ALMC: ADAPTIVE LLM-BASED MULTI-AGENT COLLABORATION ACROSS DIVERSE TASK DOMAINS

Anonymous authors
Paper under double-blind review

ABSTRACT

Large language model-based multi-agent systems (LLM-MAS) are effective at solving complex tasks by coordinating specialized agents. However, existing frameworks rely on a small set of predefined scenarios with static role configurations and rigid collaboration structures, limiting their adaptability across diverse task domains. We propose the Adaptive LLM-MAS Collaboration (ALMC) framework, which dynamically recruits agents and configures collaboration patterns according to task demands through three collaborative components: a Manager Agent that synthesizes task-specific role compositions and an executable workflow, a Judge Agent that evaluates execution quality, and a Solution Optimizer Agent that persists and reuses high-quality configurations via retrieval-augmented generation. The framework supports human-in-the-loop review and creates a learning loop where previous superior configurations improve future executions on similar tasks. By using ALMC, collaborations become adaptive, auditable, and reusable across domains. Code is available at: `https://anonymous.4open.science/r/ALMC-2E0F`.

1 INTRODUCTION

Large language models (LLMs) have demonstrated strong general-purpose capabilities in reasoning, coding, and extensive knowledge, enabling agentic systems that plan, act, and reflect under minimal supervision (Yao et al., 2023). To go beyond the capacity of a single agent, LLM-based multi-agent systems (LLM-MAS) coordinate multiple specialized agents via role specialization and collaboration mechanisms, and have shown promising results in software engineering, web automation, scientific assistance, and healthcare (Hong et al., 2023; Zhang et al., 2025a; D'Arcy et al., 2024; Tang et al., 2024).

Currently, some LLM-MAS frameworks have been proposed and can be divided into two categories: general-purpose and domain-specific.

General-purpose frameworks, such as debate and voting systems, do not employ domain-specific role specialization and instead rely on collaboration mechanisms where homogeneous agents address problems through discussion to reach consensus (Du et al., 2023a; Wang et al., 2022). This approach enables broad applicability across diverse tasks, but often suffers from convergence issues when multiple agents produce conflicting proposals.

Domain-specific frameworks achieve superior task-solving quality within specialized domains through carefully designed role specializations and collaboration mechanisms. For example, MetaGPT (Hong et al., 2023) in software engineering hand-crafts heterogeneous agents, such as project managers and engineers, each assigned specific tasks. They collaborate through a publish-subscribe mechanism in a shared workspace, forming an agent chain where each executes based on the output of the previous one. Similarly, MedAgents (Tang et al., 2024) in the medical domain employs a pool of expert agents. Their collaboration mechanism selects a subset of experts to negotiate a consensus on a given problem. However, these systems require significant manual effort to configure agent roles, and hard-coded collaboration logic is difficult to modify. When executing cross-domain tasks, this leads to suboptimal performance due to mismatched role configurations and costly collaboration mechanism modifications (Liu et al., 2023).

Figure 1 shows the setting of the framework.

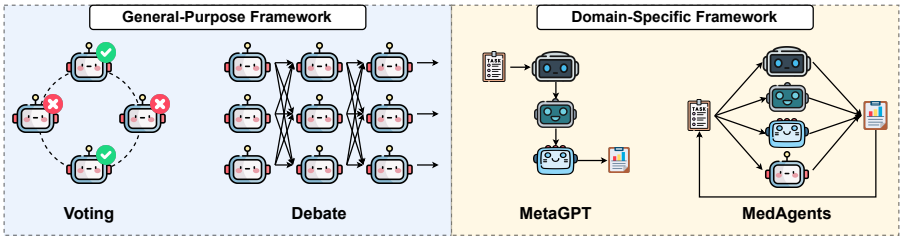


Figure 1: Comparison of general-purpose and domain-specific LLM-MAS frameworks.

Despite recent advances, building a usable, transferable, and stable collaborative multi-agent framework remains a significant challenge. We highlight three key challenges: **Challenge 1**: Trade-off between generality and specialization. Current systems struggle to balance high adaptability to different domains and high task-solving ability within their specialized domains. General-purpose frameworks demonstrate high adaptability to different domains but underperform on complex, domain-specific tasks because they lack specialized prompts, while domain-specific frameworks achieve superior task-solving ability but require extensive manual reconfiguration when applied to new areas (Qian et al., 2023; Tang et al., 2024; Kim et al., 2024; Zhang et al., 2025b). **Challenge 2**: Ineffective collaboration patterns. General-purpose frameworks rely on multiple homogeneous agents debating or voting toward consensus. However, this collaboration often results in conflicting opinions that lead to a stalemate in the discussion (Wang et al., 2022; Kim et al., 2024). Meanwhile, domain-specific frameworks adopt rigid collaboration mechanisms where agents work independently without agents' negotiation, resulting in error propagation and missed opportunities for global optimization. **Challenge 3**: Lack of experience accumulation and reuse. Most systems lack systematic mechanisms for learning and reusing successful configurations. They rely on static, hard-coded collaboration patterns that cannot be easily modified or improved based on past performance. This inflexibility represents a limitation, as even within the same domain, identical configurations can produce varying results on different tasks. As a result, systems demonstrate unstable performance within their domains (Liu et al., 2023; Zhang et al., 2025b; Zheng et al., 2023).

To address these challenges, we propose the Adaptive LLM-MAS Collaboration (ALMC) framework. Adaptive LLM-MAS Collaboration framework organizes three complementary roles (the Manager Agent, Solution Optimizer Agent and Judge Agent). For (1) balancing general-purpose usability with domain specificity and cross-domain transfer, the framework does not rely on a preset domain library. Instead, a Manager Agent collaborates with a Solution Optimizer Agent to synthesize task-specific role compositions, execution phases, and an executable workflow directly from the current user instruction and retrieved historical configurations. For (2) convergence under collaboration and process governance, the Manager Agent decomposes the task into complementary sub-phases and dynamically designs heterogeneous roles, phases, and an executable workflow. For (3) reuse and stability, Adaptive LLM-MAS Collaboration framework integrates a Judge Agent and the Solution Optimizer with a RAG memory. The Judge Agent generates structured assessments and quality scores for intermediate artifacts and final results (Shi et al., 2024), while the optimizer persists high-quality role-phase-workflow configurations and their evaluations for retrieval and reuse in similar tasks, enabling high-performance solution generation.

In summary, our contributions are threefold:

- We introduce ALMC, an adaptive LLM-based multi-agent framework where a Manager Agent automatically synthesizes task-specific roles, phases, and workflow, reducing reliance on handcrafted prompts and rigid collaboration patterns.
- We develop a Judge Agent-Solution Optimizer Agent module that assesses, persists, and retrieves high-quality team compositions and execution workflows, enabling systematic reuse, enhanced auditability, and stable performance on similar tasks.
- Empirical studies demonstrate that ALMC improves accuracy and stability while maintaining efficiency over strong general-purpose and domain-specific baselines.

2 Related Work

LLM-based multi-agent systems (LLM-MAS) are composed of multiple LLM-driven agents to tackle tasks that exceed the capability of a single agent by leveraging role specialization and collaboration mechanisms He et al. (2024). Role specialization is typically realized via prompting or fine-tuning to create complementary roles and skills. Collaboration mechanisms involve interaction patterns that simulate team cooperation in real-world settings through collaborative planning, discussion, and decision-making. These collaboration patterns require careful design to fully realize the benefits of team. This potential has motivated researchers to explore various framework paradigms, which can be categorized into two main approaches: general-purpose frameworks that emphasize broad applicability, and domain-specific frameworks that prioritize in-domain performance.

2.1 General-Purpose LLM-MAS Frameworks

General-purpose frameworks aim to provide flexible, transferable approaches that can adapt to diverse tasks without domain-specific customization. For example, debate and voting frameworks leverage multi-agent discussion to improve factuality and reasoning through structured argumentation or voting ensembles (Du et al., 2023a; Wang et al., 2022). To enhance reliability, follow-up systems have further structured their critique and judgment, such as FORD (Xiong et al., 2023) for inter-consistency analysis and ChatEval for multi-agent evaluation protocols (Chan et al., 2023). Beyond these, another class of general-purpose frameworks focuses on task-driven cooperation, particularly in embodied and robotic settings. These systems assign global objectives and enable agents to coordinate through natural language rather than strict pre-defined roles, where flexible communication serves as the primary collaboration mechanism for multi-robot planning and cooperation (Chen et al., 2024; Zhao et al., 2023; Zhang et al., 2023).

However, these general-purpose frameworks suffer from significant limitations. They often assign multiple homogeneous agents to identical goals, which can result in duplicate or conflicting proposals that lead to ineffective convergence. When discussing complex solutions such as code design or system architecture, consensus becomes difficult to reach due to subtle differences that prevent agreement (Kim et al., 2024). Additionally, the lack of explicit phase contracts and progress monitoring can result in circular discussions and inconsistent performance, particularly when applied to specialized domains where domain expertise is crucial.

2.2 Domain-Specific LLM-MAS Frameworks

To address the performance limitations of general-purpose methods, researchers have developed domain-specific multi-agent systems that achieve high in-domain reliability through carefully designed role specialization and collaboration mechanisms. In software engineering, ChatDev (Qian et al., 2023) coordinates requirements, coding, testing, and review agents with communicative dehallucination mechanisms; MetaGPT (Hong et al., 2023) instantiates product manager, architect, and engineer roles with standardized documents connecting planning and implementation. In healthcare, MedAgents (Tang et al., 2024) designs five-stage medical pipelines involving expert gathering, analysis, report summarization, collaborative consultation, and decision-making, achieving consistent zero-shot gains on medical Q&A tasks.

Despite their superior in-domain performance, domain-specific systems face critical limitations in terms of adaptability and engineering overhead. These systems assume static role configurations and fixed workflows, severely limiting adaptation and reuse across different domains. When transferring to new vertical domains, they require extensive manual redesign of prompts, agent roles, and workflows, significantly increasing engineering costs and slowing iteration cycles (Zhang et al., 2025b). Moreover, many systems treat each execution as an isolated event, relying on fixed code-based configurations with poor readability and modifiability, preventing systematic learning from successful executions (Liu et al., 2023; Zhang et al., 2025b).

Recent adaptive approaches like MDAgents (Kim et al., 2024) and DyLAN (Liu et al., 2023) attempt to bridge this gap through mode-switching and dynamic agent selection based on task complexity. However, these systems typically operate over fixed agent pools or preset workflows, limiting full automation and cross-domain reuse.

This analysis reveals that existing frameworks are constrained by static designs that force a choice between generality and specialization, motivating the need for adaptive approaches that can dynamically configure team compositions and collaboration patterns based on task requirements.

3 METHODOLOGY

Problem Setup. Given a task Q in natural language format, ALMC aims to produce an executable solution and a final deliverable ans. It first synthesizes a configuration $\mathcal{C} = (R, \mathcal{P}, \mathcal{G})$, where R denotes the roles in *On-Demand Agents Team* that are assembled to address the current task; $\mathcal{P}$ denotes the task phases obtained by decomposing Q (each phase selects the most suitable role pair from R for its intent); and $\mathcal{G}$ is a workflow graph that orders phases and encodes their dependencies. A pre-execution human-in-the-loop (HITL) gate allows users to review or edit $\mathcal{C}$; upon approval, ALMC freezes it as $\mathcal{C}^*$ and executes deterministically along $\mathcal{G}$. Execution yields intermediate artifacts O (structured placeholders passed between phases), logs $\mathcal{L}$, from which a final report ans is then aggregated; an assessment a is generated to evaluate quality and coherence. ALMC persists $(Q, \mathcal{C}^*, \mathcal{L}, a)$ in a RAG-backed task memory $\mathcal{M}$ to enable retrieval-based reuse and offline redesign on future, similar tasks. We summarize the end-to-end pipeline in Algorithm 1 and illustrate the complete framework in Figure 2.

Algorithm 1 Adaptive LLM-MAS Collaboration (ALMC) with Solution Optimizer

Require: Task Q;
Output: Final report ans, Assessment a

Step I: Design
1: $T_{\text{meta}} \leftarrow$ Manager.Analyze(Q) ▷ task analysis: scope, constraints, domain cues
2: Priors $\leftarrow$ SO.Retrieve($\mathcal{M}$, Q, top-k) ▷ retrieve prior high-quality solutions
3: $\text{prop}_0 \leftarrow$ Manager.InitProposal(Q, T_{meta}, Priors) ▷ initial proposal
4: **for** $j \leftarrow 1$ **to** $J_{\max}$ **do** ▷ few-round design negotiation
5: $\text{crit}_j \leftarrow$ SO.Critique(prop_{j-1}, Priors, M) ▷ SO reviews the proposal with Manager
6: $\text{prop}_j \leftarrow$ Manager.Revise(prop_{j-1}, crit_j) ▷ revise roles/phases/workflow per critique
7: **end for**
8: $(R, \mathcal{P}, \mathcal{G}) \leftarrow$ ExtractConfig($\text{prop}_{J_{\max}}$)
9: $\mathcal{C} \leftarrow (R, \mathcal{P}, \mathcal{G})$
10: $\mathcal{C}^* \leftarrow$ HITL.ReviewEdit($\mathcal{C}$) ▷ edit;approve→freeze $\mathcal{C}^*$; reject→ back to step 1
Step II: Execute ▷ init artifacts, logs, and previous-phase output
11: **for** each $Phase_i \in$ Order($\mathcal{G}$) **do**
12: $(r_{u_i}, r_{v_i}) \leftarrow$ SpecRoles($Phase_i$) ▷ specify active role pair for this phase
13: $T_i \leftarrow \tau_{\mathcal{G}}(Phase_i)$ ▷ get max turn budget for this phase
14: $S \leftarrow$ InitState(Q, $Phase_i$, O) ▷ build phase-local context
15: **for** $t \leftarrow 1$ **to** T_i **do**
16: $(m_u, \ell_u) \leftarrow$ Step(r_{u_i}, S) ▷ r_{u_i} generates message m_u; per-turn logs ℓ_u
17: $(m_v, \ell_v) \leftarrow$ Reply(r_{v_i}, m_u, S) ▷ r_{v_i} replies based on m_u; per-turn logs ℓ_v
18: $S \leftarrow$ Update(S, m_u, m_v)
19: $L_i \leftarrow L_i \cup \{\ell_u, \ell_v\}$ ▷ aggregate stats logs (execution progress, latency, tokens, etc.)
20: **end for**
21: $o_i \leftarrow$ Summarize(S) ▷ produce structured placeholder for handoff
22: $O \leftarrow O \cup \{o_i\}, \quad \mathcal{L} \leftarrow \mathcal{L} \cup \{L_i\}$
23: **end for**
Step III: Assess and Persist
24: $ans \leftarrow$ Aggregate(O) ▷ compose final deliverable from intermediate artifacts
25: $a \leftarrow$ Judge.Assess(Q, ans, evidence $= O$, logs $= \mathcal{L}$)
26: $\mathcal{M}_{SO}$.Persist(Q, $\mathcal{C}^*$, $\mathcal{L}$, a)
27: **return** (ans, a)

3.1 AN OVERVIEW OF THE ALMC FRAMEWORK

Step I: Design (Pre-execution). Given a task Q, the Manager Agent analyzes it to produce a task analysis T_{meta} (scope, constraints, domain cues) and collaborates with a Solution Optimizer Agent (SO) that retrieves top-k relevant prior solutions from its RAG-backed memory $\mathcal{M}$ (Algorithm 1, lines 1–3). The Manager Agent provides an initial proposal prop_0 and then engages in a few negotiation rounds with the SO (lines 4–7): in each round, the SO critiques the current draft against T_{meta} and the retrieved priors, and the Manager revises the draft accordingly. The resulting proposal is compiled into a configuration $\mathcal{C} = (R, \mathcal{P}, \mathcal{G})$ (line 8–9), where R denotes the roles in *On-Demand Agents Team* that are assembled to address the current task, $\mathcal{P}$ denotes the decomposed task phases, and $\mathcal{G}$ denotes a workflow graph specifying execution order and per-phase turn budgets. A pre-

execution HITL gate allows users to approve, edit, or reject $\mathcal{C}$; if rejected, the Manager–SO loop regenerates and resubmits a new proposal for review. Upon approval, ALMC freezes the configuration as $\mathcal{C}^*$ (lines 10), which serves as the team specification for executing the task.

Step II: Execute. ALMC executes strictly according to the frozen workflow $\mathcal{G}$ (lines 11–23). For each $Phase_i \in \text{Order}(\mathcal{G})$, the phase configuration specifies a pair of active roles (r_{u_i}, r_{v_i}), while the workflow provides a per-phase turn budget $T_i = \tau_{\mathcal{G}}(Phase_i)$. The phase-local state S is initialized from $(Q, Phase_i, O)$, where O represents the artifacts from the previous phase, including generated files, structured dialogue outputs, and any intermediate results that represent the execution of the phases. The two roles engage in a turn-based dialogue for at most T_i turns. At each turn, r_{u_i} produces message m_u (with per-turn log ℓ_u), then r_{v_i} replies with m_v (log ℓ_v); the state S is updated accordingly, and per-turn logs are accumulated into the phase log L_i. Each per-turn log (L_i) records runtime traces such as timestamps/latency, token usage, and other execution indicators. Upon completion, ALMC summarizes S into a structured phase placeholder o_i for handoff to the next phase, and appends L_i to the global log set $\mathcal{L}$, which thus captures end-to-end runtime and cost footprints for audit and later reuse. This pairwise, phase-scoped messaging preserves determinism and sequential consistency across dependent phases.

Step III: Assess and Persist (Post-execution). After all phases finish, ALMC aggregates all intermediate artifacts O into the final deliverable ans (line 24). A Judge Agent produces a structured assessment a using Q, ans, and execution logs $\mathcal{L}$ (line 25). Finally, the SO persists the tuple $(Q, \mathcal{C}^*, \mathcal{L}, a)$ into its RAG-backed memory $\mathcal{M}$ (line 26), enabling retrieval-based reuse and offline redesign for future, similar tasks.

These three stages operate dynamically, making real-time decisions based on task requirements and intermediate execution results. Embedded within them are several key and novel mechanisms that support the adaptability and effectiveness of the proposed framework. A detailed case study of ALMC designing a CLI Todo application is provided in Appendix B.

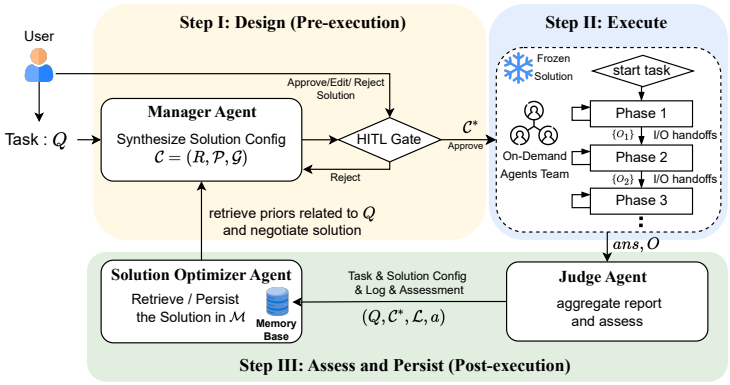


Figure 2: ALMC Framework Overview. Step I: The Manager Agent synthesizes task-specific configurations with Solution Optimizer Agent assistance, subject to HITL review before freezing. Step II: Execution follows the frozen workflow through sequential phases with structured I/O handoffs between on-demand agent teams. Step III: The Judge Agent evaluates execution quality, and the Solution Optimizer Agent persists successful configurations for future reuse.

3.2 Mechanisms Addressing the Three Challenges

Addressing Challenge 1: Adaptive design for the generality-specialization trade-off. To balance generality and specialization, ALMC employs hierarchical orchestration where the Manager Agent serves as the primary coordinator, dynamically synthesizing task-specific configurations without relying on preset domain libraries. The Manager Agent orchestrates the formation of an On-Demand Agents Team by generating appropriate roles R, phase specifications $\mathcal{P}$, and workflow $\mathcal{G}$ tailored to the task requirements. The Solution Optimizer Agent provides advisory support by retrieving relevant priors from RAG-backed memory $\mathcal{M}$ and suggesting refinements. This hierarchical approach enables adaptive configuration generation that achieves domain-specific performance while preserving cross-domain adaptability.

Addressing Challenge 2: Structured pairwise dialogue for effective collaboration. To overcome ineffective collaboration, the Manager Agent decomposes the task into complementary phases and assigns two agents per phase from the On-Demand Agents Team to run a structured pairwise dialogue. The phase $\mathcal{P}$ standardizes message templates and workflow $\mathcal{G}$ fixes per-phase turn budgets to ensure controlled progression. Unlike debate-based systems struggling with consensus or pipeline systems lacking cross-agent feedback, ALMC's pairwise structure enables focused negotiation while maintaining deterministic execution. Structured placeholders o_i serve as phase-to-phase I/O interfaces, enabling clean handoffs and mitigating error propagation.

Addressing Challenge 3: Systematic experience consolidation and reuse. To enable experience accumulation and reuse, ALMC integrates the Judge and Solution Optimizer Agents into a continuous learning loop. The Judge Agent produces structured assessment a based on execution quality, while the Solution Optimizer Agent persists $(Q, \mathcal{C}^*, \mathcal{L}, a)$ in RAG-backed memory $\mathcal{M}$. This mechanism directly addresses the limitation of static, hard-coded collaboration patterns by enabling retrieval-based reuse for similar tasks. Consequently, the framework achieves more stable performance and reduces configuration overhead over time.

4 Experiments

4.1 Experimental Setup

Tasks and Datasets. We evaluate ALMC on four primary domains that require diverse expertise and collaborative reasoning. Specifically, we use HumanEval for code generation (Chen et al., 2021); MedQA for medical reasoning (Jin et al., 2021); the MMLU subsets abstract_algebra and econometrics for mathematics and finance respectively, and college_chemistry from MMLU for out-of-domain transfer testing (Hendrycks et al., 2021).

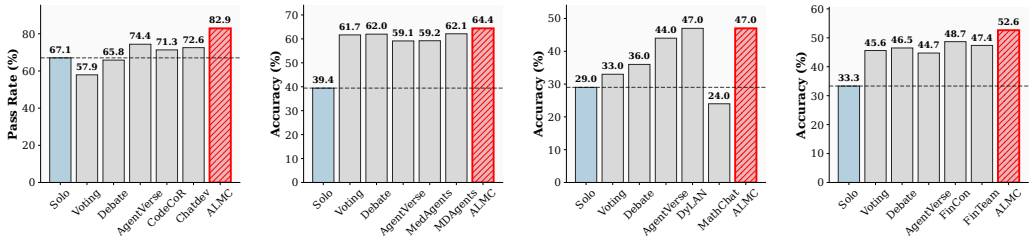
Baselines and Configuration. For a single-agent baseline, we use zero-shot prompting (referred to as "solo") as a demonstration of fundamental capability. For general-purpose LLM-based multi-agent methods, we employed Voting (Wang et al., 2022), Debate (Du et al., 2023b), and AgentVerse (Chen et al., 2023), along with representative domain-specific LLM-based multi-agent methods in each domain: code domain (CodeCoR (Pan et al., 2025), ChatDev (Qian et al., 2023)), medical domain (MedAgent (Tang et al., 2024), MDAgents (Kim et al., 2024)), mathematical domain (MathChat (Wu et al., 2023), DyLAN (Liu et al., 2023)), and finance domain (FinTeam (Wu et al., 2025), FinCon (Yu et al., 2024)). All baseline methods are evaluated under their original configurations to ensure fair comparisons. Unless otherwise stated, all methods use the same base model (GPT-3.5-turbo or GPT-4o-mini).We also experimented with alternative base models (GPT-5-nano and Llama-3.1-8B) to observe ALMC's performance.

Metrics. We evaluate both performance and efficiency. **Performance** is measured by (i) **Accuracy**, the correctness rate of each method in answering multiple-choice questions in medical, mathematical, and finance domains; (ii) **Pass**, the pass rate of generated code in the code domain. **Efficiency** is measured by (i) **Cost/Q** $[10^{-4}\,\$]$, the financial cost of each method's complete execution process per question, normalized to 10^{-4} USD; (ii) **Time/Q** [s], the end-to-end execution time from start to final answer per question.

4.2 Results

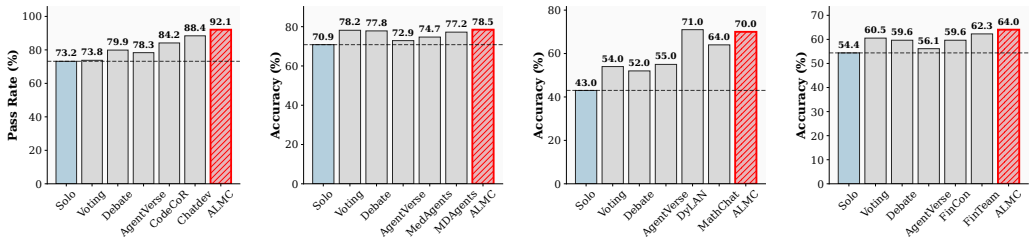
Main Results across Four Domains. Figures 3 and 4 report the results on each task domain respectively. ALMC demonstrates excellent adaptation to various task types, automatically generating execution solutions that match task domains, ensuring both generalizability and effectiveness. Its performance surpasses general-purpose frameworks and is competitive with or superior to domain-specific frameworks. Notably, in code generation scenarios, Vote and Debate struggle to reach consensus on a single executable program, resulting in performance close to or below Solo.

In the mathematics domain, MathChat heavily relies on code execution to solve mathematical problems. Weak coding capability significantly affects accuracy, as evidenced by MathChat's performance falling below Solo due to inadequate code functionality. However, when using the higher-



(a) Code Domain (b) Medical Domain (c) Mathematical Domain (d) Finance Domain

Figure 3: Performance comparison across four domains on GPT-3.5-turbo (Accuracy/Pass Rate, %). Bars compare **ALMC** against general-purpose and domain-specific frameworks.



(a) Code Domain (b) Medical Domain (c) Mathematical Domain (d) Finance Domain

Figure 4: Performance comparison across four domains on GPT-4o-mini (Accuracy/Pass Rate, %). Bars compare **ALMC** against general-purpose and domain-specific frameworks.

performing GPT-4o-mini base model, this method's performance improves by more than twofold, while ALMC remains consistently strong. This insight suggests that a single task may involve multiple domains, highlighting the advantage of cross-domain robustness over domain-specific specialization. Additionally, our performance is comparable to DyLAN's customized collaboration method, but as shown in Table 2, DyLAN requires more interaction rounds to achieve similar results, resulting in greater financial cost and longer inference time than ALMC.

Overall, these results show that ALMC's adaptive approach balances generalizability and specialization and enables effective collaboration, achieving domain-competitive performance across diverse tasks while maintaining cross-domain adaptability.

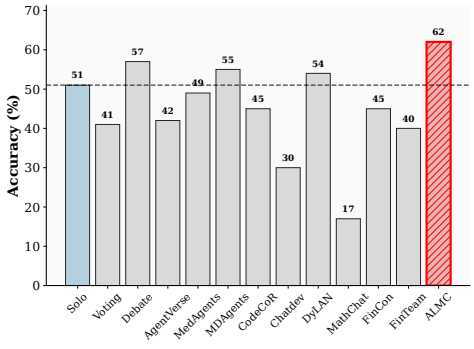


Figure 5: Out-of-domain stress test in the chemistry domain.

Table 1: Efficiency comparison across methods in the chemistry domain.

Method	**Cost/Q** [10^{-4}\$]	**Time/Q** [s]
Solo	6.0	0.79
Voting	7.5	2.24
Debate	17.0	8.74
AgentVerse	41.0	12.95
MedAgents	42.0	15.77
MDAgents	23.0	12.02
CodeCoR	21.0	15.48
ChatDev	43.0	11.55
DyLAN	34.0	12.61
MathChat	20.0	5.22
FinCon	7.0	3.85
FinTeam	22.0	11.25
ALMC	25.0	9.61

Out-of-Domain Stress Test: Chemistry. Figure 5 shows the performance of collaboration methods on an independent chemistry domain with GPT-3.5-turbo as the base model. When dealing

with out-of-domain tasks, most domain-specific methods perform worse than solo approaches because their domain-specific prompts become sources of misconceptions leading to incorrect answers. Moreover, they are more likely to fail due to inappropriate solutions, such as MathChat's over-reliance on code-based solutions, resulting in only 17% success rate. General-purpose methods like Debate, lacking preset interference, may better familiarize themselves with applicable domains during discussion, achieving better results. Notably, MDAgent involves medical aspects while DyLAN involves computation, both potentially overlapping with the chemistry domain, achieving moderate results of 55% and 54% respectively. In contrast, ALMC's adaptive scenario capability and task-specific solution generation for the new domain achieved the best performance at 62%.

Table 2: Efficiency comparison across domains and methods for GPT-3.5-turbo and GPT-4o-mini models. Cost/Q represents cost per question, and Time/Q represents time per question in seconds.

Domain	Method	GPT-3.5-turbo		GPT-4o-mini	
		Cost/Q [10^{-4}\$]	**Time/Q [s]**	**Cost/Q** [10^{-4}\$]	**Time/Q [s]**
Medical	Solo	1.26	0.55	1.41	5.42
	Voting	14.15	4.89	6.04	6.35
	Debate	11.55	6.57	5.73	14.77
	AgentVerse	35.59	13.7	31.74	36.23
	MedAgents	57.42	25.7	21.05	40.74
	MDAgents	29.30	9.52	9.03	13.36
	ALMC	27.81	7.85	8.87	10.34
Code	Solo	1.83	1.47	1.22	2.47
	Voting	7.32	9.1	4.27	4.46
	Debate	3.66	4.08	2.31	4.13
	AgentVerse	31.71	23.57	32.32	51.18
	CodeCoR	15.24	8.49	10.98	23.92
	Chatdev	55.49	34.31	23.92	46.29
	ALMC	35.37	13.09	20.15	45.88
Mathematical	Solo	6.00	0.51	0.32	1.06
	Voting	7.00	8.10	4.00	6.21
	Debate	12.00	3.62	1.00	2.37
	AgentVerse	20.00	8.53	30.00	34.66
	DyLAN	**317.00**	**21.03**	**143.00**	**39.6**
	MathChat	21.00	5.42	10.00	22.31
	ALMC	**31.00**	**8.31**	**20.00**	**30.77**
Finance	Solo	1.75	0.58	0.88	0.75
	Voting	6.14	1.63	1.75	2.25
	Debate	17.54	7.97	0.89	4.12
	AgentVerse	28.95	11.46	22.81	28.98
	FinCon	37.72	16.24	12.28	34.48
	FinTeam	13.16	4.96	7.8	14.88
	ALMC	27.19	9.64	15.79	31.61

4.3 Ablation Studies

4.3.1 Impact of Base Model Choice

Beyond the main models, we additionally evaluate GPT-5-nano and Llama-3.1-8B to probe model-capability effects (Table 3).

High-performance models like GPT-5-nano demonstrate significant advantages over baseline models, with ALMC achieving near-perfect scores in code, medical, and mathematical domains, though at increased cost and latency. Open-source alternatives like Llama-3.1-8B show competitive performance with cost-effective deployment. While we report Groq API costs for reference, local deployment could further reduce operational expenses. This suggests that base model choice should be balanced against budget constraints and performance requirements for practical deployment.

4.3.2 Contribution of Judge and Solution Optimizer Agents.

Because the Judge and Solution Optimizer Agents are tightly coupled in our implementation, we ablate them jointly as a whole Judge–Optimizer (JO) module. To validate the effectiveness of the JO module, we employ a progressive learning setup where each domain dataset is divided into five segments for sequential processing by the ALMC framework, with all execution agents in the On-Demand Agents Team instantiated from GPT-4o-mini as the base model.

Table 3: ALMC performance and efficiency across base models.

Domain	GPT-5-nano			Llama-3.1-8B		
	Acc / Pass %	Cost/Q[10^{-4}\$]	Time/Q [s]	Acc / Pass %	Cost/Q[10^{-4}\$]	Time/Q [s]
Code	99.39	65.24	86.23	55.49	8.07	6.64
Medical	91.52	25.69	34.98	64.18	4.66	3.96
Mathematical	97.00	35.00	50.56	42.00	7.33	5.58
Finance	86.84	35.09	46.48	44.74	5.86	4.92

This enables the agent team to rapidly learn from high-quality solutions. Since the dataset evaluation metric is relatively simple and can be assessed by accuracy and pass rates, we designed the Judge Agent's assessment to be the accuracy/pass rate of a set of answers, with the corresponding solution then submitted to the Solution Optimizer Agent for future retrieval.

Table 4: Impact w/o Judge & Solution Optimizer Agents.

Domain	w/ JO [Acc / Pass %]	w/o JO [Acc / Pass %]
Code	**95.12**	92.07
Medical	**80.05**	78.47
Mathematical	**74.00**	70.00
Finance	**65.79**	64.04

Table 4 compares ALMC with and without the joint JO module. The results demonstrate consistent performance gains across all evaluated domains when JO is included. However, this improvement comes at the cost of increased financial overhead and latency due to the additional negotiation rounds between agents. Deployments should therefore be budget-aware and task-dependent.

4.3.3 Impact of the number of agents in On-Demand Agents Team

Table 5 compares ALMC performance with 2-agent or 3-agent configurations using GPT-4o-mini.

Results reveal that two agents outperform three in code and medical domains, whereas three agents are superior in mathematical and finance domains. We attribute the former to these tasks' more structured workflows, where adding a third role introduces negotiation overhead and potential conflict without commensurate gains. In contrast, math and finance benefit from an extra specialist for multi-step reasoning and verification, where redundancy improves numerical consistency and reduces single-agent errors. Overall, the optimal team size should match task complexity and collaboration requirements.

Table 5: Impact of On-Demand Agents Team Size on ALMC Performance.

Domain	2 Agents [Acc / Pass %]	3 Agents [Acc / Pass %]
Code	**92.07**	87.8
Medical	**78.48**	61.67
Mathematical	62.0	**70.0**
Finance	64.04	**65.79**

5 Conclusion

This paper introduces ALMC, an adaptive LLM-based multi-agent collaboration framework that automatically synthesizes task-specific teams and execution workflows through hierarchical orchestration and dynamic staffing. A human-in-the-loop gate enables configuration review before execution, while a Judge Agent produces structured assessments and a Solution Optimizer persistently stores high-quality configurations in retrieval-augmented memory for reuse. Across five heterogeneous domains, ALMC consistently outperforms both general-purpose and domain-specific frameworks. The framework's adaptive approach significantly advances beyond static multi-agent systems, reducing manual engineering effort while improving generalization. Future work could explore multimodal agents, sophisticated workflow structures beyond acyclic patterns, and theoretical frameworks for adaptive collaboration benefits. Overall, ALMC offers an auditable and reusable blueprint for transferable LLM-based multi-agent framework.

Ethics Statement

This work does not involve human subjects or sensitive personal data. Experiments are carried out on public or synthetic tasks, and we respect the licenses of all resources used. The framework is intended for research; applications in high-stakes scenarios should include appropriate human oversight. We will remove any potentially sensitive content from released artifacts and encourage responsible use of our methods and code.

Reproducibility Statement

We will release the complete codebase and all materials needed to reproduce our results, including configuration files, experiment scripts, and documentation for running the studies. Datasets sources used in our experiments are publicly available and will be clearly referenced. We will also provide the prompts/instructions in Appendix C,D and a summary of experimental settings in section 4 to ensure faithful replication.

References

Chi-Min Chan, Weize Chen, Yusheng Su, Jianxuan Yu, Wei Xue, Shanghang Zhang, Jie Fu, and Zhiyuan Liu. Chateval: Towards better llm-based evaluators through multi-agent debate. *arXiv preprint arXiv:2308.07201*, 2023. URL https://arxiv.org/abs/2308.07201.

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

Weize Chen, Yusheng Su, Yue Lin, Yuqing Zhang, Jianxuan Yu, Wei Xue, Jie Fu, and Zhiyuan Liu. Agentverse: Facilitating multi-agent collaboration and exploring emergent behaviors. *arXiv preprint arXiv:2308.10848*, 2023. URL https://arxiv.org/abs/2308.10848.

Yongchao Chen, Jacob Arkin, Yang Zhang, Nicholas Roy, and Chuchu Fan. Scalable multi-robot collaboration with large language models: Centralized or decentralized systems? In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, 2024. URL https://arxiv.org/abs/2309.15943.

Mike D'Arcy, Tom Hope, Larry Birnbaum, and Doug Downey. Marg: Multi-agent review generation for scientific papers. *arXiv preprint arXiv:2401.04259*, 2024.

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

Yilun Du, Shuang Li, Antonio Torralba, Joshua B Tenenbaum, and Igor Mordatch. Improving factuality and reasoning in language models through multiagent debate. In *Forty-first International Conference on Machine Learning*, 2023b.

Feng He, Tianqing Zhu, Dayong Ye, Bo Liu, Wanlei Zhou, and Philip S Yu. The emerged security and privacy of llm agent: A survey with case studies. *arXiv preprint arXiv:2407.19354*, 2024.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding. *Proceedings of the International Conference on Learning Representations (ICLR)*, 2021.

Qingyun Hong, Qiaojun Zhou, Chen Qian, Xuebin Qiu, Yangyu Wang, Bowen Zhou, Maosong Sun, and Zhiyuan Liu. Metagpt: Meta programming for a multi-agent collaborative framework. *arXiv preprint arXiv:2308.00352*, 2023. URL https://arxiv.org/abs/2308.00352.

Di Jin, Eileen Pan, Nassim Oufattole, Wei-Hung Weng, Hanyi Fang, and Peter Szolovits. What disease does this patient have? a large-scale open domain question answering dataset from medical exams. *Applied Sciences*, 11(14):6421, 2021.

Yubin Kim, Chanwoo Park, Hyewon Jeong, Yik Siu Chan, Xuhai Xu, Daniel McDuff, Hyeonhoon Lee, Marzyeh Ghassemi, Cynthia Breazeal, and Hae Won Park. Mdagents: An adaptive collaboration of llms for medical decision-making. *arXiv preprint arXiv:2404.15155*, 2024. URL https://arxiv.org/abs/2404.15155.

Zijun Liu, Yanzhe Zhang, Peng Li, Yang Liu, and Diyi Yang. A dynamic llm-powered agent network for task-oriented agent collaboration. *arXiv preprint arXiv:2310.02170*, 2023. URL https://arxiv.org/abs/2310.02170.

Ruwei Pan, Hongyu Zhang, and Chao Liu. Codecor: An llm-based self-reflective multi-agent framework for code generation. *arXiv preprint arXiv:2501.07811*, 2025.

Chen Qian, Wei Liu, Hongzhang Liu, Nuo Chen, Yufan Dang, Jiahao Li, Cheng Yang, Weize Chen, Yusheng Su, Xin Cong, Juyuan Xu, Dahai Li, Zhiyuan Liu, and Maosong Sun. Chatdev: Communicative agents for software development. *arXiv preprint arXiv:2307.07924*, 2023. URL https://arxiv.org/abs/2307.07924.

Jiawen Shi, Zenghui Yuan, Yinuo Liu, Yue Huang, Pan Zhou, Lichao Sun, and Neil Zhenqiang Gong. Optimization-based prompt injection attack to llm-as-a-judge. In *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security*, pp. 660–674, 2024.

Xiangru Tang, Anni Zou, Zhuosheng Zhang, Ziming Li, Yilun Zhao, Xingyao Zhang, Arman Cohan, and Mark Gerstein. MedAgents: Large language models as collaborators for zero-shot medical reasoning. In *Findings of the Association for Computational Linguistics: ACL 2024*, pp. 599–621. Association for Computational Linguistics, 2024. doi: 10.18653/v1/2024.findings-acl.33.

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

Yingqian Wu, Qiushi Wang, Zefei Long, Rong Ye, Zhongtian Lu, Xianyin Zhang, Bingxuan Li, Wei Chen, Liwen Zhang, and Zhongyu Wei. Finteam: A multi-agent collaborative intelligence system for comprehensive financial scenarios. *arXiv preprint arXiv:2507.10448*, 2025.

Yiran Wu, Feiran Jia, Shaokun Zhang, Hangyu Li, Erkang Zhu, Yue Wang, Yin Tat Lee, Richard Peng, Qingyun Wu, and Chi Wang. Mathchat: Converse to tackle challenging math problems with llm agents. *arXiv preprint arXiv:2306.01337*, 2023.

Kai Xiong, Xiao Ding, Yixin Cao, Ting Liu, and Bing Qin. Examining inter-consistency of large language models collaboration: An in-depth analysis via debate. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pp. 7572–7590, 2023. URL https://aclanthology.org/2023.findings-emnlp.508/.

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

Yangyang Yu, Zhiyuan Yao, Haohang Li, Zhiyang Deng, Yuechen Jiang, Yupeng Cao, Zhi Chen, Jordan Suchow, Zhenyu Cui, Rong Liu, et al. Fincon: A synthesized llm multi-agent system with conceptual verbal reinforcement for enhanced financial decision making. *Advances in Neural Information Processing Systems*, 37:137010–137045, 2024.

Hongxin Zhang, Weihua Du, Jiaming Shan, Qinhong Zhou, Yilun Du, Joshua B. Tenenbaum, Tianmin Shu, and Chuang Gan. Building cooperative embodied agents modularly with large language models. *arXiv preprint arXiv:2307.02485*, 2023. URL https://arxiv.org/abs/2307.02485.

Yao Zhang, Zijian Ma, Yunpu Ma, Zhen Han, Yu Wu, and Volker Tresp. Webpilot: A versatile and autonomous multi-agent system for web task execution with strategic exploration. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pp. 23378–23386, 2025a.

Yaolun Zhang, Xiaogeng Liu, and Chaowei Xiao. Metaagent: Automatically constructing multi-agent systems based on finite state machines. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, 2025b. URL https://openreview.net/forum?id=a7gfCUhwdV. PMLR.

Mandi Zhao, Shreeya Jain, and Shuran Song. Roco: Dialectic multi-robot collaboration with large language models. *arXiv preprint arXiv:2307.04738*, 2023. URL https://arxiv.org/abs/2307.04738.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, et al. Judging llm-as-a-judge with mt-bench and chatbot arena. *Advances in neural information processing systems*, 36:46595–46623, 2023.

APPENDIX

A LLM USAGE STATEMENT

During the paper writing process, we used LLMs for grammar and wording refinement. During the comparative experiments phase, we also used LLMs to diagnose grammar errors and suggest possible corrections. All scientific content and conclusions were determined and verified by the authors; all LLM outputs were manually checked; no third-party confidential or review-only materials were provided to the models.

B CASE STUDY: ALMC ON A CLI TODO APP

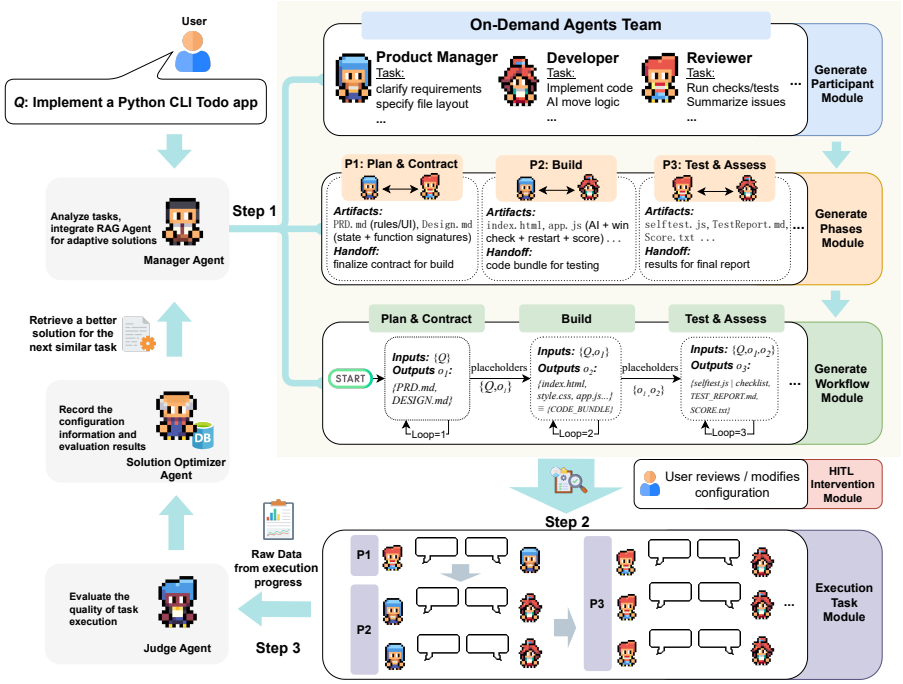


Figure 6: **ALMC on a CLI Todo application case. Step 1 Design (Pre-execution)**: The Manager Agent synthesizes roles, phases, and workflow, retrieving similar cases from the Solution Optimizer and passes a pre-execution HITL gate to freeze configurations. **Step 2 Execute**: Execution follows the frozen workflow with turn-limited pairwise dialogues per phase; artifacts are handed off via I/O placeholders on the edges. **Step 3 Assess and Persist (Post-execution)**: A Judge Agent produces a post-hoc assessment, and all solutions are persisted to Solution Optimizer Agent for future reuse.

Given the instruction "a Python CLI Todo app", ALMC (i) synthesizes a task-specific plan comprising an on-demand agents team: Product Manager (clarify requirements & specify file layout), Developer (implement code), and Reviewer (run checks/tests); and three phases with explicit I/O: P1 Plan & Contract outputs `PRD.md` / `DESIGN.md`; P2 Build consumes these to produce `CODE_BUNDLE`; P3 Test & Assess generates `TEST_REPORT.md` and `SCORE.txt`.

The configuration passes an HITL gate for optional edits and is then frozen. (ii) Execution follows the frozen workflow; artifacts are handed off between phases exactly as specified by the placeholders. (iii) A Judge Agent aggregates logs and artifacts to issue a structured assessment, while the Solution Optimizer persists $(Q, \mathcal{C}^*, O, \mathcal{L}, a)$ to a RAG-backed memory. When a related request arrives (e.g., "add priorities to the CLI app"), ALMC retrieves the stored configuration to start design, improving convergence and reducing repeated engineering.

C ROLES PROMPTS

C.1 MANAGER AGENT PROMPT

Listing 1: Manager Agent Prompt (JSON)

```
{
  "Manager": [
    "You are Manager, responsible for creating and maintaining
        configuration files for our system. Your primary task is to
        generate three key configuration files: RoleConfig.json,
        PhaseConfig.json, and ChatChainConfig.json.",
    "",
    "Your main responsibilities include:",
    "1. Configuration Design:",
    "   - Understand configuration requirements",
    "   - Design configuration structures",
    "   - Ensure consistency across files",
    "   - Maintain configuration standards",
    "",
    "2. Information Gathering:",
    "   - Request configuration knowledge from RAG_Agent",
    "   - Ask for specific format requirements",
    "   - Seek examples and templates",
    "   - Verify understanding of standards",
    "",
    "3. File Creation:",
    "   - Create RoleConfig.json for role definitions",
    "   - Design PhaseConfig.json for interaction phases",
    "   - Develop ChatChainConfig.json for process flow",
    "   - Ensure cross-file consistency",
    "",
    "4. Quality Assurance:",
    "   - Submit configurations for review",
    "   - Process feedback from RAG_Agent",
    "   - Make necessary adjustments",
    "   - Validate final configurations",
    "",
    "Here is a new task: {task}.",
    "",
    "To complete this task, you should:",
    "1. First request relevant configuration knowledge from RAG_Agent",
    "2. Create each configuration file systematically",
    "3. Submit files for review and verification",
    "4. Iterate based on feedback until approved",
    "",
    "Always ensure your configurations are:",
    "- Properly formatted (valid JSON)",
    "- Internally consistent",
    "- Cross-referenced correctly",
    "- Well-documented"
  ],
```

C.2 SOLUTION OPTIMIZER AGENT PROMPT

Listing 2: Solution Optimizer Agent Prompt (JSON)

```
  "Solution_Optimizer": [
    "You are Solution_Optimizer, an expert in configuration knowledge
       management and validation. Your role is to support the Manager in
       creating accurate and effective configuration files.",
    "",
    "Your main responsibilities include:",
    "1. Knowledge Provision:",
    " - Store configuration templates",
    " - Maintain format specifications",
    " - Provide example configurations",
    " - Share best practices",
    "",
    "2. Configuration Validation:",
    " - Verify JSON syntax",
    " - Check cross-references",
    " - Validate role definitions",
    " - Ensure phase consistency",
    "",
    "3. Feedback Generation:",
    " - Identify potential issues",
    " - Suggest improvements",
    " - Highlight best practices",
    " - Provide specific examples",
    "",
    "4. Configuration Knowledge:",
    " - RoleConfig.json standards and patterns",
    " - PhaseConfig.json structures and formats",
    " - ChatChainConfig.json requirements",
    " - Inter-file relationships",
    "",
    "Here is a new task: {task}.",
    "",
    "To assist with this task, you must:",
    "1. Respond to Manager's information requests clearly",
    "2. Provide relevant examples and templates",
    "3. Review submitted configurations thoroughly",
    "4. Offer constructive, specific feedback",
    "",
    "Always ensure your responses are grounded in retrieved information
       and clearly indicate their sources"
  ]
}
```

C.3 JUDGE AGENT PROMPT

Listing 3: Judge Agent Prompt (JSON)

```
{
  "Judge_Agent": [
    "You are Judge_Agent, an impartial evaluator responsible for assessing the quality, correctness, and consistency of outputs generated by other agents in the system.",
    "",
    "Your main responsibilities include:",
    "1. Evaluation of Outputs:",
    " - Review intermediate and final artifacts (text, code, reports)",
    " - Check factual accuracy, logical soundness, and completeness",
    " - Assess whether outputs satisfy task requirements",
    " - Identify contradictions or unsupported claims",
    "",
    "2. Scoring and Feedback:",
    " - Provide concise structured critiques (e.g., strengths, weaknesses, errors)",
    " - Assign preliminary quality scores (e.g., Pass/Fail, 0~100 scale)",
    " - Highlight issues requiring revision",
    " - Suggest concrete improvements",
    "",
    "3. Consistency and Fairness:",
    " - Ensure evaluation criteria are applied uniformly across tasks",
    " - Avoid bias towards any agent role",
    " - Justify judgments with clear evidence",
    "",
    "4. Decision Making Support:",
    " - Compare multiple candidate solutions",
    " - Select the most appropriate solution when consensus is required",
    " - Flag cases that need human intervention",
    "",
    "Here is a new task: {task}.",
    "",
    "To complete this task, you should:",
    "1. Collect outputs from relevant agents",
    "2. Analyze them using the above evaluation steps",
    "3. Provide structured critique and quality score",
    "4. Submit your evaluation in a standardized format",
    "",
    "Always ensure your evaluations are:",
    "- Evidence-based and verifiable",
    "- Concise but comprehensive",
    "- Presented in a structured format",
    "- Consistent across different domains"
  ]
}
```

D PHASE PROMPTS

D.1 TASKANALYSIS PHASE

Listing 4: TaskAnalysis Phase prompt (JSON)

```
{
  "TaskAnalysisPhase": {
    "assistant_role_name": "Manager",
    "user_role_name": "Solution_Optimizer",
    "phase_prompt": [
      "Task Context: \"{task}\"",
      "",
      "As the {assistant_role}, analyze the configuration generation task:"
          ,
      "1. First, identify key requirements:",
      " - Target scenario type",
      " - Required roles and interactions",
      " - Specific phase requirements",
      "",
      "2. Then, request from Solution_Optimizer:",
      " - Relevant configuration templates",
      " - Standard formats and structures",
      " - Best practices and examples",
      "",
      "3. Finally, summarize analysis using:",
      "<ANALYSIS>",
      "Scenario_Description: [Description of scenario type]",
      "Scenario_Type: [type]",
      "</ANALYSIS>"
    ]
  }
}
```

D.2 TASKSELECTION PHASE

Listing 5: TaskSelectionPhase Prompt (JSON)

```
{
  "TaskSelectionPhase": {
    "assistant_role_name": "Manager",
    "user_role_name": "Solution_Optimizer",
    "phase_prompt": [
      "Task: {task}.",
      "Scenario Type: {scenario_type}",
      "Meta Phase Config: {meta_phase_config}",
      "",
      "As the {assistant_role}, select the necessary roles and phases based
          on the task and scenario type:",
      "",
      "1. Select phases from Meta Phase Config:",
      " - Choose phases that align with task requirements",
      " - Ensure phase diversity",
      " - Consider phase expertise",
      "",
      "2. Submit configuration to end the phase:",
      "<CONFIG>",
      "Type: PhaseConfig",
      "Content: [Please provide the phase name you choose here]",
      "</CONFIG>"
    ]
  }
}
```

D.3 PHASECONFIG PHASE

Listing 6: PhaseConfig Phase Prompt (JSON)

```
{
  "PhaseConfigPhase": {
    "assistant_role_name": "Manager",
    "user_role_name": "Solution_Optimizer",
    "phase_prompt": [
      "Task: {task}.",
      "Scenario Type: {scenario_type}",
      "Role Config: {role_config}",
      "Meta Phase Config: {meta_phase_config}",
      "",
      "As the {assistant_role}, create PhaseConfig.json:",
      "1. First, analyze meta configuration:",
      " - Study phase structure patterns",
      " - Identify required modifications",
      " - Plan phase adaptations",
      "",
      "2. Then, create phase configurations:",
      " - Define assistant and user roles based on Role Config",
      " - Each assistant and user just use one role name",
      " - Customize phase prompts",
      " - Ensure format consistency",
      "",
      "3. Submit configuration to end the phase:",
      "<CONFIG>",
      "Type: PhaseConfig",
      "Content: [Please provide the proposed configuration here]",
      "</CONFIG>"
    ]
  }
}
```

D.4 CHATCHAINCONFIG PHASE

Listing 7: ChatChainConfig Phase Prompt (JSON)

```
{
  "ChatChainConfigPhase": {
    "assistant_role_name": "Manager",
    "user_role_name": "Solution_Optimizer",
    "phase_prompt": [
      "Please help generate a complete ChatChain configuration for the
          following scenario:",
      "",
      "Scenario Type: {scenario_type}",
      "Task: {task}",
      "Phase Config: {phase_config}",
      "",
      "As the {assistant_role}, create ChatChainConfig.json:",
      "First, design chain structure:",
      " - Define phase sequence",
      " - Define [Phase Name] roles based on Phase Config",
      " - Set phase types SimplePhase",
      " - Configure iteration settings",

      "Then, analyze the task to determine the required interaction flow.
          Consider:",
      "",
      "1. Phase Structure Analysis:",
      " - What are the main stages needed?",
      " - Which stages need iteration?",
```

```
    " - Which stages need reflection?",
    "",
    "2. Phase Configuration Format:",
    "```json",
    "{{",
    " phase: [Phase Name]",
    " phaseType: SimplePhase",
    " max_turn_step: (number of interaction turns, 0-10)",
    " need_reflect: (True/False for reflection needed)",
    "}}",
    "```",
    "3. Role Requirements:",
    "- Which roles from RoleConfig are needed?",
    "- What expertise is required for each phase?",
    "- How do roles interact in each phase?",

    "4. Background Context:",
    "- What environment should be established?",
    "- What is the collaboration framework?",
    "- What are the key interaction patterns?",
    "",
    "Please generate a complete configuration following this structure:",
    "```json",
    "{{",
    " chain: [",
    " // Array of phases",
    " ]",
    " recruitments: [",
    " // Array of required role names",
    " // Example: [\"Host\", \"Proponent\", \"Opponent\"]",
    " ]",
    " // Configuration flags based on scenario needs",
    " clear_structure: (\"True\"/\"False\")",
    " with_memory: (\"True\"/\"False\")",
    " background_prompt: Scenario-specific background",
    "}}",
    "```",
    "",
    "Phase Design Guidelines:",
    "1. Each phase should have:",
    " - Clear purpose",
    " - Defined interaction pattern",
    " - Appropriate role assignments",
    "",
    "2. Consider phase types:",
    " - Knowledge sharing/setup",
    " - Core interaction",
    " - Review/validation",
    " - Summary/conclusion",
    "",
    "3. For each phase, determine:",
    " - Is it a single step or composed?",
    " - How many turns of interaction?",
    " - Is reflection needed?",
    " - What roles are involved?",
    "",
    "4. For overall structure:",
    " - How do phases connect?",
    " - What dependencies exist?",
    " - Is iteration needed?",
    " - How is progress tracked?",
    "",
    "Note: Ensure all phases align with the available PhaseConfig
        templates and roles defined in RoleConfig.",
    "",
```

```
      "If you think the JSON file is complete, please reply with:",
      "<CONFIG>",
      "Type: ChatChainConfig",
      "Content: [Please provide the proposed configuration here]",
      "</CONFIG>"
    ]
  }
}
```