OMAC: A Holistic Optimization Framework for LLM-Based Multi-Agent Collaboration

Shijun Li [1] **Hilaf Hasson** [2] **Joydeep Ghosh** [1]

Abstract

Agents powered by advanced large language models (LLMs) have demonstrated impressive capabilities across diverse complex applications. Recently, Multi-Agent Systems (MAS), wherein multiple agents collaborate and communicate with each other, have exhibited enhanced capabilities in complex tasks, such as high-quality code generation and arithmetic reasoning. However, the development of such systems often relies on handcrafted methods, and the literature on systematic design and optimization of LLM-based MAS remains limited. In this work, we introduce **OMAC**, a general framework designed for holistic optimization of LLM-based MAS. Specifically, we identify five key optimization dimensions for MAS, encompassing both agent functionality and collaboration structure. Building upon these dimensions, we first propose a general algorithm, utilizing two actors termed the Semantic Initializer and the Contrastive Comparator, to optimize any single dimension. Then, we present an algorithm for joint optimization across multiple dimensions. Extensive experiments demonstrate the superior performance of OMAC on diverse tasks against recent approaches. Codes are available at: https://github.com/xiwenchao/OMAC.

1. Introduction

Autonomous agents leveraging advanced large language models (LLMs) have shown significant potential in addressing complex problems and executing diverse tasks (Shinn et al., 2023). Recently, employing multiple collaborating agents has emerged as a prominent research direction for overcoming the inherent limitations of single-agent approaches in handling tasks within sophisticated environments ((Li et al., 2023a; Wang et al., 2023c)). An advanced collaborative paradigm also involves a multi-step process wherein agents sequentially resolve tasks by utilizing outputs from preceding steps. This Multi-Agent Systems (MAS) approach has already yielded substantial advancements in various applications, such as code generation (Shinn et al., 2023), reasoning (Du et al., 2023), and decision making (Sun et al., 2024).

However, the design of existing MAS predominantly relies on hand-crafted strategies. Regarding agent construction, prior studies typically employ methods based either on human expertise (Liu et al., 2023) or LLM generation (Wang et al., 2023c). Although some research has explored optimizing the functionality of a single agent via techniques such as fine-tuning (Huang et al., 2024) or prompt-tuning (Wang et al., 2023a), these efforts don't explicitly address agent optimization within the context of MAS involving multi-step collaborative processes. Concerning collaboration structures, existing approaches typically define fixed architectures tailored to specific application scenarios, encompassing cooperative tasks (e.g., code generation (Dong et al., 2023), decision-making (Qian et al., 2023)) and competitive tasks (e.g., debate (Du et al., 2023)). However, the design of these structures predominantly relies on human prior knowledge or particular empirical findings, thereby limiting their generality and the flexibility required for autonomous optimization towards more effective structural configurations. A recent study, DyLAN (Liu et al., 2024), represents a promising effort toward optimizing collaboration structures in MAS. However, its approach centers solely on the unsupervised optimization of agent team composition, employing metrics derived from preliminary trials and LLM-based judgments, rather than utilizing supervised signals for validation and optimization. Similarly, other recent works such as G-Designer (Zhang et al., 2025b) and MaAS (Zhang et al., 2025a) focus on architectural search for identifying effective multi-agent structures, thereby only addressing the structural optimization dimension of MAS.

To address these challenges, we introduce **OMAC**, a comprehensive framework designed for the integrated optimization of MAS. Specifically, we identify and summarize **five**

[*]Equal contribution [1]Department of Electrical and Computer Engineering, The University of Texas at Austin, Austin, United States [2]Intuit AI Research, Mountain View, United States. Correspondence to: Shijun Li <shijunli@utexas.edu>.

Proceedings of the 43rd International Conference on Machine Learning, Seoul, South Korea. PMLR 306, 2026. Copyright 2026 by the author(s).

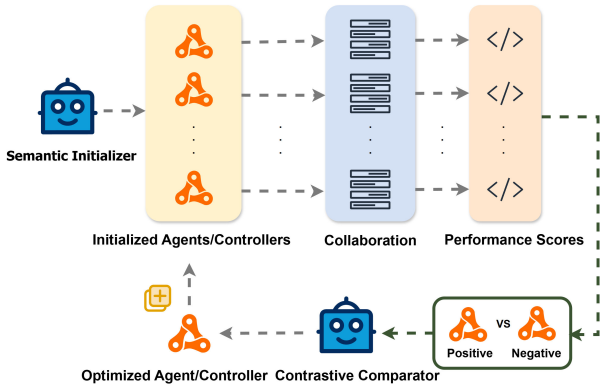

Figure 1. OMAC workflow for a single dimension.

key optimization dimensions pertaining to both agent functionality and collaboration structure. The functional dimensions include optimizing existing agents and constructing new agents tailored for collaboration. Regarding the structural aspect, we optimize[1] LLM-based controllers to manage collaboration structures, encompassing decisions on the overall candidate agent teams, dynamic agent selection for individual collaboration steps, and the mechanisms governing inter-agent communication.

Then we first propose a general algorithm employing two LLM-powered actors, the **Semantic Initializer** and the **Contrastive Comparator**, to optimize any individual dimension. Specifically, the Semantic Initializer leverages the knowledge and reasoning capabilities of LLMs to generate an initial collection of agents or controllers corresponding to the dimension being optimized by exploring the relevant semantic space. Subsequently, each initialized agent or controller undergoes evaluation within the multi-agent collaboration process on the training set to determine its performance score. A positive-negative pair (high-performing and low-performing) is then sampled based on these scores. The Contrastive Comparator then performs contrastive reasoning on this pair to identify factors underlying the performance gap, subsequently generating a refined agent or controller. This refined agent/controller is then re-evaluated via the collaboration process, and the cycle of sampling, contrastive comparison, and refinement is iterated. Beyond single-dimension optimization, we further propose an algorithm for iterative, joint optimization across multiple dimensions, enabling further MAS enhancement through synergistic optimization of agents and controllers. More implementation details of the actors and optimization algorithm are illustrated in Section 3.2.

We conduct extensive experiments across diverse tasks, including general reasoning, code generation, and arithmetic reasoning. The results demonstrate that OMAC consistently outperforms strong baselines when optimizing each of the five individual dimensions. Furthermore, joint optimization across multiple dimensions is shown to further significantly enhance MAS's performance.

Our key contributions can be summarized as follows:

- We introduce OMAC, a general supervised framework for optimizing multi-agent systems engaged in multi-step collaboration. To achieve this, we identify five key optimization dimensions covering both agent functionality and collaboration structure.
- We propose a general algorithm, utilizing two actors termed the Semantic Initializer and the Contrastive Comparator, for optimizing each of the five dimensions. Furthermore, we present an algorithm enabling iterative, joint optimization across multiple dimensions.
- We evaluate OMAC on benchmark tasks including general reasoning, code generation, and arithmetic reasoning. Empirical results demonstrate that OMAC significantly outperforms existing approaches through the automated optimization of functional and structural MAS designs.

2. Problem Definition

We study the optimization of MAS within the context of multi-step collaboration process. The definition of agents and the fundamental collaboration workflow are as follows:

Definition 1: Agents. An LLM-powered agent is governed by natural language prompts to generate solutions for completing a given task. Formally, we define an agent a as a function: $\mathcal{A} : \mathcal{P} \times \mathcal{I} \rightarrow \mathcal{O}$. Here, $p \in \mathcal{P}$ represents the instruction prompt defining the role and functionality of the agent. In an in-context learning setting, the prompt may also include few-shot examples serving as demonstrations to guide the agent's behavior. The input $i \in \mathcal{I}$ typically comprises the query information, task description, and potentially instructions or solutions from other agents in MAS. The output $o \in \mathcal{O}$ generally encompasses the agent's generated analyses, reasoning, and solutions to the given task.

Definition 2: Multi-Step Collaboration. In this work, we consider multiple agents collaborating within a multi-step procedure. Typically, each step $s_t \in \mathcal{S}$ involves a subset of agents from the overall team, denoted as $\{a_{t,1}, a_{t,2}, ..., a_{t,n}\} \subseteq s_t$. This multi-step approach aims to enhance problem-solving by enabling agents at each step to leverage solutions and outcomes produced by agents in preceding steps. For instance, when solving a complex mathematical problem, agents in the initial step may first decompose the problem into a series of simpler subproblems. Agents in subsequent steps can then address each subproblem by integrating analyses and solutions generated

[1] Note that we use the word "optimize" in a colloquial way to signify improvement instead of mathematical guarantee of an optimal solution.

earlier with relevant contextual information. Especially, we propose determining and refining this collaboration structure (e.g., selecting agents to participate at each step) using **LLM-based controllers**. The functionalities of these controllers are detailed in Section 3.1.

3. Methodology

In this section, we first detail the five dimensions we identified for optimizing multi-agent collaboration, covering both agent functionality and collaboration structure. Subsequently, we present our proposed algorithm for optimizing each dimension individually. Finally, we describe our algorithm for jointly optimizing multiple dimensions.

3.1. Five Dimensions for Multi-Agent Collaboration Optimization

As discussed in Section 2, agent functionality and collaboration structure are fundamental components of MAS. For the holistic optimization of such systems, we identify five key dimensions: two related to agent functionality and three concerning the collaboration structure, as detailed below:

- **Optimizing existing agents (Fun-1).** This dimension focuses on refining an existing agent within the current MAS. Specifically, the goal is to optimize an agent's functionality through components such as instruction prompts, few-shot examples, tool-use policies, and memory management, thereby improving its task-specific performance.
- **Optimizing construction of new agents (Fun-2).** This dimension addresses the creation of new agents. Specifically, given the task context and existing MAS configuration, the objective is to generate and optimize the functional components like the instruction prompt used to power a new LLM-based agent. This newly constructed agent will then be integrated into the existing MAS, targeting to enhance the overall collaborative capability in completing the given task.
- **Optimizing candidate agent selection (Str-1).** This dimension involves selecting suitable candidate agents from all available agents given the specific task before the collaboration. Specifically, the objective is to optimize an LLM-based controller. This controller is instructed to identify the most beneficial subset of agents for the multi-agent collaboration process, a decision informed by the provided task context and functionalities of existing agents.
- **Optimizing dynamic agent participation (Str-2).** This dimension concerns the dynamic selection of agents for participation at individual steps of the multi-step collaboration. Specifically, the objective is to optimize an LLM-based controller to choose the most suitable agents from the candidate team for participation in the current collaboration step. In contrast to Str-1, this controller additionally incorporates output solutions from previous agents as contextual input, enabling the dynamic selection of agents anticipated to contribute most effectively and efficiently at the current step of multi-agent collaboration.
- **Optimizing agent communication flows (Str-3).** This dimension addresses the optimization of communication flows among agents. Specifically, the goal is to optimize an LLM-based controller to determine whether the output from one agent should serve as input context for another agent during collaboration. The controller, guided by the optimization process, makes these communication routing decisions based on the given task context and the involved agents' functionalities.

These five proposed dimensions are designed to comprehensively cover the fundamental optimizable aspects of multi-step multi-agent collaboration. We derive them by conceptualizing the collaboration process in MAS as information flow over a graph, where agents correspond to nodes and communication channels correspond to edges. Thus, the two functional dimensions optimize node capabilities, either improving existing agents or constructing new ones. And the three structural dimensions define graph construction, encompassing global and dynamic local agent selection, as well as inter-agent communication routes. Together, these five dimensions comprehensively capture the essential aspects required to optimize the information-flow graph of a multi-agent system. While additional task-specific factors like budget management and computational efficiency may arise in particular applications, our proposed dimensions highlight the core, generalizable elements of multi-agent collaboration, namely the nodes and edges of the collaboration graph. More illustrations of the underlying intuition, implementation details, and examples are given in Appendix B and Appendix E.2.

3.2. Optimization for a Single Dimension

We first propose a unified algorithm designed for optimizing a single dimension. This algorithm employs two core LLM-powered actors: the Semantic Initializer and the Contrastive Comparator. Notably, our algorithm is generally applicable to any of the five dimensions identified before, requiring only minor adaptations to the contextual information supplied to the Semantic Initializer and Contrastive Comparator. The overall framework is shown in Figure 1.

Initialization of Collection. The first step involves generating an initial collection of agents or controllers corresponding to the dimension being optimized. Specifically, we utilize an LLM-powered actor named the Semantic Initializer to accomplish this. For each of the five dimensions, we

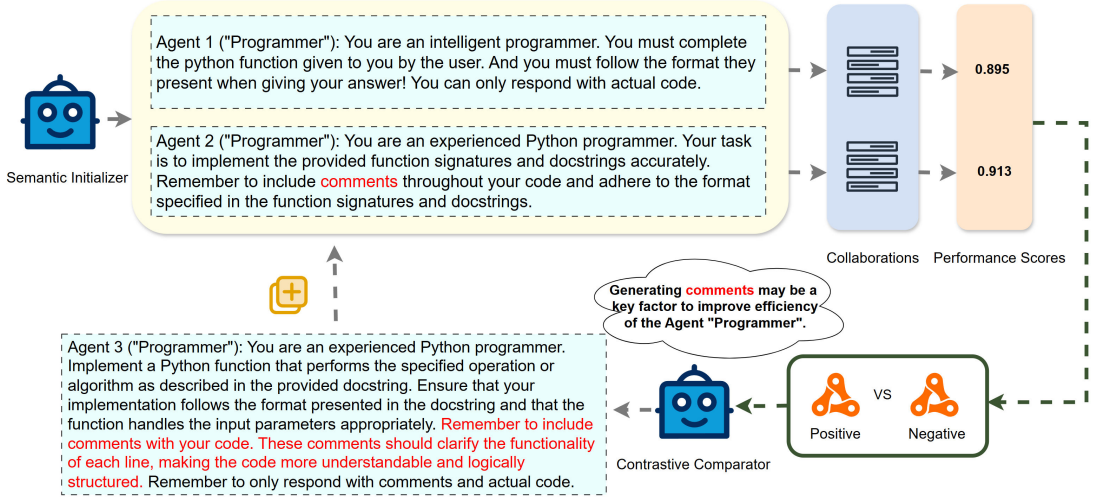

Figure 2. An example of optimization of a single dimension of OMAC.

instruct the Semantic Initializer by first describing essential contextual information regarding the existing MAS configuration and the given task. Next, we specify the expected functionality of the generations according to the dimension being optimized: either to power an agent (Fun-1 and Fun-2) or to guide the controller managing the collaboration structure (Str-1 to Str-3). Then, we provide a one-shot example and indicate the number of initial agents/controllers to be generated. If the optimized agents or controllers operate under a few-shot learning setting, the Semantic Initializer can also be instructed to generate few-shot examples using the same procedure.

The rationale behind this design is to leverage the knowledge and reasoning capabilities of LLMs to systematically explore the semantic space for instructing an agent or controller. This exploration adheres to specified functionality corresponding to the dimension being optimized, while simultaneously introducing controlled variations in focal emphasis and implementation details. Such variations promote diversity and inject stochasticity into the initial collection, which is utilized by subsequent optimization with contrastive reasoning.

Evaluation and Sampling. After obtaining the initial collection of agents or controllers to be optimized, we first evaluate their performance by integrating each one into the MAS with other existing agents/controllers and executing the collaboration process on the training data. Performance scores are then calculated based on the MAS's overall performance associated with each agent/controller in the initial collection. Subsequently, we sample a positive-negative pair of agents/controllers based on these performance scores. Specifically, we define two thresholds, h and l, as upper and lower bounds for selecting the positive and negative ones. Given the current collection of size n, the top $\lfloor n \times h \rfloor$ performing ones are classified as positive, whereas the bottom $\lfloor n \times l \rfloor$ are classified as negative. A positive-negative pair is then randomly sampled from these two groups. The rationale behind sampling within thresholds is to diversify the positive-negative pairs obtained in each iteration, thereby enhancing the generality and robustness of subsequent contrastive reasoning.

Contrastive Reasoning for Optimization. Once we get the positive-negative pair, we feed it into the Contrastive Comparator. This LLM-powered comparator is tasked to carefully compare this pair of agents/controllers and reason the underlying factors for their performance gap. Then, it is instructed to generate a new agent/controller (i.e., its instruction prompt, few-shot examples, etc.) that is expected to perform even better than the positive example. Similarly, information regarding the given task and the dimension being optimized is provided as the context for the reasoning. After that, this newly generated agent/controller is incorporated into the initial collection and evaluated through the collaboration process. The cycle of evaluation, sampling, contrastive comparison and refinement is then repeated until reaching the predefined maximum number of iterations.

The rationale here is to leverage the advanced reasoning capabilities of LLMs to analyze the performance difference by contrasting the positive-negative pair with the given task context. This performance gap constitutes a supervised signal derived from evaluations on the training data. By reasoning about this gap, the LLM can refine the corresponding agent/controller design, aiming to enhance or amplify factors correlated with positive performance while mitigating or removing factors associated with negative performance.

Demonstration Example. Figure 2 illustrates an example of optimizing an existing agent functioning as a "Program-

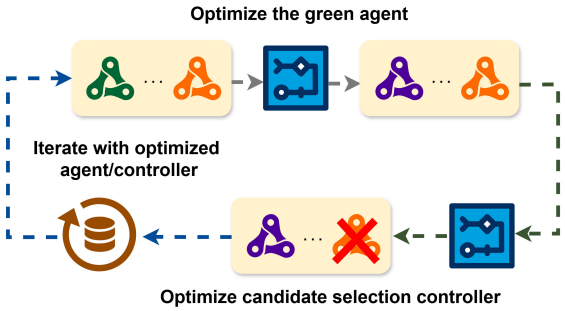

Figure 3. OMAC workflow for multiple dimensions.

mer" (Fun-1) using OMAC. The Semantic Initializer first generates two prompts, both designed to instruct an agent with the role as a Programmer for the code generation task. The primary difference is that the second one explicitly requires including comments throughout the code. Each agent is then evaluated by the overall performance of the MAS on the training data. Subsequently, a positive-negative pair is sampled and provided to the Contrastive Comparator. By analyzing the performance difference, the Contrastive Comparator identifies code commenting as a key factor enhancing the programmer agent's performance. Consequently, it generates a new prompt emphasizing the importance of comments and elaborating on their purpose and proper usage. Ideally, this refined prompt can guide the programmer agent to produce more accurate and logical code, further improving the overall performance of MAS.

3.3. Optimization for Multiple Dimensions

Beyond the single dimension optimization, we further propose an algorithm for jointly optimizing multiple dimensions. Specifically, our method iteratively optimizes each dimension individually while keeping the other dimensions fixed. For example, to jointly optimize an existing agent (Fun-1) and the controller to select candidate agents (Str-1), we first execute the single dimension optimization described in Section 3.2 for Fun-1. After that, we retain the agent from the agent collection exhibiting the highest performance score and subsequently optimize the controller responsible for candidate selection (Str-1). After deriving the optimized controller, we repeat this iterative process until predefined termination conditions (e.g., reaching a maximum number of iterations) are satisfied. Figure 3 illustrates this process.

The motivation of iterative optimization is to maintain the effectiveness of contrastive reasoning. Specifically, by limiting variations within positive-negative pairs to a single dimension at a time, we ensure consistency across other factors. Consequently, the Contrastive Comparator can clearly identify the reasons for performance differences, thus avoiding complexities arising from multiple interacting variables simultaneously affecting overall performance.

3.4. Inference and Computation Efficiency

Inference. After optimizing with either single dimension or multiple dimensions, we select the agent(s) and/or controller(s) configuration that yielded the highest performance on the training set. These optimized agents/controllers are then utilized within the MAS to conduct inference and evaluation on the test set.

Computational Efficiency. In Section 3.3, we introduced our algorithm for iteratively optimizing multiple dimensions jointly. However, iterative optimization may result in substantial computational demands due to the exponential growth of dimension combinations. For instance, mutually optimizing two dimensions over two iterations results in four times the computational cost compared to optimizing a single dimension. To mitigate this, we propose selectively incorporating only those dimensions that individually demonstrate the most significant performance improvements into the iterative joint optimization process. We validate the effectiveness of this strategy in our experiments (4.2). We further discuss OMAC's overall training and inference computational efficiency in detail in Appendix C.2.4.

4. Experiments

Tasks and Datasets. We primarily evaluate OMAC across three task domains: code generation, general reasoning, and arithmetic reasoning. For code generation, we utilize the HumanEval benchmark (Chen et al., 2021), which contains human-authored function-level code completions accompanied by unit tests. We employ Pass@1 as the evaluation metric, representing the proportion of generated code solutions that successfully pass the unit tests. For general reasoning tasks, we leverage the MMLU dataset (Hendrycks et al., 2021a), which comprises multiple-choice questions across humanities, social sciences, hard sciences, and other fields. We measure performance using answer accuracy. For arithmetic reasoning, we use the MATH dataset (Hendrycks et al., 2021b), encompassing mathematical problems spanning seven subareas, again employing accuracy as our evaluation metric. Additional details on tasks and datasets can be found in Appendix C.1. Besides these classic benchmarks, we further conducted experiments on more complex benchmarks, MBPP (Austin et al., 2021) on crowd-sourced Python programming tasks, and GAIA (Mialon et al., 2024) with hard real-world open questions. Due to the space limitation, we report relevant results on MBPP and GAIA in Appendix C.2.6.

Baselines. As prior studies typically adopt different agent functionalities and collaboration structures tailored to specific applications, we incorporate distinct baselines for each task domain. Specifically, we compare against single-agent methods, multi-agent methods with fixed configurations,

Table 1. Performance on code generation tasks with single-dimension optimization.

Method	Baselines					OMAC (Structural Dimension)		
	CodeT	AgentVerse	ADAS	AFlow	DyLAN	Str-1	Str-2	Str-3
Pass@1	67.50±1.68	78.29±2.34	75.61±2.06	85.63±2.10	85.74±2.83	**86.76**±1.22	**86.92**±2.27	**87.55**±2.46

Method	OMAC (Functional Dimension)							
	Fun-1.1	Fun-1.2	Fun-1.3	Fun-1.4	Fun-1.5	Fun-1.6	Fun-1.7	Fun-2
Pass@1	**88.39**±2.54	**86.31**±2.21	**88.87**±1.36	**89.25**±1.30	**88.74**±2.67	**88.39**±1.22	**88.34**±1.42	**86.77**±2.43

Table 2. Performance on general reasoning tasks with single-dimension optimization.

Method	Baselines					OMAC (Structural Dimension)		
	SE	LLM Debate	ADAS	AFlow	DyLAN	Str-1	Str-2	Str-3
Accuracy	65.76±2.31	68.74±2.67	69.02±2.44	70.06±2.57	69.42±2.16	**73.14**±2.24	**72.06**±1.96	**73.18**±1.47

Method	OMAC (Functional Dimension)							
	Fun-1.1	Fun-1.2	Fun-1.3	Fun-1.4	Fun-1.5	Fun-1.6	Fun-1.7	Fun-2
Accuracy	**72.33**±2.47	**73.23**±1.83	**72.06**±2.41	**74.22**±2.22	**73.15**±2.86	**72.07**±2.92	**71.83**±2.74	**71.02**±2.57

and latest MAS optimization methods including DyLAN (Liu et al., 2024), ADAS (Hu et al., 2025), and AFlow (Zhang et al., 2025c). Specifically, DyLAN and ADAS are prompt evolution–based baselines, while AFlow is a MCTS-based, supervised optimization method.

For code generation tasks, we select CodeT (Chen et al., 2023) as the single-agent baseline and AgentVerse (Chen et al., 2024a) as the multi-agent approach. For general reasoning and arithmetic reasoning tasks, we utilize a single agent directly generating answers as the single-agent baseline, denoted as Single Execution (SE). We also include the Self-Consistency (SC) (Wang et al., 2023b) for comparison on arithmetic reasoning. For multi-agent baseline, we compare LLM Debate (Du et al., 2023). All baseline approaches retain their original configurations and/or optimization procedure to ensure fair comparisons.

OMAC Setup. OMAC is designed to optimize the functionality and collaboration structure of an existing MAS. As such, we adopt the agent designs and collaboration structures from DyLAN (Liu et al., 2024) as the default configuration for OMAC across all datasets. Specifically, the default MAS includes 7 agents for code generation, 7 agents for general reasoning, and 4 agents for arithmetic reasoning tasks. Names and functionalities of these agents are detailed in Appendix C.1. The default collaboration structure is "fully-connected", meaning all existing agents participate in each step, and each agent in the current step receives all outputs from agents in the previous step as input. We utilize the same evaluation metrics on each dataset described above to construct positive-negative pairs, setting thresholds $l = h = 0.5$ consistently. Regarding other hyperparameters, the Semantic Initializer generates an initial collection of size 3, and we set the maximum number of contrastive reasoning iterations to 3. For fair comparisons, we employ `gpt-3.5-turbo-1106` with temperature of 0.8 as the base LLM across all baselines and our OMAC. Experiments with other base LLM are reported in Appendix C.2.5. All experiments are repeated three times, and the mean and standard deviation are reported. More implementation details are provided in Appendix C.1.

4.1. Single-Dimension Optimization Results

We first employ OMAC to individually optimize each of the five dimensions defined in Section 3.1. Specifically, for Fun-1, we optimize each existing agent in the default MAS separately, leading to sub-dimensions denoted as Fun-1.1 to Fun-1.7 for 7 agents in general reasoning and code generation tasks. For arithmetic reasoning tasks, we follow DyLAN to employ four agents sharing the same prompts and few-shot examples; hence, we optimize either the instruction prompts or few-shot examples for all of them, resulting in sub-dimensions Fun-1.1 and Fun-1.2. Table 2, Table 1, and Table 3 present results on each benchmark respectively.

The results indicate that optimizing each of the five dimensions individually leads to significant performance improvements across all three tasks in most cases. Note that the average relative optimization gains achieved by OMAC (3.6%, 2.8%, and 4.9% across the three benchmarks per dimension) are significant and comparable to those reported by recent MAS optimization studies, such as DyLAN (Liu et al., 2024) under similar experimental conditions. These results validate our delineation of the five optimization dimensions

Table 3. Performance on arithmetic reasoning tasks with single-dimension optimization.

Method	Baselines					
	SE	SC	LLM Debate	ADAS	AFlow	DyLAN
Accuracy(%)	28.72±1.75	29.14±2.07	29.42±2.33	28.94±2.04	32.49±2.62	32.35±1.94

Method	OMAC (Structural Dimension)			OMAC (Functional Dimension)		
	Str-1	Str-2	Str-3	Fun-1.1	Fun-1.2	Fun-2
Accuracy(%)	**33.34**±1.45	**33.41**±1.37	**33.70**±1.62	**35.17**±1.96	**34.91**±2.03	**33.95**±1.30

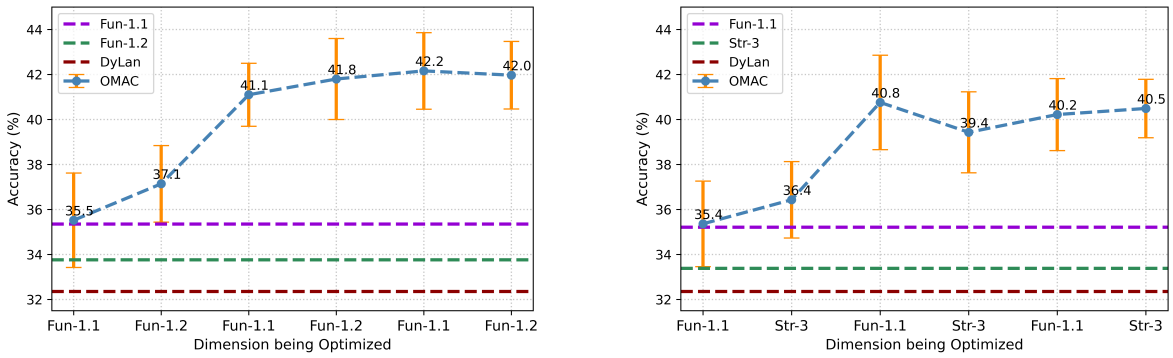

Figure 4. Performance during iterative optimization for multiple dimensions on arithmetic reasoning tasks. The X-axis represents the dimensions undergoing three iterations of optimization, while each point indicates the MAS's performance with the optimized dimensions on the test set.

for multi-agent collaboration and confirm the effectiveness of our single-dimension optimization algorithm. Furthermore, comparisons between single-agent and multi-agent methods highlight the advantages of leveraging multiple agents for complex tasks. Finally, while both DyLAN and OMAC achieve improvements through structural optimization, our approach demonstrates superior efficacy by utilizing supervised signals derived from training data evaluations. More experiments and results examining the effects of hyper-parameters, such as the size of the initial collection, the maximum number of iterations, and the sampling thresholds, are presented in Appendix C.2.1.

4.2. Multi-Dimension Optimization Results

We further experiment on jointly optimizing multiple dimensions with OMAC. Following the strategy outlined in Section 3.4, we selectively incorporate the dimensions exhibiting the most substantial individual performance improvements into our multi-dimensional optimization process. Specifically, we experiment by jointly optimizing the two best-performing dimensions by individual optimization (which, across all tasks, correspond to the two functional dimensions), as well as pairing the best functional dimension with the best structural dimension. These selected dimensions are then iteratively optimized following the procedure illustrated in Section 3.3, repeating for three iterations.

Figure 4 presents the experimental results on arithmetic reasoning tasks. Similar trends were observed in code generation and general reasoning tasks, where the results are provided in Appendix C.2.3. The figures clearly illustrate that iterative optimization of multiple dimensions **yields substantially larger performance improvements compared to optimizing only a single dimension** (e.g, improve from 2.9% to 9.6% in Figure 4). Furthermore, jointly optimizing dimensions that individually demonstrate the most significant improvements consistently yields greater benefits compared to optimizing suboptimal dimension pairs. This supports the effectiveness of our dimension-selection strategy for mitigating the computational overhead with multi-dimensional optimization. Experiments validating the iterative optimization design are reported in Appendix C.2.3. The results show that simultaneously optimizing multiple dimensions by asking the Contrastive Comparator to reason over multiple varying factors at once leads to significantly smaller performance gains and higher variance for OMAC.

4.3. Ablation Study

We further conduct experiments to validate the two actors of our optimization algorithm: Semantic Initializer and Contrastive Comparator. Specifically, we introduce an ablation model, OMAC-C, which excludes the Contrastive Comparator from the optimization pipeline described in Section 3.2. Thus, OMAC-C comprises of only the Semantic Initializer,

Table 4. Accuracy (%) of OMAC-C and OMAC on arithmetic reasoning task.

Method	Str-1	Str-2	Str-3	Fun-1.1	Fun-1.2	Fun-2
OMAC-C	32.64$\pm$1.98	32.67$\pm$2.10	32.76$\pm$2.31	34.20$\pm$2.87	33.69$\pm$2.32	32.71$\pm$2.03
OMAC	33.34$\pm$1.45	33.41$\pm$1.37	33.70$\pm$1.62	35.17$\pm$1.96	34.91$\pm$2.03	33.95$\pm$1.30

which generates an initial collection of agents or controllers. Each generated agent/controller is subsequently evaluated through the MAS collaboration process on the training set, after which the agent or controller achieving the highest performance score is selected for evaluation on the test set.

Table 4 presents the results on arithmetic reasoning tasks. The comparison between OMAC-C and OMAC clearly demonstrates the significant advantage provided by the Contrastive Comparator, which leverages contrastive reasoning to further optimize the agents or controllers generated by the Semantic Initializer. Nevertheless, OMAC-C consistently outperforms the strongest baseline, DyLAN, across all optimization dimensions. It indicates that even solely exploring the semantic space via LLM-based initialization for the agents or controllers in MAS contributes meaningful performance improvements. Additional results for the other two tasks are provided in Appendix C.2.2.

4.4. Computation Cost

In general, OMAC significantly reduces inference-time computational cost relative to baselines, benefiting from its dynamic agent selection and communication flow optimization. Since inference cost directly impacts both scalability and commercial competitiveness, we identify this as an important practical advantage of OMAC. During training, we further investigated the trade-off between training data size and computational cost, yielding valuable insights into OMAC's training efficiency. In particular, we explored a simple yet effective strategy: evaluating candidate agents or controllers on randomly sampled subsets of the training data rather than the full training set. Our findings show that this strategy provides a practical way to balance computational efficiency and evaluation robustness during OMAC optimization. For completeness, we present the detailed results and discussions in Appendix C.2.4.

5. Related Work

Multi-Agent Systems. Multi-Agent Systems that leverage multiple LLM-based agents communicating and collaborating, are increasingly employed to tackle complex, multi-step challenges (Tran et al., 2025). Existing research and industrial applications have demonstrated the effectiveness and robustness of these systems across diverse domains, such as code generation (Barbarroxa et al., 2024), reasoning (Du et al., 2023), question answering (Das et al., 2023), and decision making (Sun et al., 2024). Two critical factors in designing MAS are agent composition and the collaboration structure (Tran et al., 2025). We categorize these factors as relating to the functional and structural properties of MAS.

Agent Construction in MAS. The construction of agents directly determines the system's overall functional capabilities. Most existing works on MAS employ either manually designed prompts or prompts generated by LLMs to construct agents tailored to specific application scenarios. For example, Das et al. (2023) constructed agent teams with specific roles using hand-crafted instruction prompts for software development. Wang et al. (2023c) utilized LLMs to generate role-specific prompts for agents in response to task-specific queries. However, both manual prompt design and LLM-driven agent generation rely heavily on human prior knowledge, requiring empirical verification through trial-and-error. Although previous studies have explored agent optimization through fine-tuning (Li et al., 2023b) or prompt-tuning techniques (Khattab et al., 2023), these methods do not explicitly address the optimization of agents within MAS involving multi-step collaborative processes.

Collaboration Structure in MAS. Another crucial aspect of MAS is the collaboration structure, which defines how candidate agents collaborate and communicate to ultimately resolve the given task. Existing research has proposed various structures, such as centralized (Jiang et al., 2023), decentralized (Liang et al., 2023), and hierarchical (Hao et al., 2023) approaches. However, these structures are typically manually designed for specific task categories and remain static throughout the collaboration process. A recent work, DyLAN (Liu et al., 2024), represents an initial effort in dynamically optimizing the collaboration structure within MAS. However, its focus is limited to optimizing agent team composition, relying on an unsupervised metric called "Agent Importance Score" computed using heuristic rules combined with LLM-based judgments. In contrast, OMAC comprehensively examines both functional and structural optimization of MAS in a joint supervised manner, addressing more fine-grained optimization of collaboration structures.

Multi-Agent Collaboration Optimization. Existing agentic optimization methods can be broadly divided into gradient-based and non-gradient-based approaches. Gradient-based methods, such as supervised fine-tuning (Li et al., 2023b) and prompt tuning (Khattab et al., 2023), mainly focus on single-agent optimization and do not naturally extend to MAS settings that require multi-step col-

laboration among role-specialized agents. Although recent RL-based methods have been explored for agentic systems (Shao et al., 2024; Qian et al., 2025) and MAS optimization (Liu et al., 2025; Chen et al., 2025), they are often limited to simplified single-interaction or shared-policy scenarios. Non-gradient-based methods, including ADAS (Hu et al., 2025), AFlow (Zhang et al., 2025c), G-Designer (Zhang et al., 2025b), and MaAS (Zhang et al., 2025a), further advance MAS optimization through prompt evolution, MCTS, or architectural search, but typically address only agent functionality or collaboration structure. In contrast, OMAC provides a unified framework that jointly optimizes both functional and structural dimensions of multi-step, role-specialized MAS through direct contrastive reasoning with supervised signals.

We've included expanded discussion of related works on MAS optimization in Appendix A.

6. Conclusion

In this study, we introduce OMAC, a unified framework for optimizing LLM-based multi-agent systems in multi-step collaboration. We formalize five key optimization dimensions addressing both agent functionality and collaboration structure. Based on them, we develop a general algorithm for individually optimizing each dimension. The algorithm leverages two LLM-powered actors to explore diverse semantic possibilities for instructing agents or controllers, and to exploit supervised contrastive pairs for refining MAS through contrastive reasoning. Furthermore, we propose an iterative algorithm for jointly optimizing multiple dimensions. Extensive experiments demonstrate the effectiveness and superiority of our optimization framework.

Acknowledgment

We would like to thank Kamalika Das from Intuit AI Research for her valuable insights and suggestions. This work was supported by the Intuit University Collaboration Program. The authors acknowledge the Texas Advanced Computing Center (TACC) at The University of Texas at Austin for providing computational resources that have contributed to the research results reported within this paper.

Impact Statement

This paper presents work whose goal is to advance the field of machine learning. There are many potential societal consequences of our work, none of which we feel must be specifically highlighted here.

References

Austin, J., Odena, A., Nye, M., Bosma, M., Michalewski, H., Dohan, D., Jiang, E., Cai, C., Terry, M., Le, Q., et al. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*, 2021.

Barbarroxa, R., Ribeiro, B., Gomes, L., and Vale, Z. Benchmarking autogen with different large language models. In *2024 IEEE Conference on Artificial Intelligence (CAI)*, pp. 263–264. IEEE, 2024.

Chen, B., Zhang, F., Nguyen, A., Zan, D., Lin, Z., Lou, J.-G., and Chen, W. Codet: Code generation with generated tests. In *Proceedings of The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=ktrw68Cmu9c.

Chen, G., Yang, S., Li, C., Liu, W., Luan, J., and Xu, Z. Heterogeneous group-based reinforcement learning for llm-based multi-agent systems. *arXiv preprint arXiv:2506.02718*, 2025.

Chen, M., Tworek, J., Jun, H., Yuan, Q., Ponde de Oliveira Pinto, H., Kaplan, J., Edwards, H., Burda, Y., Joseph, N., Brockman, G., Ray, A., Puri, R., Krueger, G., Petrov, M., Khlaaf, H., Sastry, G., Mishkin, P., Chan, B., Gray, S., Ryder, N., Pavlov, M., Power, A., Kaiser, L., Bavarian, M., Winter, C., Tillet, P., Petroski Such, F., Cummings, D., Plappert, M., Chantzis, F., Barnes, E., Herbert-Voss, A., Hebgen Guss, W., Nichol, A., Paino, A., Tezak, N., Tang, J., Babuschkin, I., Balaji, S., Jain, S., Saunders, W., Hesse, C., Carr, A. N., Leike, J., Achiam, J., Misra, V., Morikawa, E., Radford, A., Knight, M., Brundage, M., Murati, M., Mayer, K., Welinder, P., McGrew, B., Amodei, D., McCandlish, S., Sutskever, I., and Zaremba, W. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

Chen, W., Su, Y., Zuo, J., Yang, C., Yuan, C., Chan, C.-M., Yu, H., Lu, Y., Hung, Y.-H., Qian, C., Qin, Y., Cong, X., Xie, R., Liu, Z., Sun, M., and Zhou, J. Agentverse: Facilitating multi-agent collaboration and exploring emergent behaviors. In *The Twelfth International Conference on Learning Representations*, 2024a. URL https://openreview.net/forum?id=EHg5GDnyq1.

Chen, Y., Arkin, J., Zhang, Y., Roy, N., and Fan, C. Scalable multi-robot collaboration with large language models: Centralized or decentralized systems? In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 4311–4317. IEEE, 2024b.

Das, A., Chen, S.-C., Shyu, M.-L., and Sadiq, S. Enabling synergistic knowledge sharing and reasoning in large language models with collaborative multi-agents. In *2023*

IEEE 9th International Conference on Collaboration and Internet Computing (CIC), pp. 92–98. IEEE, 2023.

Deng, Y., Zhang, W., Lam, W., Ng, S.-K., and Chua, T.-S. Plug-and-play policy planner for large language model powered dialogue agents. *arXiv preprint arXiv:2311.00262*, 2023.

Dong, Y., Jiang, X., Jin, Z., and Li, G. Self-collaboration Code Generation via ChatGPT. *arXiv preprint arXiv:2304.07590*, 2023.

Du, Y., Li, S., Torralba, A., Tenenbaum, J. B., and Mordatch, I. Improving factuality and reasoning in language models through multiagent debate. *arXiv preprint arXiv:2305.14325*, 2023.

Hao, R., Hu, L., Qi, W., Wu, Q., Zhang, Y., and Nie, L. Chatllm network: More brains, more intelligence. *arXiv preprint arXiv:2304.12998*, 2023.

Hendrycks, D., Burns, C., Basart, S., Zou, A., Mazeika, M., Song, D., and Steinhardt, J. Measuring massive multitask language understanding. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2021a.

Hendrycks, D., Burns, C., Kadavath, S., Arora, A., Basart, S., Tang, E., Song, D., and Steinhardt, J. Measuring mathematical problem solving with the math dataset. In *Proceedings of Thirty-fifth Conference on Neural Information Processing Systems*, 2021b.

Hu, S., Lu, C., and Clune, J. Automated design of agentic systems. *Proceedings of the International Conference on Learning Representations (ICLR)*, 2025.

Huang, D., Zhang, J. M., Luck, M., Bu, Q., Qing, Y., and Cui, H. Agentcoder: Multi-agent-based code generation with iterative testing and optimisation. *arXiv preprint arXiv:2312.13010*, 2023.

Huang, Y., Shi, J., Li, Y., Fan, C., Wu, S., Zhang, Q., Liu, Y., Zhou, P., Wan, Y., Gong, N. Z., and Sun, L. Metatool benchmark for large language models: Deciding whether to use tools and which to use. *ICLR*, 2024.

Jiang, D., Ren, X., and Lin, B. Y. Llm-blender: Ensembling large language models with pairwise ranking and generative fusion. *arXiv preprint arXiv:2306.02561*, 2023.

Jimenez, C. E., Yang, J., Wettig, A., Yao, S., Pei, K., Press, O., and Narasimhan, K. Swe-bench: Can language models resolve real-world github issues? *arXiv preprint arXiv:2310.06770*, 2023.

Khattab, O., Santhanam, K., Li, X. L., Hall, D., Liang, P., Potts, C., and Zaharia, M. Demonstrate-search-predict: Composing retrieval and language models for knowledge-intensive nlp, 2023.

Li, G., Hammoud, H., Itani, H., Khizbullin, D., and Ghanem, B. Camel: Communicative agents for" mind" exploration of large language model society. *Advances in Neural Information Processing Systems*, 36:51991–52008, 2023a.

Li, M., Zhao, Y., Yu, B., Song, F., Li, H., Yu, H., Li, Z., Huang, F., and Li, Y. Api-bank: A comprehensive benchmark for tool-augmented llms. In *Proceedings of the 2023 conference on empirical methods in natural language processing*, pp. 3102–3116, 2023b.

Li, Y., Liu, S., Zheng, T., and Song, M. Parallelized planning-acting for efficient llm-based multi-agent systems. *arXiv preprint arXiv:2503.03505*, 2025.

Liang, T., He, Z., Jiao, W., Wang, X., Wang, Y., Wang, R., Yang, Y., Shi, S., and Tu, Z. Encouraging divergent thinking in large language models through multi-agent debate. *arXiv preprint arXiv:2305.19118*, 2023.

Liu, B., Guertler, L., Yu, S., Liu, Z., Qi, P., Balcells, D., Liu, M., Tan, C., Shi, W., Lin, M., et al. Spiral: Self-play on zero-sum games incentivizes reasoning via multi-agent multi-turn reinforcement learning. *arXiv preprint arXiv:2506.24119*, 2025.

Liu, X., Yu, H., Zhang, H., Xu, Y., Lei, X., Lai, H., Gu, Y., Ding, H., Men, K., Yang, K., Zhang, S., Deng, X., Zeng, A., Du, Z., Zhang, C., Shen, S., Zhang, T., Su, Y., Sun, H., Huang, M., Dong, Y., and Tang, J. Agentbench: Evaluating llms as agents. 2023.

Liu, Z., Zhang, Y., Li, P., Liu, Y., and Yang, D. A dynamic llm-powered agent network for task-oriented agent collaboration. In *First Conference on Language Modeling*, 2024.

Mialon, G., Fourrier, C., Wolf, T., LeCun, Y., and Scialom, T. Gaia: a benchmark for general ai assistants. In *International Conference on Learning Representations*, volume 2024, pp. 9025–9049, 2024.

Plummer, B. A., Wang, L., Cervantes, C. M., Caicedo, J. C., Hockenmaier, J., and Lazebnik, S. Flickr30k entities: Collecting region-to-phrase correspondences for richer image-to-sentence models. In *Proceedings of the IEEE international conference on computer vision*, pp. 2641–2649, 2015.

Qian, C., Cong, X., Liu, W., Yang, C., Chen, W., Su, Y., Dang, Y., Li, J., Xu, J., Li, D., Liu, Z., and Sun, M. Communicative agents for software development. *arXiv preprint arXiv:2307.07924*, 2023.

Qian, C., Acikgoz, E. C., He, Q., Wang, H., Chen, X., Hakkani-Tür, D., Tur, G., and Ji, H. Toolrl: Reward is all tool learning needs. *arXiv preprint arXiv:2504.13958*, 2025.

Shao, Z., Wang, P., Zhu, Q., Xu, R., Song, J., Bi, X., Zhang, H., Zhang, M., Li, Y., Wu, Y., et al. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.

Shinn, N., Cassano, F., Gopinath, A., Narasimhan, K., and Yao, S. Reflexion: Language agents with verbal reinforcement learning. *Advances in Neural Information Processing Systems*, 36:8634–8652, 2023.

Sun, C., Huang, S., and Pompili, D. Llm-based multi-agent reinforcement learning: Current and future directions. *arXiv preprint arXiv:2405.11106*, 2024.

Tran, K.-T., Dao, D., Nguyen, M.-D., Pham, Q.-V., O'Sullivan, B., and Nguyen, H. D. Multi-agent collaboration mechanisms: A survey of llms. *arXiv preprint arXiv:2501.06322*, 2025.

Wang, X., Li, C., Wang, Z., Bai, F., Luo, H., Zhang, J., Jojic, N., Xing, E. P., and Hu, Z. Promptagent: Strategic planning with language models enables expert-level prompt optimization. 2023a.

Wang, X., Wei, J., Schuurmans, D., Le, Q. V., Chi, E. H., Narang, S., Chowdhery, A., and Zhou, D. Self-consistency improves chain of thought reasoning in language models. In *The Eleventh International Conference on Learning Representations*, 2023b. URL https://openreview.net/forum?id=1PL1NIMMrw.

Wang, Z., Mao, S., Wu, W., Ge, T., Wei, F., and Ji, H. Unleashing cognitive synergy in large language models: A task-solving agent through multi-persona self-collaboration. *arXiv preprint arXiv:2307.05300*, 2023c.

Wu, S., Zhao, S., Huang, Q., Huang, K., Yasunaga, M., Cao, K., Ioannidis, V., Subbian, K., Leskovec, J., and Zou, J. Y. Avatar: Optimizing llm agents for tool usage via contrastive reasoning. *Advances in Neural Information Processing Systems*, 37:25981–26010, 2024.

Yao, S., Zhao, J., Yu, D., Du, N., Shafran, I., Narasimhan, K. R., and Cao, Y. React: Synergizing reasoning and acting in language models. In *The eleventh international conference on learning representations*, 2022.

Yao, S., Zhao, J., Yu, D., Du, N., Shafran, I., Narasimhan, K. R., and Cao, Y. React: Synergizing reasoning and acting in language models. In *Proceedings of The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=WE_vluYUL-X.

Zhang, G., Niu, L., Fang, J., Wang, K., Bai, L., and Wang, X. Multi-agent architecture search via agentic supernet. *arXiv preprint arXiv:2502.04180*, 2025a.

Zhang, G., Yue, Y., Sun, X., Wan, G., Yu, M., Fang, J., Wang, K., Chen, T., and Cheng, D. G-designer: Architecting multi-agent communication topologies via graph neural networks. *arXiv preprint arXiv:2410.11782*, 2025b.

Zhang, J., Xiang, J., Yu, Z., Teng, F., Chen, X., Chen, J., Zhuge, M., Cheng, X., Hong, S., Wang, J., et al. Aflow: Automating agentic workflow generation. 2025c.

A. Expanded Related Work

Gradient-Based Agentic Optimization Methods. As previously discussed, prior research has explored several gradient-based optimization methods for agent systems, such as supervised fine-tuning (Li et al., 2023b) and prompt-tuning techniques (Khattab et al., 2023). However, most of these approaches are designed for single-agent optimization and do not extend naturally to MAS that require multi-step collaborative interactions among multiple agents. To the best of our knowledge, research on gradient-based methods specifically targeting MAS optimization in multi-step collaboration settings remains very limited.

In scenarios involving multiple interacting agents, gradient-based optimization becomes particularly challenging due to the presence of numerous confounding variables and the difficulty of jointly performance optimization with multiple functional and structural components. For instance, while some works have explored the use of reinforcement learning for optimizing agentic systems, these methods typically focus on single-agent settings (Shao et al., 2024; Qian et al., 2025), which differ fundamentally from the multi-agent optimization problem addressed in our study. A few very recent works, such as Spiral (Liu et al., 2025) and MHGPO (Chen et al., 2025), have made initial attempts at applying RL to MAS optimization. However, their formulations are generally constrained to single-interaction and shared-policy scenarios, where all agents share the same control policy and perform only one round of communication or problem-solving. This setup is considerably simpler and less flexible than the setting studied in our work, which involves multi-step collaboration among role-specialized agents operating within dynamic interaction structures. Consequently, these approaches are not directly comparable to OMAC, which provides a general and extensible optimization framework for optimizing multi-step, role-specialized collaborations among multiple agents.

Non-Gradient-Based MAS Optimization Methods. A very recent work, ADAS (Hu et al., 2025), takes an initial step toward applying prompt evolution techniques combined with search tools to optimize MAS. However, ADAS focuses on agent functionality optimization, leaving the collaboration structure predefined and fixed. This contrasts with OMAC, which introduces a general framework capable of comprehensively optimizing both the functional and structural aspects of MAS. Beyond ADAS, a couple of recent studies have explored non-gradient-based optimization for MAS from other perspectives. For example, AFlow (Zhang et al., 2025c) leverages supervised signals to guide Monte Carlo Tree Search (MCTS) for discovering effective agentic workflows. In contrast, OMAC uses supervised signals directly for contrastive reasoning–based optimization, representing a distinct paradigm that avoids reliance on search-based methods. Similarly, other recent works such as G-Designer (Zhang et al., 2025b) and MaAS (Zhang et al., 2025a) focus on architectural search for identifying effective multi-agent structures, thereby addressing primarily the structural optimization dimension. For example, in the MaAS implementation, the authors directly adopt well-optimized and near-optimal agent configurations from prior work like AgentCoder (Huang et al., 2023) and ReAct (Yao et al., 2023) as fixed functional components, and only optimize the collaboration structure. OMAC, however, begins with simple and minimally engineered agent configurations and is designed to optimize both agent functionality and MAS structure. In summary, while these works have advanced non-gradient MAS optimization from various angles, they typically address only a single aspect, either functional or structural, or use supervised signals indirectly within evolutionary search algorithms like MCTS. By contrast, OMAC provides a unified framework that jointly optimizes both functionality and collaboration structure, employing direct contrastive reasoning with supervised signals rather than relying on evolutionary or search-based methods.

B. Optimization Dimension Details

We first articulate the intuition and motivation underlying the proposal of these five dimensions. As outlined in Section 3.1, they are deliberately designed to capture the fundamental aspects of MAS optimization. While task-dependent considerations such as computational efficiency may arise, our proposed dimensions highlight the core, generalizable elements of multi-agent collaboration, which constitute the central conceptual contribution of this work.

Specifically, we conceptualize the collaboration process in MAS as information flow over a graph, where agents correspond to nodes and communication channels to edges. Within this abstraction, the two functional dimensions focus on optimizing node-level capabilities, either by enhancing existing agents or constructing new ones. On the other hand, the three structural dimensions pertain to graph structure design: they govern the mechanisms of global and dynamic local agent selection, as well as the configuration of inter-agent communication pathways.

Taken together, these five dimensions form a comprehensive framework that addresses both the functional and structural foundations of MAS. By doing so, they provide principled coverage of the essential components required for optimizing the

multi-agent collaboration graph. Since the central theme of the paper is to establish a MAS optimization framework, we intentionally focus on defining the optimization dimensions and methodology rather than conducting a rigorous theoretical analysis of the optimization landscape or deriving formal guarantees of optimal configurations. We'd like to leave these explorations for future work. See Appendix D.6 for a brief discussion.

We next present more details of the rationale and implementation details of the five dimensions.

Fun-1: Optimizing existing agents. This dimension focuses on enhancing the functionality of an existing agent within a MAS. Specifically, we optimize the agent's instruction prompt and/or associated few-shot examples (in few-shot learning settings) following the optimization procedure detailed in Section 3.2.

When optimizing this dimension, the input to the Semantic Initializer includes the task description and the functional role of the agent. We also provide the original instruction prompt and/or associated few-shot examples of the agent being optimized as a one-shot example for the expected output. For the Contrastive Comparator, the input context consists of the task description, the functional illustration of the agent, and the sampled positive-negative pair. Both the Semantic Initializer and the Contrastive Comparator output newly generated prompts and/or few-shot examples to power the targeted agent.

Fun-2: Optimizing construction of new agents. This dimension aims to optimize the design and construction of a new agent, which is subsequently integrated into the existing MAS to enhance task performance. Specifically, we generate and optimize the instruction prompt and/or associated few-shot examples used to guide a new LLM-based agent which will be incorporated into existing MAS for collaboration.

During optimization, the Semantic Initializer receives the task description and the functional roles of all existing agents in the MAS as input context. We then provide a handcrafted instruction prompt as a one-shot example of the expected output from the Semantic Initializer. For the Contrastive Comparator, the input context includes the task description along with the sampled positive-negative pair. Both the Semantic Initializer and Contrastive Comparator generate the instruction prompts and/or associated few-shot examples powering the new agent.

Str-1: Optimizing candidate agent selection. This dimension focuses on optimizing an LLM-powered controller to select appropriate candidate agents from all available agents before collaboration begins. Specifically, we optimize the instruction prompt that guides the controller's selection process based on the provided task context and the functional roles of all available agents.

During optimization of it, the Semantic Initializer receives as input the task description and the expected functionality of the controller. Also, a handcrafted instruction prompt is given as a one-shot example. For the Contrastive Comparator, the input context consists of the task description, the controller's expected functionality, and the sampled positive-negative pair. Both the Semantic Initializer and Contrastive Comparator produce the instruction prompts for guiding the controller in candidate agent selection.

The rationale for optimizing this controller is to selectively incorporate only those agents most beneficial for resolving the given task, thus reducing harmful disturbance and enhancing the overall effectiveness and efficiency of the MAS.

Str-2: Optimizing dynamic agent participation. This dimension targets to optimize an LLM-powered controller that dynamically selects agents from the candidate pool for participation in the current collaboration step. Specifically, the controller's instruction prompt is optimized to enable it to choose the most suitable agents based on the task context, the functional roles of candidate agents, and the agents' outputs generated in previous steps.

During optimization for this dimension, the input to the Semantic Initializer includes the task description, the expected functionality of the controller, and a handcrafted instruction prompt as a one-shot example. Similarly, the input context for the Contrastive Comparator comprises the task description, the controller's intended functionality, and the sampled positive-negative pair. Both the Semantic Initializer and Contrastive Comparator generate prompts instructing the controller to dynamically select appropriate agents.

The rationale for optimizing this controller is to enable the selection of only those agents most relevant and beneficial for the current collaboration step, based on analysis of the task context, agent functionality, and agents' outputs from previous steps. For instance, an agent responsible for initial problem decomposition would ideally be selected by this controller only during the early stages of the collaboration.

Str-3: Optimizing agent communication flows. This dimension is designed to optimize an LLM-powered controller responsible for determining whether the output from one agent should be incorporated as input context for another agent

during collaboration. Specifically, we optimize the controller's instruction prompt to guide communication-structure decisions based on the given task context and the functional roles of candidate agents.

During the optimization procedure for this dimension, the input provided to the Semantic Initializer includes the task description, the expected functionality of the controller, and a handcrafted instruction prompt as a one-shot example. For the Contrastive Comparator, the input context comprises the task description, the controller's intended functionality, and the sampled positive-negative pair. Both the Semantic Initializer and the Contrastive Comparator generate prompts instructing the controller to manage communication flows between agents.

The rationale behind optimizing this controller is to route information selectively, ensuring that an agent receives input only from other agents whose outputs are directly relevant and beneficial to the recipient agent's task resolution. For example, in the code-generation task, the output from an agent serving as a "Tester," which generates and executes unit tests to evaluate previously generated code, should ideally be routed by the controller only to agents responsible for further refining the code further, rather than to another "Tester" agent.

C. Experiment Details and Additional Results

C.1. Experimental Setup

As outlined in Section 4, we inherit most of the experimental settings from the SOTA method DyLAN (Liu et al., 2024), as these configurations are standard and widely validated in existing literature for direct and fair comparison. Specifically, for all datasets, we partition the data into training, validation, and testing sets using a 3:1:1 ratio. We directly inherited most hyperparameters from DyLAN to maintain fair comparisons and, given limited computational resources, did not perform additional tuning on a validation set. The maximum token length is set to 2048 for code generation and arithmetic reasoning tasks, and 1024 for general reasoning tasks. The ranking and selection procedures for controllers optimized under Str-1 and Str-2 follow a listwise approach. To avoid positional bias, agent messages from the preceding collaboration step are randomly shuffled before being passed to agents in the subsequent step. Additionally, we employ an early-stopping mechanism to reduce unnecessary computational costs when agent outputs remain consistent across consecutive steps. Further details are available in the original DyLAN paper (Liu et al., 2024). For all baseline methods, we follow the implementation details and optimization configurations reported in the original papers. All experiments are repeated three times, and the mean and standard deviation are reported.

Experiments on general reasoning tasks. We randomly sample 168 multiple-choice questions from the MMLU dataset (Hendrycks et al., 2021a) and evenly split them into training and testing sets. The default MAS comprises seven agents with the following functional roles: "Economist", "Doctor", "Lawyer", "Mathematician", "Psychologist", "Programmer", "Historian". The default instruction prompts for these agents are adopted directly from DyLAN's implementation. Answers for the agents are extracted from the agent outputs by matching the final occurrence of "(X" or "(X)", where "X" denotes one of the choices A, B, C, or D. The final answer is determined by selecting the option that receives the highest number of votes from agents in the last step of collaboration. The maximum number of collaboration steps is set to four, with dynamic agent selection occurring at the third step. Performance is always measured by the average classification accuracy across all questions in the four categories.

Experiments on code generation tasks. We sample 180 function-level code completion tasks from the HumanEval benchmark (Chen et al., 2021) and evenly divide them into training and testing sets. Following DyLAN, the default MAS comprises four code-writing agents and four code-reviewing agents. Among code reviewers, three are optimizable while the fourth remains fixed as required by the method workflow. The code writers include "Python Assistant", "Algorithm Developer", "Computer Scientist", and "Programmer". The optimizable code reviewers are "Syntax Checker", "Unit Tester", and "Reflector". Solutions provided by code writers undergo a review process with a maximum of six rounds. Specifically, at time steps $t = 1, 3, 4, 6$, code writers generate solutions, while at $t = 2, 5$, code reviewers provide feedback. Dynamic agent selection occurs at the fourth step. The final code is randomly selected from the top five code completions across all agents that pass most tests from code reviewers. Performance evaluation is based on the average pass rate (Pass@1) of generated code across all code completion tasks.

Experiments on arithmetic reasoning tasks. We randomly sample 140 mathematical problems from the MATH dataset (Hendrycks et al., 2021b), evenly covering seven subareas: algebra, counting and probability, geometry, intermediate algebra, number theory, pre-algebra, and pre-calculus. Also, these samples are split evenly into training and testing sets. In DyLAN, the authors found that collaborating agents in different domains (e.g., algebra and geometry experts) do not make significant

improvement, therefore they adopt four agents with identical prompts and few-shot examples. We follow this setting for a fair comparison. The maximum number of collaboration steps is set to four, with dynamic agent selection taking place at the third step. We also follow DyLAN in using the answer extraction method described by (Hendrycks et al., 2021b). The average accuracy across all questions serves as the performance evaluation.

C.2. Additional Experimental Results

C.2.1. Sensitivity of Hyper-Parameters for Single-Dimension Optimization

Size of initial collection. We first evaluate the hyper-parameter sensitivity by varying the size of initial collection (denoted as z) generated by the Semantic Initializer, while keeping all other settings as default. Table 5 summarizes the results on the MATH dataset for arithmetic reasoning tasks. The results demonstrate that increasing the size of the initial collection generally leads to improved performance with OMAC. This is intuitively reasonable because allowing the Semantic Initializer to explore more diverse agent/controller designs increases the likelihood of obtaining higher-performing solutions. While a larger collection entails greater computational costs since each initialized agent/controller must be evaluated via the full multi-agent collaboration on the training data, we observe substantial improvements over the SOTA DyLAN even with a modest collection size of three.

Table 5. Accuracy (%) of OMAC on each dimension with different sizes of initial collection on arithmetic reasoning tasks.

Collection Size	Str-1	Str-2	Str-3	Fun-1.1	Fun-1.2	Fun-2
$z = 1$	32.71±1.63	32.75±1.98	32.80±2.32	33.19±2.34	32.98±2.82	32.80±2.14
$z = 2$	33.14±1.50	32.97±1.41	33.25±2.65	34.46±2.40	33.90±2.15	33.33±2.49
$z = 3$	33.34±1.45	33.41±1.37	33.70±1.62	35.17±1.96	34.91±2.03	33.95±1.30
$z = 4$	33.69±1.53	33.73±1.42	33.96±1.27	35.63±1.52	35.21±1.72	34.17±1.53
$z = 5$	33.92±1.23	33.96±0.96	34.25±1.27	35.89±2.14	35.67±1.42	34.51±1.31

Maximum number of contrasting iterations. We then conduct experiments to assess the impact of the maximum number of contrastive reasoning iterations (denoted as w) used by Contrastive Comparator. Table 6 presents the results on the MATH dataset. The results indicate a similar trend: increasing the number of iterations generally results in enhanced performance by consistently refining the agents/controllers through contrastive reasoning. Furthermore, setting the iteration count to three is sufficient to achieve significant performance improvements compared to DyLAN.

Table 6. Accuracy (%) of OMAC on each dimension with different maximum number of contrastive reasoning iterations on arithmetic reasoning tasks.

Number of Iterations	Str-1	Str-2	Str-3	Fun-1.1	Fun-1.2	Fun-2
$w = 1$	32.84±1.23	32.72±2.73	32.83±2.41	33.45±2.23	33.03±2.31	32.82±2.16
$w = 2$	33.02±2.44	32.94±2.14	33.07±2.61	34.12±2.24	34.12±2.60	33.43±2.57
$w = 3$	33.34±1.45	33.41±1.37	33.70±1.62	35.17±1.96	34.91±2.03	33.95±1.30
$w = 4$	33.85±1.23	34.15±1.14	34.41±1.52	35.74±1.12	35.26±0.67	34.34±0.93
$w = 5$	34.05±0.84	34.77±0.73	34.82±1.14	36.04±1.62	35.62±1.26	34.79±1.52

Sampling thresholds. Lastly, We evaluate the impact of varying sampling thresholds l and h, as described in Section 3.2. Table 6 presents the results for dimensions Str-1 and Fun-1.1 on the MATH dataset for arithmetic reasoning tasks. From these results, we observe that OMAC is robust to variations in the sampling thresholds, with performance fluctuations limited to within 1%. Additionally, we note a slight performance decrease when the gap between l and h widens. This may be attributed to imbalanced sampling of positive and negative examples, potentially leading to less accurate and fair reasoning by the Contrastive Comparator.

C.2.2. Additional Results of Ablation Study

Tables 8 and Table 9 summarize the results of the ablation study described in Section 4.3 on code generation and general reasoning tasks. The findings are consistent: the ablation model, OMAC-C, which removes the Contrastive Comparator from the optimization pipeline, performs significantly worse than the full OMAC framework. These results highlight the substantial advantage provided by the Contrastive Comparator's contrastive reasoning on supervised positive-negative pairs.

Table 7. Accuracy (%) of OMAC on Str-1 (left) and Fun-1.1 (right) with different combinations of sampling thresholds on arithmetic reasoning tasks.

	$l = 0.3$	$l = 0.4$	$l = 0.5$
$h = 0.3$	32.94	33.02	32.86
$h = 0.4$	32.90	33.18	33.10
$h = 0.5$	32.79	**33.41**	33.34

	$l = 0.3$	$l = 0.4$	$l = 0.5$
$h = 0.3$	34.92	34.95	34.90
$h = 0.4$	35.02	35.01	**35.19**
$h = 0.5$	34.87	35.10	35.17

Table 8. Pass@1 (%) of OMAC-C and OMAC with each optimization dimension on the code generation tasks.

Method	Str-1	Str-2	Str-3
OMAC-C	86.23±1.67	85.66±2.34	86.31±2.80
OMAC	86.76±1.22	86.92±2.27	87.55±2.46

Method	Fun-1.1	Fun-1.2	Fun-1.3	Fun-1.4	Fun-1.5	Fun-1.6	Fun-1.7	Fun-2
OMAC-C	86.74±2.96	85.92±2.42	86.46±2.21	87.86±1.88	87.11±3.56	87.26±2.04	86.97±1.95	86.01±2.86
OMAC	88.39±2.54	86.31±2.21	88.87±1.36	89.25±1.30	88.74±2.67	88.39±1.22	88.34±1.42	86.77±2.43

C.2.3. Additional Results of Multi-Dimension Optimization

Results on other tasks. We present experimental results of multi-dimension optimization using our OMAC on code generation tasks and general reasoning tasks in Figure 5 and Figure 6 respectively. The trends are similar to the results shown in Section 4.2, which demonstrate that iterative optimizing multiple dimensions can bring in **significant performance improvements compared to optimizing a single dimension**. Also, only optimizing dimensions that individually demonstrate the most significant improvements is an effective strategy to bring in significant performance improvement considering the computation resource constraints.

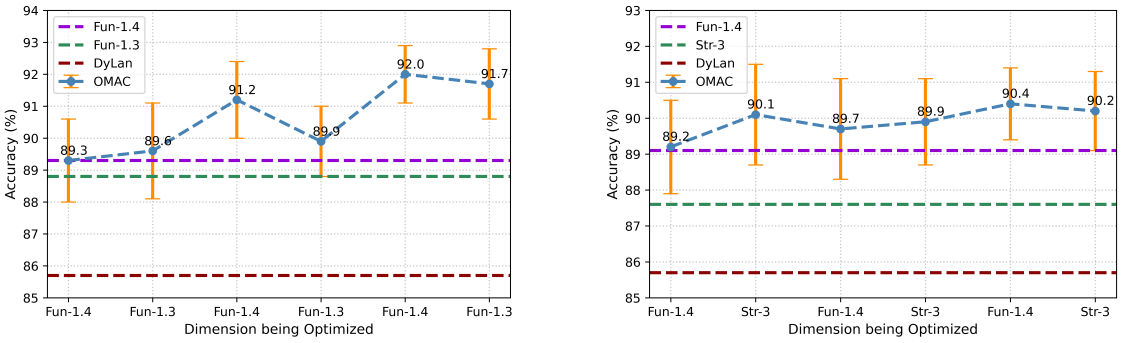

Figure 5. Performance during iterative optimization for multiple dimensions on code generation tasks. The X-axis represents the dimensions undergoing three iterations of optimization, while each point indicates the MAS's performance with the optimized dimensions on the test set. The error bar denotes the standard deviation.

Validation of iterative optimization. We further conduct experiments to validate our iterative optimization design when jointly optimizing multiple dimensions. Specifically, we propose to optimize one dimension at a time while keeping other dimensions fixed, thereby preserving the effectiveness of contrastive reasoning by limiting variability within positive-negative pairs. To verify this, we conduct experiments where two dimensions are simultaneously varied during optimization, meaning the Contrastive Comparator generates two agents/controllers based on the positive-negative pair in each iteration.

By comparing Figure 7 and Figure 4, it is evident that simultaneously optimizing multiple dimensions, which asks the Contrastive Comparator to reason over multiple variable factors at once, results in significantly reduced performance gains and larger variance for OMAC. These findings validate the rationale behind our iterative multi-dimensional optimization design.

Refine the following text: Also, as noted in the figures, some performance drop can be observed between adjacent optimization steps. We attribute to two reasons. First, when the MAS performance has largely converged, further exploration

Table 9. Accuracy (%) of OMAC-C and OMAC with each optimization dimension on the general reasoning tasks.

Method	Str-1	Str-2	Str-3
OMAC-C	71.13±1.94	69.86±1.77	70.71±1.75
OMAC	73.14±2.24	72.06±1.96	73.18±1.47

Method	Fun-1.1	Fun-1.2	Fun-1.3	Fun-1.4	Fun-1.5	Fun-1.6	Fun-1.7	Fun-2
OMAC-C	70.88±2.55	70.47±2.04	70.46±2.74	71.70±2.15	71.33±3.10	70.19±2.35	70.23±2.05	69.74±2.93
OMAC	72.33±2.47	73.23±1.83	72.06±2.41	74.22±2.22	73.15±2.86	72.07±2.92	71.83±2.74	71.02±2.57

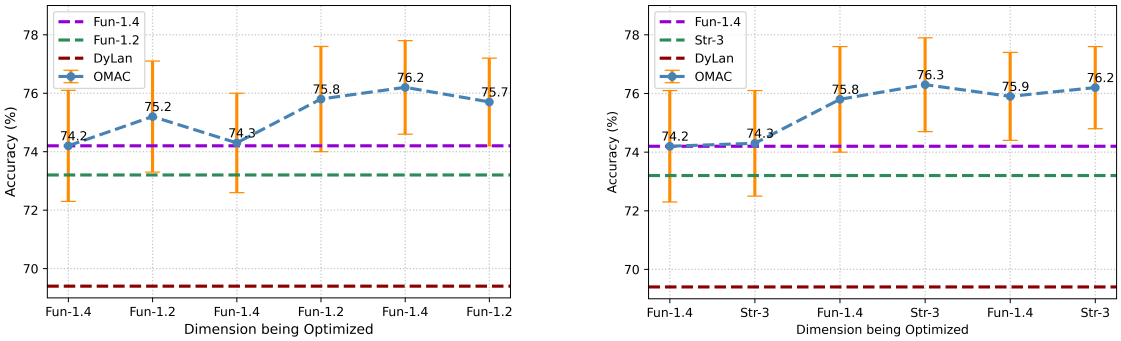

Figure 6. Performance during iterative optimization for multiple dimensions on general reasoning tasks. The X-axis represents the dimensions undergoing three iterations of optimization, while each point indicates the MAS's performance with the optimized dimensions on the test set. The error bar denotes the standard deviation.

around already near-optimal agent combinations may introduce noise and can lead to a drop in test performance. Second, as suggested by Figure 4 and the other result tables, the variance of MAS performance can be relatively large due to randomness amplified by the multi-step collaboration process across multiple agents. Since our evaluation uses a relatively small set and a limited number of repeated runs due to efficiency reasons, this variance may also contribute to the observed decrease in the empirical average performance.

C.2.4. Computation Cost

First, OMAC improves online inference efficiency, which is a key factor for the practical deployment of agent systems, through its effective structural control. Specifically, OMAC optimizes to dynamically select only the most important and valuable agents to participate in the collaboration (Str-1 and Str-2) and intelligently determines the input context by incorporating only relevant information (Str-3). These optimizations substantially reduce token consumption and API calls. We report the average number of API calls and token cost with GPT-3.5-Turbo required to solve a single problem on the code generation task (HumanEval benchmark) and the arithmetic reasoning task (MATH dataset), shown in Table 10 and Table 11. The results demonstrate that OMAC significantly reduces inference time computational cost by dynamically optimizing the collaboration structure, outperforming other multi-agent frameworks.

Table 10. Inference cost on code generation tasks.

Method	Baselines				OMAC			
	CodeT	AgentVerse	DyLAN	AFlow	Str-1	Str-2	Str-3	Str-2+Str-3
#API Calls	20.72	23.15	17.94	17.25	16.34	15.76	17.03	14.80
Cost ($)	0.1430	0.1958	0.1288	0.1134	0.1032	0.0973	0.0920	0.0883

For the cost on the training stage, as illustrated in Section 3.2, OMAC requires evaluating agents or controllers by executing the full collaborative process across the training dataset. While this approach ensures robust and representative performance evaluations, it can increase computational costs especially when the size of training data is large and the given tasks are complex. For example, OMAC requires around 1,400 API calls for training to optimize each dimension on the HumanEval

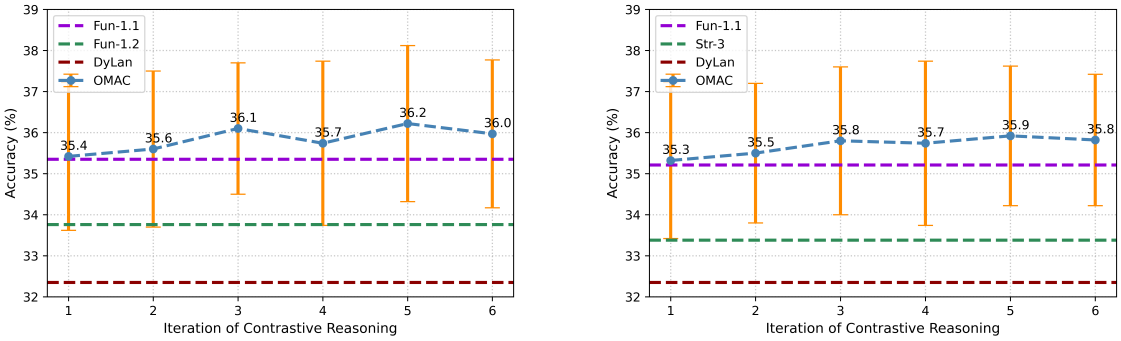

Figure 7. Performance of simultaneously optimizing multiple dimensions on arithmetic reasoning tasks. The X-axis denotes the iteration number of contrastive reasoning, and each point indicates the MAS performance achieved using the optimized dimensions generated by the Contrastive Comparator. The left figure illustrates results from jointly optimizing Fun-1.1 and Fun-1.2, while the right figure corresponds to jointly optimizing Fun-1.1 and Str-3. The error bar denotes the standard deviation.

Table 11. Inference cost on arithmetic reasoning tasks.

Method	Baselines				OMAC			
	LLM-Blender	LLM-Debate	DyLAN	AFlow	Str-1	Str-2	Str-3	Str-2+Str-3
#API Calls	7.12	8.26	7.30	7.72	6.85	6.23	7.12	5.74
Cost ($)	0.0458	0.0530	0.0446	0.0477	0.0369	0.0352	0.0337	0.0316

benchmark.

However, we'd like to highlight that several recent studies optimizing agent systems in a supervised and iterative manner for task-level performance, such as AVATAR (Wu et al., 2024) and PPDPP (Deng et al., 2023), also impose similar computational demands, often involving thousands of API calls during system optimization. For instance, AVATAR, which is designed to optimize only a single agent, requires over 3,000 API calls for just one training round on the FLICKR30K ENTITIES dataset (Plummer et al., 2015). Therefore, we regard these computational constraints as a common challenge across current agent system optimization methods, rather than a limitation unique to OMAC.

Nevertheless, balancing OMAC's effectiveness with computational efficiency remains a valuable and promising direction for the practical implementation of OMAC, particularly for academic researchers with limited computational resources. To investigate this, we propose a simple yet effective strategy: evaluating candidate agents or controllers on randomly sampled subsets of the training data. Although this may introduce variance, it provides a good starting point for exploring the trade-off between efficiency and robustness.

Specifically, we conducted experiments on the HumanEval benchmark, where only 50% of the training data was randomly sampled for each evaluation. This approach reduced the number of API calls per optimization by approximately 50%, lowering the total cost of a complete training run with GPT-3.5-turbo to around 5 dollars. Despite the reduced evaluation set, OMAC continued to outperform the strongest baseline, DyLAN, with only a minor decrease in performance compared to evaluations conducted on the full training set. These findings suggest that subset-based evaluation offers a practical and effective way to balance computational efficiency and evaluation robustness for optimization in OMAC.

Table 12. Training cost on code generation tasks with sampled training data.

Method	Baseline	OMAC										
	DyLAN	Fun-1.1	Fun-1.2	Fun-1.3	Fun-1.4	Fun-1.5	Fun-1.6	Fun-1.7	Fun-2	Str-1	Str-2	Str-3
Accuracy	85.74	88.27	86.22	88.62	89.13	88.51	88.20	88.23	86.56	86.43	86.79	87.47
Cost ($)	4.47	4.95	5.21	5.02	5.40	5.67	5.28	5.15	4.73	4.62	5.41	5.24

Last, we include a comprehensive comparison on the training cost, inference cost, and performance of OMAC with the main baselines including AgentVerse, DyLAN, and AFlow. The experiments are also conducted on the HumanEval benchmark, and we report the average cost and performance across all optimization dimensions of OMAC for comparison.

Table 13. Efficiency comparison between OMAC's average performance and other MAS baselines on code generation tasks.

Method	AgentVerse	DyLAN	AFlow	OMAC (avg.)
Training Cost ($)	-	**4.47**	8.24	5.15
Inference Cost ($)	0.1928	0.1288	0.1134	**0.0974**
Accuracy (%)	78.26	85.74	85.63	**87.68**

From Table 13, it's clear that OMAC achieves substantial improvements in both inference efficiency and performance than all compared baselines, with only a minor increase in training cost compared to DyLAN. These findings further confirm OMAC's strong cost–performance balance and practical applicability for real-world multi-agent systems.

C.2.5. Results with Different Base LLMs

In addition to using GPT-3.5-Turbo as the base model, we also conducted experiments with three other LLMs: GPT-4o-mini and two open-source models, DeepSeek-V2.5 and Qwen-2.5-72B. The corresponding results are reported separately in Table 14, Table 15, and Table 16. The findings consistently demonstrate OMAC's robustness and its sustained performance advantages over baseline methods across all optimization dimensions. These results further highlight the cross-model transferability of the optimization capabilities provided by the proposed OMAC framework.

Table 14. Performance on code generation tasks with GPT-4o-mini.

Method	Baselines					OMAC (Structural Dimension)		
	CodeT	AgentVerse	ADAS	AFlow	DyLAN	Str-1	Str-2	Str-3
Pass@1	79.24±2.14	84.72±1.95	83.83±2.04	87.31±2.70	87.56±2.13	**88.69**±1.67	**88.73**±1.74	**89.08**±1.97

Method	OMAC (Functional Dimension)							
	Fun-1.1	Fun-1.2	Fun-1.3	Fun-1.4	Fun-1.5	Fun-1.6	Fun-1.7	Fun-2
Pass@1	**89.30**±1.41	**88.78**±1.64	**88.95**±1.80	**90.42**±1.75	**89.43**±2.06	**88.60**±1.42	**89.36**±1.04	**88.45**±0.97

Table 15. Performance on code generation tasks with DeepSeek-V2.5.

Method	Baselines					OMAC (Structural Dimension)		
	CodeT	AgentVerse	ADAS	AFlow	DyLAN	Str-1	Str-2	Str-3
Pass@1	79.88±1.52	85.11±2.62	83.98±2.69	87.95±2.06	87.74±1.95	**89.21**±1.76	**88.92**±2.31	**89.37**±1.47

Method	OMAC (Functional Dimension)							
	Fun-1.1	Fun-1.2	Fun-1.3	Fun-1.4	Fun-1.5	Fun-1.6	Fun-1.7	Fun-2
Pass@1	**89.90**±2.52	**88.95**±1.92	**89.35**±1.52	**90.35**±1.47	**89.74**±1.67	**89.21**±2.04	**89.90**±1.81	**89.02**±1.74

C.2.6. Experiments on Other Benchmarks

Firstly, we'd like to highlight that the benchmarks we've chosen in the main paper (HumanEval, MMLU, MATH) are well-established in the MAS literature and widely utilized by recent prominent studies such as DyLAN (Liu et al., 2024), ADAS (Hu et al., 2025), LLM Debate (Du et al., 2023), and Agentverse (Chen et al., 2024a), for evaluating both multi-turn MAS and their optimization methods. Thus, we consider these benchmarks sufficiently representative to evaluate MAS performance, ensuring that our results are reliable and comparable with existing literature.

Nevertheless, to substantiate evaluate the effectiveness and generalizability of OMAC on more challenging and complex scenarios, we conducted additional experiments on the on the MBPP (Austin et al., 2021) benchmark and GAIA (Mialon et al., 2024) benchmark. MBPP benchmark consists of crowd-sourced Python programming tasks designed to emulate more complex and realistic problem-solving scenarios encountered by entry-level programmers. Similarly, we adopt DyLAN's setup on the HumanEval benchmark as the default setting for OMAC, since both benchmarks focus on code generation

Table 16. Performance on code generation tasks with Qwen-2.5-72B.

Method	Baselines					OMAC (Structural Dimension)		
	CodeT	AgentVerse	ADAS	AFlow	DyLAN	Str-1	Str-2	Str-3
Pass@1	79.27±2.68	84.14±2.41	83.29±2.67	86.95±2.49	87.02±2.72	**88.32**±1.97	**88.56**±2.15	**88.91**±1.87

Method	OMAC (Functional Dimension)							
	Fun-1.1	Fun-1.2	Fun-1.3	Fun-1.4	Fun-1.5	Fun-1.6	Fun-1.7	Fun-2
Pass@1	**89.26**±1.87	**88.83**±2.41	**88.74**±1.77	**90.02**±1.65	**89.13**±1.98	**88.64**±1.83	**88.95**±2.01	**88.16**±1.33

tasks. As shown in Table 17, the results confirm OMAC's superior performance compared to most recent MAS optimization frameworks, including DyLAN (Liu et al., 2024) and AFLOW (Zhang et al., 2025c), on more challenging tasks.

The GAIA benchmark consists of challenging real-world questions that require a broad set of fundamental capabilities, including reasoning, multi-modal understanding, web browsing, and general tool-use proficiency. Following prior work on similar tasks (Yao et al., 2022; Huang et al., 2023), we adopt a standard set of agents for tool-using tasks as our default configuration. Specifically, the default agent set includes Self-Consistency, Chain-of-Thought, LLM-Debate, Testing, Ensemble, Self-Refine, and ReAct. As shown in Table 18, OMAC achieves superior performance compared to existing MAS and MAS optimization approaches on this challenging tool-calling benchmark.

Table 17. Performance on MBPP benchmark.

Method	Baselines				OMAC (Structural Dimension)		
	CodeT	CAMEL	Aflow	DyLAN	Str-1	Str-2	Str-3
Pass@1(%)	65.72±2.04	70.43±2.41	81.45±1.98	79.32±2.03	**82.81**±1.87	**82.50**±1.69	**83.23**±2.35

Method	OMAC (Functional Dimension)							
	Fun-1.1	Fun-1.2	Fun-1.3	Fun-1.4	Fun-1.5	Fun-1.6	Fun-1.7	Fun-2
Pass@1(%)	**82.49**±2.81	**83.30**±2.45	**82.97**±1.84	**83.02**±1.48	**83.43**±1.60	**83.11**±1.73	**83.06**±1.36	**82.35**±1.55

Table 18. Performance on GAIA benchmark (Level 2).

Method	Baselines				OMAC (Structural Dimension)		
	AutoGPT	ADAS	Aflow	DyLAN	Str-1	Str-2	Str-3
Pass@1(%)	1.04±0.15	3.87±1.25	8.23±2.63	9.82±2.41	**10.62**±1.96	**10.93**±1.57	**11.58**±1.05

Method	OMAC (Functional Dimension)							
	Fun-1.1	Fun-1.2	Fun-1.3	Fun-1.4	Fun-1.5	Fun-1.6	Fun-1.7	Fun-2
Pass@1(%)	**10.89**±2.74	**11.15**±2.10	**11.36**±1.77	**12.67**±1.21	**11.64**±1.33	**12.98**±1.57	**11.01**±1.60	**12.87**±2.48

C.2.7. Training Curve

We further evaluate OMAC's performance after each contrastive reasoning step when iteratively optimizing two dimensions (Fun1.2 then Fun1.1) over four iterations on arithmetic reasoning tasks. In particular, we report the performance curves on both the training set and the test set in Figure 8.

Although OMAC's optimization is guided by performance on a fixed training set, the results reveal a strong alignment between training and test performance, indicating that no overfitting occurs. We attribute this robustness to two key design choices. First, the optimization dimensions are constructed to target generalizable, task-level instructions rather than narrowly tailored prompts designed for specific cases. Second, each optimization iteration evaluates performance averaged across the entire training set, which inherently discourages over-specialization to particular subsets of data.

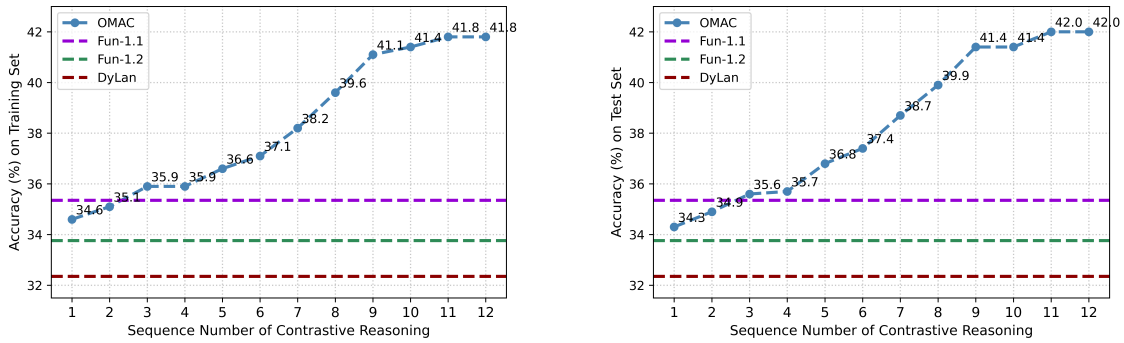

Figure 8. Performance after each contrastive reasoning on arithmetic reasoning tasks. The X-axis represents the sequence number of contrastive reasoning during four iterations of optimization, while each point indicates the MAS's performance with the optimized dimensions on the training set (left) and the test set (right).

D. Additional Discussions

D.1. Discussion on Optimization Effectiveness

In general, OMAC's effectiveness arises from its robust iterative optimization approach explicitly guided by supervised downstream task performance. More concretely, the Semantic Initializer is responsible for broadly exploring the semantic space by generating a diverse set of candidate agents/controllers. Meanwhile, the Contrastive Comparator focuses specifically on exploiting supervised feedback signals to refine the candidates by conducting targeted contrastive analyses iteratively.

During the iterative optimization, it is indeed possible for the Contrastive Comparator to occasionally generate candidate agents or controllers that underperform relative to the established positive samples. However, to mitigate this, OMAC incorporates a robust mechanism grounded in rigorous supervised performance evaluation. Each newly generated candidate undergoes thorough downstream task assessment leveraging the whole training set, and any candidate failing to surpass the performance of the existing positive sample is subsequently treated as a negative example. This negative example is then utilized in subsequent iterations for enhanced contrastive reasoning, systematically improving solution quality over successive rounds.

As illustrated in Section 3.2, both Semantic Initializer and Contrastive Comparator significantly depend on the reasoning capability of the underlying black-box LLM. However, we emphasize that OMAC addresses this limitation through an iterative optimization process explicitly guided by supervised performance feedback. This iterative approach systematically integrates insights from previous successes and failures, allowing the LLM to progressively improve its reasoning beyond single-round capabilities.

Our empirical results demonstrate substantial and consistent performance improvements from this iterative optimization, underscoring its effectiveness beyond the baseline capabilities of the static, single-pass reasoning provided by the LLM. Also, the ablation experiments demonstrate substantial and reliable performance gains achieved through this comparator-driven iterative refinement approach. These findings highlight the practical robustness and efficacy of OMAC's iterative optimization strategy, showcasing significant improvements over baseline methods and the Semantic Initializer alone. Overall, this evidence underscores the critical importance of iterative refinement in maximizing the reasoning potential and solution quality of black-box LLM-based systems.

D.2. Discussion on Optimization Performance

We present the performance improvements achieved by OMAC for single-dimension optimization in Tables 1-3, and for multi-dimension optimization in Figures 4-6. Although the average relative single-dimension optimization gains achieved by OMAC (4.6%, 2.8%, and 5.3% across the three benchmarks per dimension) may appear modest at first glance, these improvements are, in fact, significant and comparable to those reported by recent MAS optimization studies, such as DyLAN (Liu et al., 2024), conducted under similar experimental conditions.

More importantly, we emphasize that OMAC's capability to simultaneously optimize multiple dimensions results in **substantially larger performance gains compared to single-dimension optimization**. For example, as demonstrated in

Figures 4, the improvement in arithmetic reasoning tasks increases notably from 2.9% to 9.6% when jointly optimizing Fun-1.1 and Fun-1.2. These findings underscore the significant and practical effectiveness of OMAC's systematic multi-dimensional optimization approach in enhancing MAS performance.

Besides that, as illustrated in Section 3.3, we selectively incorporate only the top two performing dimensions in joint optimization for computational efficiency. Because OMAC requires iterative optimization, jointly optimizing more dimensions substantially increases the overall computation cost. Nevertheless, we conduct additional HumanEval experiments by jointly optimizing the top 3 performing dimensions. The results show that optimizing three dimensions ultimately achieves about 5% higher performance than optimizing 2 dimensions at convergence, but it also requires substantially more iterations to converge, roughly 1.8 times as many.

Last, we further conduct experiments to study the effect of dimension order. The main finding is relatively straightforward and trivial: changing the optimization order can lead to some performance differences in the early iterations, but the final performance after convergence is nearly the same across different orders. We also observe that the number of iterations required for convergence is largely unchanged.

D.3. Discussion on Novelty and Contribution

The main novelty of our proposed OMAC framework lies in the conceptual framing and integration of the structural and functional optimization components within a unified optimization framework for MAS optimization.

Specifically, we summarize our contributions as fourfold:

- We identify five key optimization dimensions that comprehensively cover both agent functionality and collaboration structure in MAS.
- We propose a general optimization framework that utilizes these two actors to optimize each of these five dimensions.
- We further introduce an iterative algorithm that enables joint optimization across multiple dimensions.
- We empirically validate our framework on multiple standard benchmarks against several MAS baselines, demonstrating consistent performance improvements and practical value.

Therefore, while some existing literature on prompt and agent optimization (Zhang et al., 2025c; Shinn et al., 2023) have explored some conceptually similar principles behind our design of the Semantic Initializer and the Contrastive Comparator, we believe the conceptual generalization, unified optimization framework, and empirical validation together constitute sufficient novelty and contribution to the field.

D.4. Discussion on Centralized MAS

The current OMAC framework design primarily targets flat MAS architectures, where agents are typically organized into sequential workflows to collaborate effectively. An alternative MAS architecture, known as centralized MAS (Li et al., 2025; Chen et al., 2024b), features a master agent coordinating other agents for collaboration and problem-solving.

We contend that the general definitions of OMAC's optimization dimensions, including both functional and structural ones, can also effectively apply to centralized MAS frameworks. Specifically, optimization dimensions such as enhancing existing agents, developing new agents, globally and locally selecting collaborating agents, and managing agent communication remain broadly applicable. Also, the implementation details for functional dimensions remain consistent across the two frameworks. However, structural dimensions, especially communication control, would require practical adaptations for centralized MAS, as communication typically occurs between the master agent and subordinate agents rather than between arbitrary pairs of agents. Investigating the effectiveness and detailed design adaptations of OMAC within centralized MAS settings represents a promising direction for future research.

D.5. Discussion on Bias Issue

As discussed in Section D.1, employing LLMs for contrastive reasoning may introduce biases. However, to mitigate this risk, OMAC integrates a robust iterative optimization process supported by rigorous supervised performance evaluations. Each candidate produced by the comparator is thoroughly validated against the entire training dataset, with weaker candidates

systematically labeled as negative examples for subsequent iterations. This iterative refinement progressively sharpens the optimization direction.

In other words, even if a candidate from the contrastive reasoning process is biased, its adverse effect is promptly detected by subsequent evaluation and prevented from influencing later optimization stages. Instead, such biased outcomes are repurposed as negative examples in contrastive reasoning, thereby reinforcing the distinction between positive and negative cases and ultimately enhancing the overall optimization design.

However, to further enhance reasoning effectiveness, an alternative strategy could be incorporating contrastive reasoning over two batches of positive and negative samples instead of a single pair, which may enhance the generalizability of contrastive reasoning and reduce biases. But this approach also significantly increases prompt length and computational costs. Exploring the balance between reasoning effectiveness and computational efficiency represents another interesting and valuable direction for future research.

D.6. Discussion of Future Work

While OMAC is designed as a general framework for optimizing MAS in complex tasks where empirical results validated its effectiveness, we identify several directions that warrant further exploration.

First, OMAC currently employs two LLM-based actors, the Semantic Initializer and the Contrastive Comparator, for MAS optimization. Although the existing literature on gradient-based methods targeting MAS optimization in multi-step collaboration settings remains very limited (as discussed in Section 5), exploring this direction is highly promising. Gradient-based optimization may enable more controllable, integrated, and efficient optimization processes. In particular, extending current RL-based methods for agentic optimization, such as Spiral (Liu et al., 2025) or MHGPO (Chen et al., 2025), into the problem setting and methodological framework of OMAC represents a valuable research avenue.

Second, tool usage has shown substantial promise and effectiveness in LLM-powered agent systems. Although our experiments on the HumanEval and GAIA benchmark have already demonstrated OMAC's potential to optimize agents' tool-use abilities, explicitly extending OMAC to optimize agent collaboration and utilization across more advanced and diverse tools for solving complex real-world problems would be a significant next step. Emerging benchmarks like SWE-bench (Jimenez et al., 2023) provide suitable environments for evaluating complex tool interactions. Future studies based on these benchmarks, along with comprehensive comparisons against existing MAS approaches, could yield valuable insights.

Third, while OMAC is designed for MAS optimization across the five general dimensions we identified, our current experiments primarily focus on prompt and in-context example optimization. Other components, such as memory management or retrieval mechanisms, may also require optimization under specific task contexts. Nonetheless, these elements still conceptually fall under agent functionality or MAS structure optimization and can therefore be naturally integrated into OMAC. By leveraging LLMs' contrastive reasoning with positive/negative sample pairs, these additional optimization tasks can be addressed following our single- and multi-dimension algorithms. Exploring OMAC's effectiveness in optimizing such components would be a valuable direction for future work.

Fourth, OMAC currently adopts an iterative optimization pattern for multi-dimension optimization. As discussed in Section 3.3 and empirically verified in Appendix C.2.3, this approach is well motivated by the necessity of maintaining effective contrastive reasoning. Nevertheless, investigating joint multi-dimension optimization remains a promising direction due to its potential for improved integration and efficiency. Methods such as gradient blending, Pareto-efficient optimization, evolutionary algorithms, data mixing, Bayesian optimization, or other multi-objective optimization techniques could be explored in combination with OMAC to enhance joint optimization. Also, exploring more efficient optimization protocols to address the high computational cost shared by OMAC and other existing MAS optimization methods is a promising direction. Although this is not the primary focus of this work, low-cost adaptation remains an important and valuable area for future research.

Finally, our five proposed optimization dimensions for MAS are designed to be general and fundamental, derived from conceptualizing the MAS collaboration process as information flow over a graph, where agents correspond to nodes and communication channels to edges (see Appendix B). This paper primarily focuses on defining these dimensions and proposing the corresponding optimization methodology, rather than conducting a rigorous theoretical analysis. A deeper theoretical study of the optimization landscape, including formal guarantees of optimal configurations, would provide valuable insights and strengthen the conceptual foundation of MAS optimization research.

E. Prompts and Examples

E.1. Prompt Templates

As described in Section 4, we adopt the agent designs and collaboration structures from the SOTA method DyLAN (Liu et al., 2024) as the default configuration for OMAC on all datasets. Specifically, the default instruction prompts for existing agents are directly inherited from DyLAN'd default setup and detailed in the original paper (Liu et al., 2024).

The prompt templates uniquely for OMAC include those designed for the Semantic Initializer and the Contrastive Comparator across the five optimization dimensions. Furthermore, as explained in Section 3.2, the only variation in the prompts for the two actors across different tasks lies in the contextual description of the MAS and the given task. Therefore, we present here the prompt templates for these two actors across the five optimization dimensions on general reasoning task. For details on other tasks, please refer to our code repository at: https://github.com/xiwenchao/OMAC.

Prompts of the Semantic Initializer for five dimensions are as follows:

Table 19. Prompts of Semantic Initializer for five dimensions.

Fun-1	Generate {`initialization number`} distinct prompts to instruct an LLM to resolve some general reasoning problems acting as the given role: {`optimized agent role`}. Each prompt should guide the model to accurately and efficiently resolve problems while adhering to the specified role. Each prompt must begin strictly with the following content: {`basic description of the optimized agent`}. Then, you should consider adding more detailed, logical, and through instructions, which can help the LLM resolve problems better acting as the given role. Do not output anything currently. Instead, I will provide a sequence number, and you should return only the corresponding prompt one by one. Do not create any specific instances of the problems in the prompt, cause they are not provided now. Ensure that the generated prompts follow the given example format but differ in content and structure from the example itself. The example is as follows: {`one-shot example`}.
Fun-2	Generate {`initialization number`} distinct prompts to instruct an LLM to resolve some general reasoning problems related to math, hard science, humanities, and social sciences. There are some existing agents in the system to resolve the problems, whose roles are: {`roles of existing agents`}. You need to generate some new roles and prompts for the LLM to better resolve the problems. First, determine the roles of these prompts. Next, create the prompts that instruct the LLM to resolve problems based on the defined roles. Do not output all the generated roles and prompts at once. Instead, I will request either the k-th role or the k-th prompt individually. When asked, directly output the corresponding content of the role name or prompt one at a time. Do not create any instances of the problems in the prompt, cause they are not provided now. You can decide the content and detailed functional instructions of the roles and prompts. You may consider adding more detailed instructions to help the LLM resolve problems. The following is an example of a role and the corresponding prompt (also ensure your output is different from the example role and prompt): {`one-shot example`}.

Str-1	Generate {`initialization number`} distinct prompts for an LLM to choose some top agents best suited for resolving some general reasoning problems related to math, hard science, humanities, social sciences, etc. Don't directly output all the generated prompts. I will provide you the sequence number of the prompt. Then you should directly output the content text of the corresponding prompt one by one. Each prompt should decide and specify the number of the chosen agents. The minimal number is 4 and maximum number is 7. Each prompt should help to accurately and efficiently identify the top agents best suited for problem-solving. Note that all information about the task and candidate agents has been previously provided as the context. The prompt generated here will be added to the context to form the final prompt for agent selection. You may consider adding more detailed and thorough instructions to help the LLM select the top agents better. The following is an example of a prompt (also ensure your output is different from the example prompt): {`one-shot example`}.
Str-2	Generate {`initialization number`} distinct prompts for an LLM to choose some top solutions for best resolving some general reasoning problems related to math, hard science, humanities, social sciences, etc. Don't directly output all the generated prompts. I will provide you with the sequence number of the prompt. Then you should directly output the content text of the corresponding prompt one by one. You can decide the number of the chosen solutions and the content of the prompt. The number of solutions should be between 2 and 7. The prompt should help to accurately and efficiently select the top solutions that resolve the given problems best. Note that all the solutions and the problem have been previously provided as the context. The prompt generated here will be added to the context to form the final prompt for solution selection. You may consider adding more detailed and thorough instructions to help the LLM select the top solutions better. The generated prompt should specify the output format like the given example (also ensure that it is different from the example prompt): {`one-shot example`}.
Str-3	Generate {`initialization number`} distinct prompts for an LLM to choose some top candidate agents whose generated solutions to some general reasoning problems may be useful as inputs for the current agent to produce improved solutions. Don't directly output all the generated prompts. I will provide you with the sequence number of the prompt. Then you should directly output the content text of the corresponding prompt one by one. You should decide the number of chosen agents and the content of the prompt. The number of chosen agents should be between 4 and 7. Each prompt should help to accurately and efficiently identify the top candidate agents whose generated solutions are helpful to be taken as input for the current agent. Note that all information about the candidate agents and the current agent has been previously provided as the context. The prompt generated here will be added to the context to form the final prompt for agent selection. You may consider adding more detailed and thorough instructions to help the LLM select the candidate agents better. The following is an example of a prompt (also ensure your output is different from the example prompt): {`one-shot example`}.

Prompts of the Contrastive Comparator for five dimensions are as follows:

Table 20. Prompts of Contrastive Comparator for five dimensions.

Fun-1	Generate and output a child prompt for an LLM to resolve some general reasoning problems acting like the given role: {`optimized agent role`}. At the end, a pair of parent prompts is provided: one positive and one negative. The positive parent prompt has been shown to be more effective and efficient in guiding the LLM to resolve problems following the given role. Your task is to carefully compare the two parent prompts, identifying the key reasons why the positive parent prompt performs better. Based on these insights, generate and output a child prompt that further improves upon the positive parent prompt to enhance problem-solving. Do not create any instances of the problem in the prompt, cause they are not provided now. The child prompt must begin strictly with the following content: {`basic description of the optimized agent`}. Then, you can consider adding more detailed, logical, and through instructions based on the insights you have gained from the comparison. Output only the content of the child prompt excluding the reasoning process. Here is the positive-negative pair of parent prompts: {`positive/negative prompts`}.
Fun-2	Generate and output a pair consisting of a role name and its corresponding prompt, designed to resolve some general reasoning problems (related to math, hard science, humanities, social sciences, etc.). First, determine the role of the LLM. Next, create a prompt that effectively instructs the LLM to resolve problems based on this role. I will provide two parent role-prompt pairs: one positive and one negative. The positive pair has been proven to be more effective in guiding the LLM to generate high-quality solutions for general problems. Your task is to carefully analyze both parent pairs, identifying the factors that make the positive pair superior. Based on this analysis, generate and output a child role and prompt pair that improves upon the positive parent pair and leads to even better problem resolution. The child prompt must be distinct from both parent prompts while incorporating the lessons learned from their comparison. Do not output the role and prompt immediately. I will request them separately, and when asked, provide only the corresponding content—either the role name or the prompt. Here is the positive-negative pair of parent prompts: {`positive/negative prompts`}.
Str-1	Create and output a child prompt for an LLM to choose some top agents that best suited for resolving some general reasoning problems related to math, hard science, humanities, social sciences, etc. I will provide you with a pair of parent prompts. Then you should only output a child prompt according the following instructions: The positive parent prompt is proven to be more helpful and efficient to instruct the LLM to select more useful and effective agents for problem resolution. You should carefully compare the two parent prompts, finding the potential reasons why the positive parent prompt is better than the negative parent prompt. Based on that, you should generate and output a child prompt that can help to choose top agents more effectively and efficiently than the positive prompt. The child prompt should follow the format of the parent prompts. The child prompt should be different from the parent prompts. And directly output the content text of the child prompt. Here is the positive-negative pair of parent prompts: {`positive/negative prompts`}.

Str-2	Create and output a child prompt for an LLM to choose some top solutions for resolving some general reasoning problems (related to math, hard science, humanities, social sciences, etc.) best. I will provide you with a pair of parent prompts. Then you should only output a child prompt according the following instructions: The positive parent prompt is proven to be more helpful and efficient to instruct the LLM to select more useful and effective solutions to resolve the problems. You should carefully compare the two parent prompts, finding the potential reasons why the positive parent prompt is better than the negative parent prompt. Based on that, you should generate and output a child prompt that can help to choose top solutions more effectively and efficiently than the positive prompt. The child prompt should follow the format of the parent prompts. The child prompt should be different from the parent prompts. And directly output the content text of the child prompt. Here is the positive-negative pair of parent prompts: {`positive/negative prompts`}.
Str-3	Create and output a child prompt for an LLM to choose some candidate agents whose generated solutions to some general reasoning problems may be useful as inputs for the current agent to produce improved solutions. I will provide you a pair of parent prompts. Then you should only output a child prompt according the following instructions: The positive parent prompt is proven to be more helpful and efficient to instruct the LLM to select more useful and effective agents. You should carefully compare the two parent prompts, finding the potential reasons why the positive parent prompt is better than the negative parent prompt. Based on that, you should generate and output a child prompt that can help to choose top agents more effectively and efficiently than the positive prompt. The child prompt should follow the format of the parent prompts. The child prompt should be different from the parent prompts. And directly output the content text of the child prompt. Here is the positive-negative pair of parent prompts: {`positive/negative prompts`}.

E.2. Examples

Besides the prompts for the two actors, we further provide examples of instruction prompts before and after optimizing corresponding to the five optimization dimensions on general reasoning task.

Dimension	Fun-1
Default Prompt	You are a mathematician. You are good at math games, arithmetic calculation, and long-term planning. Please provide a step-by-step approach to solve some provided mathematical problems.
Optimized Prompt	You are a mathematician. You are good at math games, arithmetic calculation, and long-term planning. When devising a step-by-step strategy for solving a complex optimization problem involving multiple variables and constraints, consider breaking down the problem into smaller, more manageable sub-problems. Analyze the objective function and constraints to understand their impact on the feasible solution space. Explore various solution methods such as linear programming, gradient descent, or integer programming to determine the most suitable approach for the problem at hand. Finally, implement the chosen method meticulously while constantly evaluating its efficiency and accuracy. If necessary, iterate on the strategy to improve the optimization process. Explain the rationale behind each step of your chosen approach.

Dimension	Fun-2
Default Prompt	You are a super-intelligent AI assistant capable of performing tasks more effectively than humans.
Optimized Prompt	You are a Data Analyst. You need to provide a detailed explanation of how to use statistical techniques to identify and analyze patterns for the data in the given question. Provide a step-by-step guide on how to conduct the analysis for a given dataset. Also, provide a clear and detailed explanation of the process for selecting an appropriate statistical approach.

Dimension	Str-1
Default Prompt	Here is the task and question: {`task context`}. These are the agents and their functional description: {`candidate agents' functionalities`}. Take functionality, efficiency, and necessity into consideration, choose top 5 agents best suited for resolving the given problem. Think it step by step. Put your answer in the form like [1,3,4,5,6] at the end of your response.
Optimized Prompt	Here is the task and question: {`task context`}. These are the agents and their functional description: {`candidate agents' functionalities`}. To address general reasoning problems across various disciplines such as math, hard science, humanities, and social sciences, it is crucial to identify the top 6 agents with exceptional problem-solving abilities and expertise in diverse areas. These agents should demonstrate proficiency in critical thinking, logical reasoning, and analytical skills to effectively resolve multifaceted problems. Evaluate the candidates based on their demonstrated knowledge, adaptability, and capability in tackling complex reasoning challenges. After carefully assessing these criteria, provide your response in the form [1,2,3,4,5,6] at the end of your submission.

Dimension	Str-2
Default Prompt	Here is the task and question: {`task context`}. These are the solutions to the problem from other agents: {`previous agents' solutions`}. Please choose the best 2 solutions and think step by step. Put your answer in the form like [1,2] or [3,4] at the end of your response.
Optimized Prompt	Here is the task and question: {`task context`}. These are the solutions to the problem from other agents: {`previous agents' solutions`}. Analyze the given context thoroughly and choose the top 3 solutions based on their ability to accurately and efficiently resolve the given problems. The selected top solutions should be effective in resolving reasoning problems in various fields including math, science, humanities, and social sciences. Consider practical applicability, logical soundness, and clarity of each solution. Then think step by step to clearly explain how each solution can be applied in different scenarios. Please put your answer in the form like [1,2,3] at the end of your response.

Dimension	Str-3
Default Prompt	Here is the functional description of the current agent: {`current agent' functionality`}. These are the candidate agents and their functional description: {`candidate agents' functionalities`}. Take functionality, efficiency, and necessity into consideration. Select the top 5 candidate agents whose generated solutions to some general reasoning problems can be mostly useful as inputs for the current agent to produce improved solutions. Think it step by step. Put your answer in the form like [1,2,3,4,5] at the end of your response.
Optimized Prompt	Here is the functional description of the current agent: {`current agent's functionality`}. These are the candidate agents and their functional description: {`candidate agents' functionalities`}. Consider the agents whose generated solutions are most likely to improve the current agent's problem-solving capabilities. Select the top 4 candidate agents based on the effectiveness, practicality, and relevance of their solutions. Consider their ability to address complex challenges, think outside the box, and produce innovative perspectives that could benefit the current agent in enhancing its problem-solving capabilities. Prioritize agents whose solutions offer a fresh approach, logical reasoning, and effective problem-solving strategies. Present your answer in the format [1,2,3,4] at the end of your response.