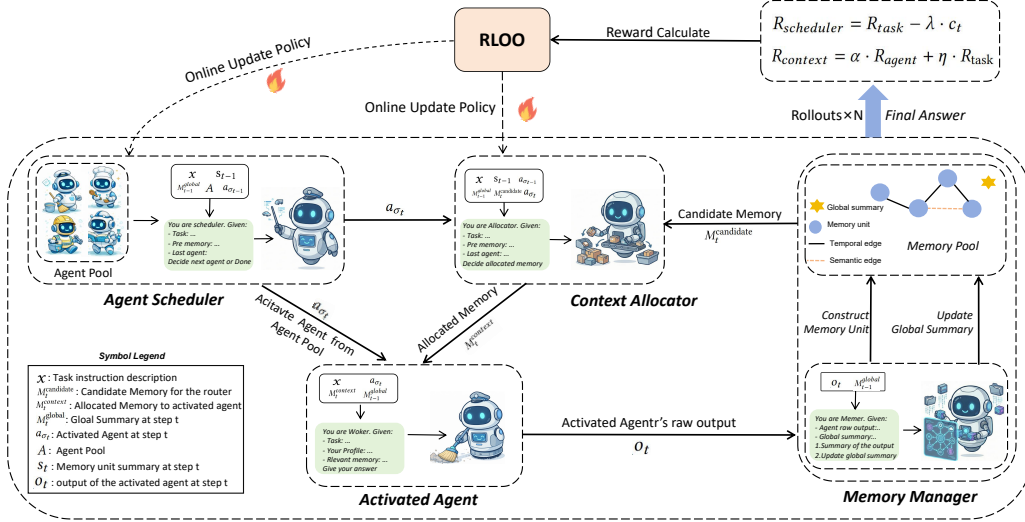


Figure 2: **Overview of the MAS-OS framework.** The system functions as a memory-centric collaborative operating system comprising three core components: (1) **Memory Manager** maintains the *MemoryPool* and updates the rolling global summary; (2) **Agent Scheduler** activates the next agent based on the current memory state; and (3) **Context Allocator** allocates a relevant memory context to the active agent.

MemoryPool. The MemoryPool is a continuously accumulating shared memory repository for MAS, which is structured as a memory graph comprising memory units, semantic edges, temporal edges, and global summaries.

The Memory Graph is a dynamically constructed that evolves with the interaction traces generated during multi-agent task execution, defined as:

$$G_t = (V_t, E_t^T, E_t^S), \quad (3)$$

where V_t is the memory node set in the interaction trace produced at step t . Each node is termed as a memory unit that stores a key-value pair (o_t, s_t) . Here, o_t retains the raw information generated by the active agent, s_t denotes a summary memory of raw information that serves as a high-level representation of the memory state at this step. E_t^T and E_t^S denote two distinct edge sets, corresponding to temporal edges and semantic edges, respectively, to support memory retrieval based on both recency and semantic relevance.

Temporal edges record the associations between memory outputs generated by adjacent agent execution steps. A new temporal edge connects the current memory node to its immediate predecessor, following the update rule below:

$$E_t^T = E_{t-1}^T \cup e_{(v_{t-1}, v_t)}, \quad (4)$$

where $e_{(v_{t-1}, v_t)}$ denotes the temporal edge constructed between memory node v_t and v_{t-1} at step t .

Semantic edges link the current node to relevant historical memory nodes to support long-term dependency tracking. A semantic edge is added between node v_t and previous memory node v_i if the similarity of their summaries exceeds a predefined threshold τ , defined as:

$$E_t^S = E_{t-1}^S \cup \{e_{(v_i, v_t)} \mid i < t, \text{sim}(s_i, s_t) \geq \tau\}, \quad (5)$$

where $\text{sim}(s_i, s_t)$ represents the cosine similarity.

Global Summary. Complementing individual memory nodes in the MemoryPool, a rolling global summary M_t^{global} is a compact natural-language representation that encapsulates task progress up to step t . Functioning as the system's conscious state, it acts as the primary signal to drive agent coordination and scheduling decisions.

The MemoryPool M_t^{Pool} , composed of a memory graph and a global summary, is formally defined as follows:

$$M_t^{\text{Pool}} = \{G_t, M_t^{\text{global}}\}, \quad (6)$$

where G_t and M_t^{global} represent the memory graph and global summary that built from interaction memories yielded by all executing agents up to step t .

3.2.2 Agent Scheduler Component

The Agent Scheduler coordinates agent collaboration in a memory-driven manner by selecting the next active agent based on the current memory

state maintained in the MemoryPool. The agent scheduler determines the next agent for execution. It draws on the historical memory of agents that have been activated in preceding steps, while conditioning its decision on four key elements: the task instruction x , the global summary in Memory-Pool M_{t-1}^{global} , the previous active agent $a_{\sigma_{t-1}}$ and its summary memory of raw information s_{t-1} .

Given the candidate agent pool $A = \{a_1, \dots, a_n\}$, the scheduler observation can be formulated as:

$$o_t^{\text{sch}} = \left[x, M_{t-1}^{\text{global}}, s_{t-1}, a_{\sigma_{t-1}}, A \right]. \quad (7)$$

The scheduler then samples the index of the next agent from a stochastic scheduling policy:

$$\sigma_t \sim \pi_\theta(\cdot | o_t^{\text{sch}}) | a_{\sigma_t} \in A, \sigma_t \in \{1, \dots, n\}, \quad (8)$$

where π_θ denotes the parameterized scheduling policy, and the induced agent execution sequence is $\pi = [a_{\sigma_1}, \dots, a_{\sigma_T}]$.

3.2.3 Context Allocator Component

The Context Allocator is designed to enable context utilization by selecting an agent-specific memory subset and feeding it to the activated agent. For an active agent a_{σ_t} , the Context Allocator retrieves agent-specific memory subset as the context input to the active agent, formulated as:

$$o_t^{\text{con}} = \left[x, M_t^{\text{candidate}}, M_{t-1}^{\text{global}}, a_{\sigma_t} \right], \quad (9)$$

where $M_t^{\text{candidate}}$ is the candidate memory pool filtered by the Memory Manager based on the Memory Graph G_t (see in Eq. 3): starting from the memory node in step $t - 1$, it traverses backward along both temporal edges and semantic edges, collecting at each step the corresponding memory node as well as its 1-hop neighbors connected via these edges, until the size of the candidate memory set reaches the maximum capacity M (i.e., $|M_t^{\text{candidate}}| \leq M$).

To achieve more precise context allocation, the allocator performs refined matching on the candidate memory; specifically, it designs customized prompts to drive LLM in executing the matching task, thereby aligning the selected memory subset with the agent’s attributes and task requirements to form an agent-specific context, which is formally defined as:

$$M_t^{\text{context}} \sim \pi_\phi(\cdot | o_t^{\text{con}}), \quad (10)$$

where π_ϕ is the parameterized allocation policy and $M_t^{\text{context}} \subseteq M_{t-1}^{\text{candidate}}$ denotes the predefined context budget $K \leq M$ (i.e., the maximum number of memory units).

3.3 RL Policy Optimization

We employ an iterative, on-policy training framework to optimize both the Agent Scheduler and Context Allocator. The overall procedure is shown in Algorithm 1 in Appendix C.

3.3.1 Reward Function in RL

In MAS-OS framework, we construct a composite reward integrating three components: Active Agent Reward, Context Allocator reward, and Agent Scheduler reward. These reward functions simultaneously optimizes the task success, efficiency, and step-level allocation quality.

Active Agent Reward. Following RMM (Tan et al., 2025), we designed a method that enables active agents to provide reward signals. At step t , the Context Allocator selects a subset of memory units M_t^{context} to the active agent. After completing its work, for each memory item $m_i \in M_t^{\text{context}}$, the agent returns a binary value $u_i \in \{0, 1\}$ to indicate whether the memory units was useful. The hit rate reward of active agents is defined as:

$$R_{\text{agent}} = \frac{1}{|M_t^{\text{context}}|} \sum_{i=1} u_i. \quad (11)$$

Context Allocator Reward. Context Allocator is trained to maximize immediate allocation utility while remaining aligned with the final task objective. The reward function is defined as:

$$R_{\text{context}} = \alpha \cdot R_{\text{agent}} + \eta \cdot R_{\text{task}}, \quad (12)$$

where α controls the strength of hit-rate shaping and η couples allocation behavior to end-task success. R_{task} denotes the reward signal for evaluating final task completion performance, with its value falling within the interval $[0, 1]$.

Agent Scheduler Reward. Agent Scheduler is trained to enable effective collaboration trajectories and avoid unnecessary computation. We use a reward objective with a per-step cost penalty, defined as:

$$R_{\text{scheduler}} = R_{\text{task}} - \lambda \cdot c_t, \quad (13)$$

where c_t is the number of tokens consumed by the activated agent at step t , and λ trades off success against token consumption.

3.3.2 Training Policy

Following MARTI (Zhang et al., 2025b), the training process follows an alternating data collection and policy optimization paradigm. For each prompt, we generate N independent rollouts following MARTI (Zhang et al., 2025b). We employ the REINFORCE Leave-One-Out (RLOO) (Ahmadian et al., 2024) estimator to compute low-variance baselines by averaging returns across peer rollouts, avoiding the need for a separate value network. The advantage is estimated as the deviation of each sample’s return from the leave-one-out baseline. Policy updates are performed via Proximal Policy Optimization (PPO) (Schulman et al., 2017) with clipped surrogate objectives. A KL-divergence penalty (Ziegler et al., 2019; Ouyang et al., 2022b) is further incorporated to regularize the active policy toward the frozen reference model, preventing mode collapse and preserving LLM capabilities.

4 Experiments

4.1 Experimental Settings

4.1.1 Datasets

To ensure comprehensive evaluation in the MAS domain, we adopt four heterogeneous benchmarks spanning distinct task types: GSM-hard (Cobbe et al., 2021; Gao et al., 2023) and SciBench (Wang et al., 2023) for quantitative reasoning and college-level scientific problem-solving; SRDD (Qian et al., 2024) for software generation from natural-language specifications; and MMLU-Pro (Wang et al., 2024) for multidisciplinary knowledge reasoning via 10-way multiple-choice questions.

4.1.2 Evaluation Metrics

Following (Dang et al., 2026), we report the *Accuracy* on the GSM-hard, SciBench, and MMLU-Pro datasets, which reflects the model’s overall performance in complex mathematical reasoning, scientific problem solving, and multidisciplinary understanding. For SRDD, we follow the dataset’s original evaluation dimensions and report an overall quality score computed as a weighted average of: (i) *Completeness*, the fraction of generated projects that contain no placeholder code (e.g., TODO/PASS); (ii) *Executability*, the fraction of projects that can successfully compile and run in the target environment; and (iii) *Consistency*, the semantic alignment between the requirement description and the produced code, measured by the

cosine similarity between their embedding representations.

4.1.3 Implementation Details

We summarize experimental and implementation details in the Appendix: Appendix A covers dataset splits, baselines, and hyperparameters; Appendix B introduces specialized worker agents; Appendix C present the algorithm flowchart of MAS-OS; and Appendix (D-G) analyzes retrieval hyperparameters, generalization with different backbones (e.g., Qwen3), scheduler scaling, and budget-matched comparisons, respectively. The anonymous code repository is available at <https://github.com/BAI-LAB/MAS-OS>.

4.1.4 Compared Methods

We compare MAS-OS with representative baselines covering direct LLM inference, workflow optimization, and multi-agent collaboration in MAS literature.

- **Base Model:** The backbone LLM is used directly without agentic orchestration.
- **AFlow (Zhang et al., 2025a):** AFlow optimizes single-agent program-like reasoning workflows through search.
- **DyLAN (Liu et al., 2024):** DyLAN dynamically selects agents according to their reasoning contributions at each round, but still relies on full-history context sharing.
- **MacNet (Qian et al., 2025):** MacNet organizes agents with a predefined DAG and performs topological execution, which limits flexible revision.
- **EvoAgent (Yuan et al., 2025):** EvoAgent uses evolutionary algorithms to generate and optimize agent roles before inference.
- **Puppeteer (Dang et al., 2026):** Puppeteer trains a centralized RL-based orchestrator for dynamic agent selection, while largely relying on global interaction histories.
- **MAS-OS:** it jointly learns agent scheduling and context allocation with reinforcement learning, using a structured memory pool to provide compact task-relevant context.

Table 1: Main results on different benchmarks. We compare our method against baselines using both Llama-3.1-8B and Llama-3.1-70B as base models. **Bold** denotes the best result, and underline denotes the SOTA baseline results. Improvement is calculated based on the SOTA baseline. * indicates statistically significant improvements of our method over the SOTA baseline on the corresponding benchmark (paired two-sided t-test, $p < 0.05$).

Model	Method	GSM-hard	SciBench	SRDD	MMLU-Pro
Llama-3.1-8B	Base Model	0.0021	0.0920	0.4615	0.3900
	AFlow	<u>0.2763</u>	0.1653	0.6417	0.3270
	DyLan	0.2170	0.1156	0.6627	0.3040
	MacNet	0.1172	0.1940	0.6812	0.3640
	EvoAgent	0.2323	<u>0.2053</u>	<u>0.6940</u>	0.3170
	Puppeteer	0.1940	0.1766	0.6847	0.3540
	MAS-OS (-RL)	0.2890	0.2548	0.6954	0.3130
	MAS-OS	0.3150*	0.2782*	0.7131*	0.3760
	Improvement (%)	+14.01% ↑	+35.51% ↑	+2.75% ↑	-
Llama-3.1-70B	Base Model	0.1021	0.2687	0.8133	<u>0.5480</u>
	AFlow	<u>0.5179</u>	0.3376	0.8372	0.5050
	DyLan	0.4358	0.3265	0.8411	0.5320
	MacNet	0.2447	0.2847	<u>0.8524</u>	0.3520
	EvoAgent	0.3975	<u>0.3423</u>	0.8496	0.4380
	MAS-OS (-RL)	0.6313*	0.3757*	0.8667*	0.6040*
	Improvement (%)	+21.89% ↑	+9.76% ↑	+1.68% ↑	+10.22% ↑

4.2 Main Results

Table 1 presents the comparative analysis across four diverse benchmarks using both Llama-3.1-8B and Llama-3.1-70B as backbones (Grattafiori et al., 2024). MAS-OS (-RL) removes RL optimization from MAS-OS while retaining dynamic scheduling and selective context allocation. Our framework demonstrates consistent superiority over existing single-agent and multi-agent baselines. We have the following observations:

(1) Our method achieves the SOTA results on computation-intensive and logic-heavy tasks (i.e., GSM-hard, SciBench, and SRDD). On the Llama-3.1-8B backbone, we observe substantial margins compared to the strongest baselines, achieving significant improvements of 14.01% on GSM-hard and 35.51% on SciBench. Such gains underscore that multi-agent collaboration enabled by MAS-OS contributes to excellent performance in complex problem spaces.

(2) In contrast to -RL variants, MAS-OS exhibits a consistently positive performance trend across all datasets, attributed to the joint reinforcement training of agent allocation and scheduling modules. This collaborative training delivers an average improvement of 10.21% over the four benchmarks, demonstrating our method is capable of discovering more efficient interaction patterns that cannot be achieved by static prompting or manual engineering alone.

(3) MAS-OS maintains strong performance when scaled to the Llama-3.1-70B backbone. Owing to limitations in computational resources, we are not able to carry out RL training on the 70B model. However, MAS-OS(-RL) variant achieves superior results over all compared approaches across all tasks, outperforming the SOTA baseline method AFlow by 21.89% on GSM-hard, EvoAgent by 9.76% on SciBench, and Macnet by 1.68% on SRDD. This indicates that our framework is model-agnostic and does not rely on the idiosyncrasies of a specific base model; instead, it acts as a general-purpose reasoning amplifier that scales with the underlying model’s capabilities.

(4) We observe a unique performance pattern on MMLU-Pro that differs from other datasets. With Llama-3.1-8B, the base model slightly outperforms all agentic frameworks, a known issue where multi-agent orchestration introduces "over-reasoning" noise for fact-based queries. However, with Llama-3.1-70B, our method successfully reverses this trend, outperforming the base model (0.6040 vs. 0.5480). This indicates that as the backbone’s instruction-following capability improves, our dynamic routing mechanism can effectively suppress coordination noise and leverage external memory even for broad knowledge tasks.

4.3 Ablation Study

To evaluate the contribution of each component in MAS-OS, we conduct ablation studies by removing

Table 2: Ablation study of MAS-OS on Llama-3.1-8B.

Method	GSM-hard	SciBench	SRDD
-Context Allocator	0.233 $\downarrow$ 26.1%	0.184 $\downarrow$ 33.9%	0.659 $\downarrow$ 7.6%
-Memory Manager	0.251 $\downarrow$ 20.2%	0.204 $\downarrow$ 26.8%	0.676 $\downarrow$ 5.2%
-Agent Scheduler	0.282 $\downarrow$ 10.6%	0.262 $\downarrow$ 5.8%	0.688 $\downarrow$ 3.6%
MAS-OS	0.315	0.278	0.713

Table 3: Detailed comparison of computational efficiency. Average token consumption in **Execution** (Input) and **Inference** (Output) processes is quantified as the core metric for overhead evaluation. **Time** represents the average end-to-end inference latency.

Method	Execution	Inference	Total Tokens	Time (s)
MacNet	6,787	3,556	10,343	21.02
DyLan	38,778	9,602	48,381	46.41
Puppeteer	19,590	8,559	26,649	42.31
EvoAgent	25,306	8,632	33,938	70.82
MAS-OS	34,156	2,179	38,635	49.03

the Context Allocator, Agent Scheduler, and Memory Manager, respectively. **-Context Allocator:** the active agent receives only the previous-step input without selecting relevant memories from other agents. **-Agent Scheduler:** agents follow a fixed hand-designed workflow, whose order is provided in Appendix B. **-Memory Manager:** the Scheduler and Allocator operate directly on raw conversation histories without the structured MemoryPool. As shown in Table 2, all three removals lead to performance drops. The Context Allocator causes the largest degradation, highlighting the importance of precise context selection for multi-step reasoning. The Memory Manager ranks second, suggesting that structured memory helps retrieve and organize long-range information. The Agent Scheduler has a milder but still positive effect, indicating that adaptive scheduling complements memory organization and context allocation.

4.4 Efficiency Analysis

We further evaluate the efficiency of MAS-OS on GSM-hard with LLaMA-3.1-8B. We measure token consumption in two stages: the execution phase, which includes processed inputs and intermediate contexts, and the inference phase, which corresponds to generated outputs. As shown in Table 3, MAS-OS introduces higher execution-token consumption due to its Memory Manager, Agent Scheduler, and Context Allocator. However, it achieves the lowest inference-token consumption among all methods, indicating that selective

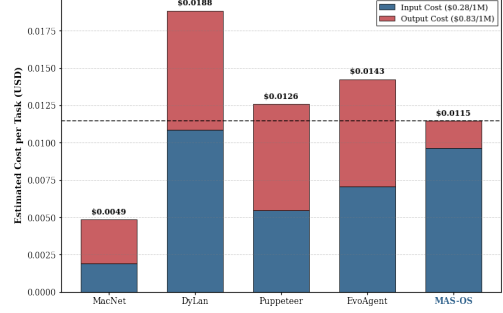


Figure 3: Estimated average cost per task on GSM-hard. The blue bars represent the cost of input tokens consumed by the framework, and the red bars represent the cost of output tokens.

memory routing effectively suppresses redundant generation. MAS-OS also obtains the best performance with competitive latency, reducing inference time by over 30% compared with EvoAgent.

4.5 Cost-benefit Analysis

The efficiency analysis suggests that reduced inference tokens benefit MAS-OS in practice. However, the additional execution overhead from memory routing raises the question of whether these gains translate into tangible economic advantages. Since production LLM APIs typically price output tokens substantially higher than input tokens, For example, under Cloudflare Workers AI pricing for Llama-3.1-8B, for instance, output tokens cost \$0.83/M compared with \$0.28/M for input tokens, making output-token reduction a key factor for cost efficiency. As illustrated in Fig. 3, MAS-OS achieves a strong accuracy-cost trade-off by using precise memory routing to provide active agents with task-relevant histories while reducing unnecessary generation, showing its advantage in lowering inference economic costs.

5 Conclusion

In this paper, we propose MAS-OS, a memory-centric operating system for LLM-based multi-agent systems. By unifying a Memory Manager, Agent Scheduler, and Context Allocator, it dynamically organizes shared memory, selects active agents, and allocates task-relevant context to achieve more adaptive and efficient multi-agent collaboration. Optimized via reinforcement learning, it achieves consistent improvements in reasoning and task-solving over strong baselines, highlighting the value of OS-level abstractions for scalable multi-agent collaboration.

6 Limitations

Despite its effectiveness, MAS-OS still has several limitations. First, its memory-centric collaboration mechanism introduces additional token overhead, since the Memory Manager, Agent Scheduler, and Context Allocator require extra intermediate prompts and decisions during execution. This may limit its applicability in latency- or cost-sensitive scenarios. Second, the current framework mainly follows a sequential agent activation process, which may underutilize potential parallelism among independent subtasks. Finally, our experiments are conducted with predefined agent roles and benchmark-specific reward signals, leaving the extension to more heterogeneous agents and broader real-world tasks as future work.

Acknowledgments

This work was supported in part by the National Natural Science Foundation of China (Grant No. 62236003).

References

- Arash Ahmadian, Chris Cremer, Matthias Gallé, Marzieh Fadaee, Julia Kreutzer, Olivier Pietquin, Ahmet Üstün, and Sara Hooker. 2024. Back to basics: Revisiting reinforce-style optimization for learning from human feedback in llms. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 12248–12267.
- Jianlyu Chen, Shitao Xiao, Peitian Zhang, Kun Luo, Defu Lian, and Zheng Liu. 2024. [M3-embedding: Multi-linguality, multi-functionality, multi-granularity text embeddings through self-knowledge distillation](#). In *Findings of the Association for Computational Linguistics: ACL 2024*, pages 2318–2335, Bangkok, Thailand. Association for Computational Linguistics.
- Weize Chen, Jiarui Yuan, Chen Qian, Cheng Yang, Zhiyuan Liu, and Maosong Sun. 2025. Optima: Optimizing effectiveness and efficiency for llm-based multi-agent system. In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 11534–11557.
- Prateek Chhikara, Dev Khant, Saket Aryan, Taranjeet Singh, and Deshraj Yadav. 2025. Mem0: Building production-ready ai agents with scalable long-term memory. *arXiv preprint arXiv:2504.19413*.
- Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, and 1 others. 2021. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*.
- Yufan Dang, Chen Qian, Xueheng Luo, Jingru Fan, Zihao Xie, Ruijie Shi, Weize Chen, Cheng Yang, Xiaoyin Che, Ye Tian, and 1 others. 2026. Multi-agent collaboration via evolving orchestration. *Advances in neural information processing systems*, 38:165025–165059.
- Jizhan Fang, Xinle Deng, Haoming Xu, Ziyang Jiang, Yuqi Tang, Ziwen Xu, Shumin Deng, Yunzhi Yao, Mengru Wang, Shuofei Qiao, and 1 others. 2025. Lightmem: Lightweight and efficient memory-augmented generation. *arXiv preprint arXiv:2510.18866*.
- Luyu Gao, Aman Madaan, Shuyan Zhou, Uri Alon, Pengfei Liu, Yiming Yang, Jamie Callan, and Graham Neubig. 2023. Pal: Program-aided language models. In *International Conference on Machine Learning*, pages 10764–10799. PMLR.
- Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, and 1 others. 2024. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*.
- Sirui Hong, Mingchen Zhuge, Jonathan Chen, Xiawu Zheng, Yuheng Cheng, Jinlin Wang, Ceyao Zhang, Steven Yau, Zijuan Lin, Liyang Zhou, and 1 others. 2024. Metagpt: Meta programming for a multi-agent collaborative framework. In *International Conference on Learning Representations*, volume 2024, pages 23247–23275.
- Jiazheng Kang, Mingming Ji, Zhe Zhao, and Ting Bai. 2025. Memory os of ai agent. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 25972–25981.
- Hengzhi Lan, Yue Yu, Li Qian, Li Peng, Jie Wu, Wei Liu, Jian Luan, and Ting Bai. 2025. Lightsearcher: Efficient deepsearch via experiential memory. *arXiv preprint arXiv:2512.06653*.
- Guohao Li, Hasan Hammoud, Hani Itani, Dmitrii Khizbullin, and Bernard Ghanem. 2023. Camel: Communicative agents for "mind" exploration of large language model society. *Advances in neural information processing systems*, 36:51991–52008.
- Shaobin Ling, Yun Wang, Chenyou Fan, Tin Lun Lam, and Junjie Hu. 2025. Elhplan: Efficient long-horizon task planning for multi-agent collaboration. *arXiv preprint arXiv:2509.24230*.
- Jun Liu, Zhenglun Kong, Changdi Yang, Fan Yang, Tianqi Li, Peiyan Dong, Joannah Nanjeyye, Hao Tang, Geng Yuan, Wei Niu, and 1 others. 2025. Rcrouter: Efficient role-aware context routing for multi-agent llm systems with structured memory. *arXiv preprint arXiv:2508.04903*.

- Zijun Liu, Yanzhe Zhang, Peng Li, Yang Liu, and Diyi Yang. 2024. A dynamic llm-powered agent network for task-oriented agent collaboration. In *First Conference on Language Modeling*.
- Reiichiro Nakano, Jacob Hilton, Suchir Balaji, Jeff Wu, Long Ouyang, Christina Kim, Christopher Hesse, Shantanu Jain, Vineet Kosaraju, William Saunders, and 1 others. 2021. Webgpt: Browser-assisted question-answering with human feedback. *arXiv preprint arXiv:2112.09332*.
- Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, and 1 others. 2022a. Training language models to follow instructions with human feedback. *Advances in neural information processing systems*, 35:27730–27744.
- Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, and 1 others. 2022b. Training language models to follow instructions with human feedback. *Advances in neural information processing systems*, 35:27730–27744.
- Charles Packer, Vivian Fang, Shishir_G Patil, Kevin Lin, Sarah Wooders, and Joseph_E Gonzalez. 2023. Memgpt: Towards llms as operating systems.
- Joon Sung Park, Joseph O’Brien, Carrie Jun Cai, Meredith Ringel Morris, Percy Liang, and Michael S Bernstein. 2023. Generative agents: Interactive simulacra of human behavior. In *Proceedings of the 36th annual acm symposium on user interface software and technology*, pages 1–22.
- Chen Qian, Wei Liu, Hongzhang Liu, Nuo Chen, Yufan Dang, Jiahao Li, Cheng Yang, Weize Chen, Yusheng Su, Xin Cong, and 1 others. 2024. Chatdev: Communicative agents for software development. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 15174–15186.
- Chen Qian, Zihao Xie, Yifei Wang, Wei Liu, Kunlun Zhu, Hanchen Xia, Yufan Dang, Zhuoyun Du, Weize Chen, Cheng Yang, and 1 others. 2025. Scaling large language model-based multi-agent collaboration. In *International Conference on Learning Representations*, volume 2025, pages 41488–41505.
- John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*.
- Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. 2023. Reflexion: Language agents with verbal reinforcement learning. *Advances in neural information processing systems*, 36:8634–8652.
- Zhen Tan, Jun Yan, I-Hung Hsu, Rujun Han, Zifeng Wang, Long Le, Yiwen Song, Yanfei Chen, Hamid Palangi, George Lee, and 1 others. 2025. In prospect and retrospect: Reflective memory management for long-term personalized dialogue agents. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 8416–8439.
- Song Wang, Zhen Tan, Zihan Chen, Shuang Zhou, Tianlong Chen, and Jundong Li. 2025a. Anymac: Cascading flexible multi-agent collaboration via next-agent prediction. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 11566–11578.
- Xiaoxuan Wang, Ziniu Hu, Pan Lu, Yanqiao Zhu, Jieyu Zhang, Satyen Subramaniam, Arjun R Loomba, Shichang Zhang, Yizhou Sun, and Wei Wang. 2023. Scibench: Evaluating college-level scientific problem-solving abilities of large language models. *arXiv preprint arXiv:2307.10635*.
- Yubo Wang, Xueguang Ma, Ge Zhang, Yuansheng Ni, Abhranil Chandra, Shiguang Guo, Weiming Ren, Aaran Arulraj, Xuan He, Ziyang Jiang, and 1 others. 2024. Mmlu-pro: A more robust and challenging multi-task language understanding benchmark. *Advances in Neural Information Processing Systems*, 37:95266–95290.
- Zhexuan Wang, Yutong Wang, Xuebo Liu, Liang Ding, Miao Zhang, Jie Liu, and Min Zhang. 2025b. Agentdropout: Dynamic agent elimination for token-efficient and high-performance llm-based multi-agent collaboration. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 24013–24035.
- Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, and 1 others. 2024. Autogen: Enabling next-gen llm applications via multi-agent conversations. In *First Conference on Language Modeling*.
- Haoran Xu, Jiacong Hu, Ke Zhang, Lei Yu, Yuxin Tang, Xinyuan Song, Yiqun Duan, Lynn Ai, and Bill Shi. 2025. Sedm: Scalable self-evolving distributed memory for agents. *arXiv preprint arXiv:2509.09498*.
- Wujiang Xu, Zujie Liang, Kai Mei, Hang Gao, Juntao Tan, and Yongfeng Zhang. 2026. A-mem: Agentic memory for llm agents. *Advances in Neural Information Processing Systems*, 38:17577–17604.
- Xiangyuan Xue, Yifan Zhou, Guibin Zhang, Zaibin Zhang, Yijiang Li, Chen Zhang, Zhenfei Yin, Philip Torr, Wanli Ouyang, and Lei Bai. 2025. Comas: Co-evolving multi-agent systems via interaction rewards. *arXiv preprint arXiv:2510.08529*.
- Siyu Yuan, Kaitao Song, Jiangjie Chen, Xu Tan, Dongsheng Li, and Deqing Yang. 2025. 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), pages 6192–6217.

- Yanwei Yue, Guibin Zhang, Boyang Liu, Guancheng Wan, Kun Wang, Dawei Cheng, and Yiyang Qi. 2025. Masrouter: Learning to route llms for multi-agent systems. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 15549–15572.
- Aohan Zeng, Mingdao Liu, Rui Lu, Bowen Wang, Xiao Liu, Yuxiao Dong, and Jie Tang. 2024. Agenttuning: Enabling generalized agent abilities for llms. In *Findings of the Association for Computational Linguistics: ACL 2024*, pages 3053–3077.
- Sihang Zeng, Kai Tian, Kaiyan Zhang, Yuru Wang, Junqi Gao, Runze Liu, Sa Yang, Jingxuan Li, Xinwei Long, Jiaheng Ma, and 1 others. 2025. Reviewrl: Towards automated scientific review with rl. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 16942–16954.
- Guibin Zhang, Muxin Fu, Kun Wang, Frank Wan, Miao Yu, and Shuicheng Yan. 2026. G-memory: Tracing hierarchical memory for multi-agent systems. *Advances in Neural Information Processing Systems*, 38:12988–13018.
- Jiayi Zhang, Jinyu Xiang, Zhaoyang Yu, Fengwei Teng, Xionghui Chen, Jiaqi Chen, Mingchen Zhuge, Xin Cheng, Sirui Hong, Jinlin Wang, and 1 others. 2025a. Aflow: Automating agentic workflow generation. In *International Conference on Learning Representations*, volume 2025, pages 34040–34077.
- Kaiyan Zhang, Kai Tian, Runze Liu, Sihang Zeng, Xuekai Zhu, Guoli Jia, Yuchen Fan, Xingtai Lv, Yuxin Zuo, Che Jiang, and 1 others. 2025b. Marti: A framework for multi-agent llm systems reinforced training and inference. In *The Fourteenth International Conference on Learning Representations*.
- Heng Zhou, Hejia Geng, Xiangyuan Xue, Li Kang, Yiran Qin, Zhiyong Wang, Zhenfei Yin, and Lei Bai. 2025. Reso: A reward-driven self-organizing llm-based multi-agent system for reasoning tasks. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 15990–16009.
- Daniel M Ziegler, Nisan Stiennon, Jeffrey Wu, Tom B Brown, Alec Radford, Dario Amodei, Paul Christiano, and Geoffrey Irving. 2019. Fine-tuning language models from human preferences. *arXiv preprint arXiv:1909.08593*.

A Implementation Details

For GSM-hard, SRDD, and MMLU-Pro, we follow Puppeteer (Dang et al., 2026) and adopt their released train/test splits. For SciBench, since the original release provides only a single split, we randomly sample 30% as the training set and use the remaining as the test set. Among the compared methods, only MAS-OS and Puppeteer involve learnable parameter updates; on each benchmark they share the same training set. The remaining baselines (AFlow, DyLAN, MacNet, EvoAgent) are training-free. All methods are evaluated on the identical test set per benchmark to ensure fair comparison. To assess the scalability and robustness of our framework, we employ two backbone models: Llama-3.1-8B and Llama-3.1-70B. For a fair comparison, all worker agents across our method and baselines share identical role definitions (See details in the appendix B). Specific adjustments were made for the Puppeteer baseline: for the Llama-3.1-8B experiments, we replaced its original 70B scheduling policy with Llama-3.1-8B to maintain consistency. Due to computational resource constraints, we exclude Puppeteer from the Llama-3.1-70B evaluation, as its policy optimization necessitates access to model weights. We use BGE-M3 (Chen et al., 2024) as the semantic embedding model and set $\tau = 0.80$. For each prompt during training, we sample $N = 16$ rollouts and compute advantages using the RLOO leave-one-out baseline. Policy optimization uses a PPO-style clipped surrogate objective with clipping range $\epsilon = 0.2$. We regularize the policy toward a frozen reference model using an explicit KL-style penalty with coefficient $\beta = 0.001$, and set the actor learning rate to 1×10^{-5} . We use a rollout batch size of 12 with micro-rollout batch size 4, and a training batch size of 8 with micro-train batch size 2 and gradient accumulation steps 4. Rewards are clipped to $[-2, 2]$. In the testing and training phases of MAS, set $K=3$ and $M=7$. The Agent Scheduler reward uses weight $\lambda = 5 \times 10^{-5}$, while the Context Allocator reward uses $\alpha = 0.25$ and $\eta = 1$. The terminal task reward R^{task} varies by benchmark to align with their respective evaluation criteria. For the reasoning benchmarks (GSM-hard, SciBench, and MMLU-Pro), R^{task} is defined as a binary indicator of correctness, while for the SRDD software generation benchmark, R^{task} is computed as the arithmetic mean of the three component scores (Qian et al., 2024). **More de-**

tails of implementation are available at <https://anonymous.4open.science/r/MAS-OS-AD34>.

B Agent Description

Following the experimental setup of Puppeteer (Dang et al., 2026), we employ a fixed cohort of specialized agents that strictly adhere to pre-defined configuration settings across four well-established benchmark datasets. The specific functional roles and designated task responsibilities of the active agent instances within this multi-agent framework are elaborated as follows:

- **PlannerAgent:** decomposes a task into an explicit step-by-step plan.
- **QuestionAgent:** breaks the main problem into a small set of key sub-questions to guide subsequent reasoning.
- **ReasoningAgent:** performs the main multi-step reasoning and produces intermediate conclusions.
- **CriticAgent:** identifies errors, gaps, or weak points in the current reasoning and suggests corrections.
- **ReflectAgent:** reviews the reasoning trajectory, summarizes what worked and what did not, and proposes next steps.
- **PythonAgent:** executes Python calculations for verification and outputs computed results.
- **SummarizerAgent:** compresses the current context into a concise summary for downstream steps.
- **Modifier:** corrects specific mistakes and provides a revised reasoning or solution.
- **ConcluderAgent:** synthesizes the final answer from previous steps in a fixed final-answer format.
- **TerminatorAgent:** decides when the process should terminate and outputs the final response.

C Implementation of MAS-OS Algorithm

The algorithmic flowchart of MAS-OS is shown in Alg 1.

Algorithm 1 MAS-OS Execution Loop

Input: Task set $\mathcal{D}$; Agent pool A ; Max steps T ; Candidate memory pool size M ; Allocated context size K ; Rollouts N

Output: optimized policy parameters π_θ, π_ϕ

- 1: Initialize scheduler policy π_θ and allocator policy π_ϕ
- 2: **while** not converged **do**
- 3: **for all** $(x, y^*) \in \mathcal{D}$ **do**
- 4: **for** $i \leftarrow 1$ **to** N **do**
- 5: **for** $t \leftarrow 1$ **to** T **do**
- 6: $o_t^{\text{sch}} \leftarrow [x, M_{t-1}^{\text{global}}, s_{t-1}, a_{\sigma_{t-1}}, A]$
- 7: $\sigma_t \sim \pi_\theta(\cdot \mid o_t^{\text{sch}})$
- 8: **if** $\sigma_t = \text{Done}$ **then**
- 9: **break**
- 10: **end if**
- 11: $M_t^{\text{Pool}} \leftarrow \{G_{t-1}, M_{t-1}^{\text{global}}\}$
- 12: $M_t^{\text{candidate}} \leftarrow \text{BUILDCANDIDATES}(M_t^{\text{Pool}}, x, M)$
- 13: $o_t^{\text{con}} \leftarrow [x, M_t^{\text{candidate}}, M_{t-1}^{\text{global}}, a_{\sigma_t}]$
- 14: $M_t^{\text{context}} \sim \pi_\phi(\cdot \mid o_t^{\text{con}})$
- 15: $\hat{y}_t \leftarrow \text{EXECUTEAGENT}(a_{\sigma_t}, x, M_{t-1}^{\text{global}}, M_t^{\text{context}})$
- 16: $(o_t, s_t) \leftarrow \text{SEPSUM}(\hat{y}_t)$
- 17: $G_t \leftarrow \text{MEMGRAPHUPDATE}(G_{t-1}, o_t, s_t)$
- 18: $M_t^{\text{global}} \leftarrow \text{UPDATEGLOBAL}(M_{t-1}^{\text{global}}, s_t)$
- 19: **end for**
- 20: Compute $R_{\text{task}}, \{R_{\text{context}}^{(t)}\}, \{R_{\text{sch}}^{(t)}\}$
- 21: Store rollout τ_i
- 22: **end for**
- 23: Estimate advantages via RLOO
- 24: Update π_θ and π_ϕ using PPO
- 25: **end for**
- 26: **end while**
- 27: **return** π_θ, π_ϕ

D Results on Qwen Backbone

To verify backbone generalization, we evaluate MAS-OS and baselines with Qwen3-8B under the same setup as the main experiments. As shown in Table 4, MAS-OS achieves the best results on all

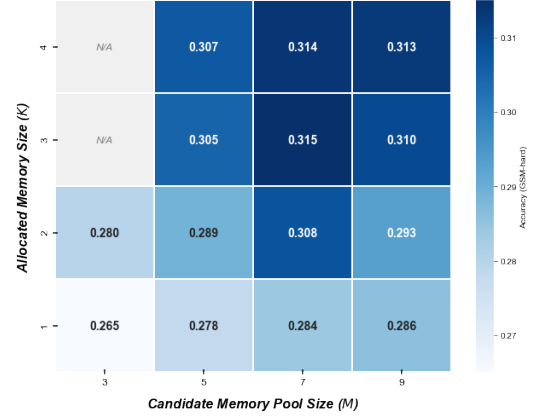


Figure 4: **Impact of retrieval hyperparameters (Candidate Memory Pool Size M vs. Allocated Memory Size K).**

four benchmarks, consistent with the Llama-3.1-8B trend.

Table 4: Results on Qwen3-8B across four benchmarks.

Method	GSM-hard	SciBench	MMLU-Pro	SRDD
Base Model	0.148	0.167	0.528	0.563
MacNet	0.171	0.214	0.549	0.624
Puppeteer	0.305	0.275	0.551	0.638
DyLAN	0.378	0.268	0.543	0.629
EvoAgent	0.452	0.362	0.514	0.648
MAS-OS	0.483	0.384	0.576	0.719

E Hyperparameter Analysis

We investigate how the candidate memory pool size M and the allocated context size K (defined in Sec. 3.2.3) jointly affect performance, with $K \in \{1, 2, 3, 4\}$ and $M \in \{3, 5, 7, 9\}$ on GSM-hard with Llama-3.1-8B (Figure 4). Increasing M from 3 to 7 steadily improves performance by enlarging the recall scope, but further expanding to 9 yields diminishing returns. Similarly, raising K from 1 to 3 supplies a richer information chain, while exceeding 3 plateaus or slightly hurts accuracy as redundant context introduces noise. We therefore adopt $M = 7, K = 3$ to balance information sufficiency and context conciseness.

F Impact of Scheduler Capability

MAS-OS enables dynamic agent selection and execution based on multi-agent context via its Agent Scheduler. To verify the impact of the scheduler’s capability on MAS-OS performance, we conduct scalability experiments by scaling the Scheduler’s backbone across three different model sizes (Llama-3.2-3B, Llama-3.1-8B, and Llama-3.1-70B) while

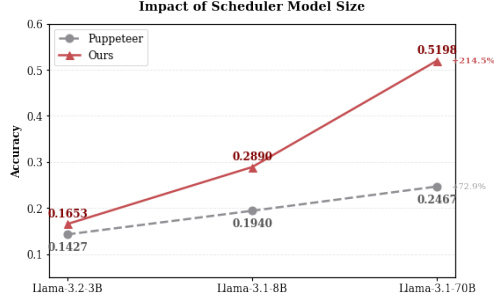


Figure 5: Impact of Agent Scheduler capability on model performance on GSM-Hard verified via Scheduler scaling from 3B to 7B and 70B.

Table 5: Budget-matched comparison. SC@K and Refl@K denote Self-Consistency and Reflection with K sampling rounds calibrated to match MAS-OS’s cost.

Method	GSM-hard	SciBench
SC@28 (8B)	0.175	0.158
SC@8 (70B)	0.270	0.245
Refl@8 (8B)	0.252	0.225
MAS-OS (8B)	0.315	0.278

keeping active agents fixed at 8B. The experimental results comparison with those of Puppeteer under the same Scheduler scaling settings are shown in Figure 5. We can see that enhancing the capability of Agent Scheduler yields performance gains for both frameworks. However, our method displays significantly higher responsiveness (+214.5%) compared to Puppeteer (+72.9%) on Llama-3.1-70B. This indicates that our intricate Agent Scheduler-Context Allocator logic involves complex decision-making (e.g., dynamic workflow planning) that is more effectively executed by larger models, allowing our framework to leverage the superior reasoning capabilities of the 70B backbone more fully than the baseline.

G Budget-Matched Comparison

To verify that MAS-OS’s gains come from its collaborative architecture rather than additional inference compute, we compare it against budget-matched baselines under identical dollar cost: Self-Consistency (SC) with 8B and 70B backbones, and Reflection with 8B. The sampling budgets ($K = 28, 8, 8$ respectively) are calibrated so that each baseline incurs the same cost as a single MAS-OS rollout. As shown in Table 5, MAS-OS (8B) outperforms all budget-matched baselines, including SC with the much stronger 70B backbone,

confirming that the improvements stem from the memory-centric collaboration mechanism rather than scaling inference compute.