HALO: Hierarchical Autonomous Logic-Oriented Orchestration for Multi-Agent LLM Systems

Zhipeng Hou[*]
Nanjing University of Posts and Telecommunications
japhonehou@gmail.com

Junyi Tang
Nanjing University of Posts and Telecommunications
b23012214@njupt.edu.cn

Yipeng Wang
Chongqing University
yipengxw@gmail.com

Abstract

Recent advancements in Multi-Agent Systems (MAS) powered by Large Language Models (LLMs) have demonstrated tremendous potential in diverse task scenarios. Nonetheless, existing agentic systems typically rely on predefined agent-role design spaces and static communication structures, limiting their adaptability as well as flexibility in complex interaction environments and leading to subpar performance on highly specialized and expert-level tasks. To address these issues, we introduce **HALO**, a multi-agent collaboration framework based on a hierarchical reasoning architecture. Specifically, we incorporate a high-level planning agent for task decomposition, mid-level role-design agents for subtask-specific agent instantiation, and low-level inference agents for subtask execution. Particularly, subtask execution is reformulated as a structured workflow search problem, where Monte Carlo Tree Search (MCTS) systematically explores the agentic action space to construct optimal reasoning trajectories. Additionally, as the majority of users lack expertise in prompt engineering, we leverage an Adaptive Prompt Refinement module to transform raw queries into task-specific prompts. Empirical evaluations on Code Generation (HumanEval), General Reasoning (MMLU), and Arithmetic Reasoning (MATH) benchmark datasets highlight the effectiveness of HALO, yielding a **14.4%** average improvement over state-of-the-art baselines. Notably, HALO achieves up to **13.3%** performance gain on the Moral Scenarios subject in the MMLU benchmark and up to **19.6%** performance gain on the Algebra subarea in the MATH benchmark, indicating its advanced proficiency in tackling highly specialized and expert-level tasks. The code repository is available at https://github.com/23japhone/HALO.

1   Introduction

Large Language Models (LLMs), such as OpenAI o3 [1] and Deepseek R1 [2], have demonstrated remarkable capabilities in both language understanding and reasoning. These advancements unlock tremendous potential for LLM-based multi-agent systems to address a broad spectrum of downstream tasks, including code generation [3, 4], mobile device control [5, 6], video gaming [7] and open-domain question answering [8, 9]. Consequently, the formulation of effective agentic systems is crucial to fully harnessing the capabilities of LLMs across diverse downstream applications.

---

[*]First author and corresponding author, correspondence to japhonehou@gmail.com
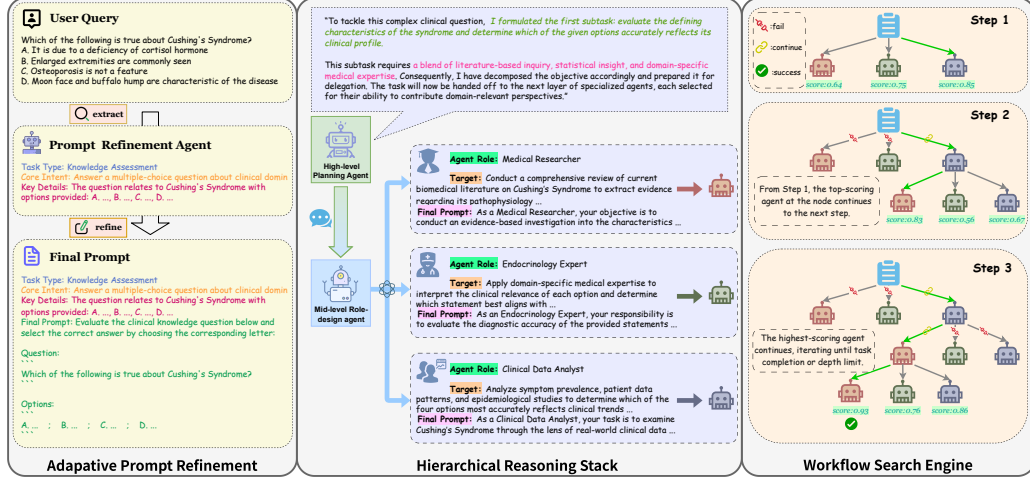
Figure 1: The overview of HALO framework. HALO consists of three modules: (1) Adaptive Prompt Refinement (Section 3.2), where user queries are refined into high-quality and LLM-comprehensible prompts; (2) Hierarchical Reasoning Stack (Section 3.3), which is responsible for task decomposition, role instantiation, and subtask execution; and (3) Workflow Search Engine (Section 3.4), which explores multi-agent collaboration and constructs optimal workflows. Green paths denote optimal reasoning trajectories, while red paths are pruned during search.

However, existing agentic systems often struggle to maintain robust performance in complex interaction environments and expert-level tasks. This limitation arises from predefined agent-role design spaces [5, 10] and static communication workflows [11, 12], which heavily depend on expert insight and manually-designed policies. Meanwhile, as the majority of users lack expertise in prompt engineering, poorly formulated queries have greatly hindered the comprehension of agents, ultimately leading to inefficient task execution. Together, these challenges underscore two fundamental problems: (1) how can agentic systems self-organize and coordinate in unfamiliar environments with minimal manual intervention; and (2) how can user queries be refined to improve the overall efficiency and effectiveness of multi-agent collaboration?

In response, we introduce **HALO**, a **H**ierarchical **A**utonomous **L**ogic-Oriented **O**rchestration framework focused on addressing complex interaction environments and expert-domain reasoning tasks. To this end, HALO incorporates an extensible agent-role instantiation mechanism and a dynamic communication architecture to replace the rigidity of predefined role spaces and static workflows. Additionally, HALO employs a prompt engineering module for user query refinement. The overview of HALO is illustrated in Figure 1.

Specifically, HALO functions in a three-stage paradigm. The first stage is **Adaptive Prompt Refinement** (detail in Section 3.2), where the raw user query is transformed into a high-quality and LLM-comprehensible prompt. This module comprises four collaborative agents responsible for different phase of prompt refinement, ranging from query parsing to final synthesis. The second stage is **Hierarchical Reasoning Stack** (detail in Section 3.3), which employs a three-tier multi-agent collaboration architecture. At the top layer, a high-level planning agent decomposes the overall task into a sequence of subtasks. At the middle layer, mid-level role-design agents dynamically instantiate specialized agents tailored to the requirement of each subtask. At the bottom layer, low-level inference agents are responsible for executing each subtask through cooperation mechanisms. The third stage is **Workflow Search Engine** (detail in Section 3.4), which explores low-level inference agents collaboration and constructs optimal workflows. We reformulate subtask execution as a workflow search process, where Monte Carlo Tree Search (MCTS) [13] guides the exploration over action spaces. Each node in the search tree corresponds to an agent-generated response or an intermediate reasoning step, while edges denote possible transitions between reasoning states. Together, these nodes and edges form a path that represents a candidate reasoning trajectory for each subtask.

Extensive experiments (detail in Section 4.2) demonstrate that HALO outperforms strong baselines across Code Generation (HumanEval) [14], General Reasoning (MMLU) [15], and Arithmetic

Reasoning (MATH) [16] benchmarks, yielding a 14.4% average improvement. Notably, HALO achieves up to **13.3%** performance gain on the Moral Scenarios subject in MMLU and up to **19.6%** performance gain on the Algebra subarea in MATH, underscoring its advanced proficiency in highly specialized and expert-level tasks.

The key contributions of this work are as follows:

- We introduce a novel framework named HALO for task-oriented agent collaboration in three stages, marking a significant advancement beyond the limitations of expert insight and manually-designed policies.

- Experiments across three diverse tasks show that our method outperforms state-of-the-art baselines, confirming the effectiveness and adaptability of HALO across complex interaction environments and expert-domain reasoning tasks.

2 Related work

2.1 Prompt optimization

Prompt design underpins how LLM agents interpret instructions and coordinate actions. Recent work automates this process to reduce manual intervention and improve modularity. Promptor [17] decomposes goals into structured, role-specific prompts, enabling agents to focus on individual tasks. CAMEL [12] integrates a task-planner agent that generates role prompts to guide multi-agent interactions through instruction-driven simulation. While these methods improve modularity and task decomposition, current agentic systems still depend on handcrafted prompt templates or task-specific engineering, which limits generalization across domains. Recent frameworks like ComfyAgent [18] begin to incorporate prompt generation as part of evolving workflows. Nonetheless, the challenge of designing prompts that are both reusable and responsive to runtime context changes remains largely open, especially in scenarios where agent roles or task objectives are not predefined.

2.2 Role design in LLM-based architecture

Role design has become a central theme in recent multi-agent architectures. Manual role assignment is a common strategy for organizing agent behavior. MetaGPT [19], for instance, aligns agents with real-world job titles and governs their behavior through standard operating procedures. This static role configuration offers clarity and reproducibility, especially in well-defined domains. However, fixed roles often struggle in open-ended or dynamic settings where agent responsibilities must evolve in response to task changes. To address this, newer frameworks introduce role generation mechanisms that allow agents to create, adapt, or inherit roles dynamically. TPTU [20] and DyLAN [21] adopt hierarchical or layered structures where agents can delegate, transform, or refine responsibilities as collaboration progresses. These approaches mark a shift toward greater flexibility, although existing agentic systems still operate within partially predefined boundaries.

2.3 Cooperation optimization strategies

As the scale and complexity of LLM-based multi-agent systems grow, optimizing inter-agent co-operation becomes increasingly critical. Broadly, current systems adopt either centralized or de-centralized paradigms. Centralized frameworks, such as AgentLaboratory [22], ScoreFlow [23], and WORKFLOW-LLM [24], rely on controller agents to manage scheduling, communication, and output aggregation. While these architectures offer clear oversight, they can suffer from scalability issues and become bottlenecks in real-time or asynchronous settings. In contrast, decentralized approaches seek to distribute control and decision-making across agents. Some utilize peer-to-peer communication or identity-based protocols [25], while others apply game-theoretic negotiation [26]. These designs offer greater autonomy and robustness in unstructured environments. Recently, several frameworks reframe cooperation as a structured search process. AFlow [27], for instance, explores layered task workflows through dynamic graph traversal rather than relying on fixed execution paths. Such methods enable agents to adaptively construct interaction plans based on task-specific signals or peer responses, highlighting a promising direction for building more context-sensitive agent teams. Notably, some researchs apply reinforcement learning to refine cooperation strategies [28–32].

Table 1: Comparative analysis of HALO and existing representative LLM-based multi-agent frameworks across five key dimensions.

Method	Structure	Multi-Role	Role Assignment	Dynamic Structure	Team Optimization
ScoreFlow [23]	Central + Score-based	✗	✗	✗	✓
CAMEL [12]	Feedback Triplet	Manual	Manual	✗	✗
AgentVerse [33]	Hierarchical Tree	✓	Generated	✗	✗
MetaGPT [19]	SOP-guided Workflow	✓	Manual	✗	✗
DyLAN [21]	DAG + Feedback Loop	✓	Manual + Gen	✓	✓
ComfyAgent [18]	Modular Workflow	✓	Generated	✓	✗
AgentLaboratory [22]	Centralized Scheduler	✓	Manual	✗	✓
TPTU [20]	Hierarchical Planner	✓	Manual + Gen	✓	✗
AFlow [27]	Multi-layer DAG	✓	Manual + Gen	✓	✓
WORKFLOW-LLM [24]	Centralized Workflow	✓	Manual	✗	✓
Game-theoretic Workflow [26]	Decentralized Negotiation	✓	Adaptive	✓	✓
HALO (Ours)	**Hierarchical Structure + MCTS**	✓	Adaptive	✓	✓

A comparative summary of these cooperation frameworks across five dimensions is provided in Table 1. In contrast, HALO introduces a hierarchical reasoning architecture and reformulates subtask execution as a structured reasoning search. This design offers a balanced pathway between modular coordination and dynamic adaptability, particularly suited for complex interaction environments.

3 HALO framework

3.1 Problem formulation

In the proposed workflow search process, a workflow $\mathcal{W}$ is composed of a sequence of subtasks $\{T_1, T_2, \ldots, T_K\}$, where each subtask T_k is handled by a series of role-specialized LLM-based agents $\mathcal{A}_k = \{a_k^{(1)}, a_k^{(2)}, \ldots\}$. We reformulate subtask execution as the workflow search process over the workflow space $\mathcal{S}$, where each candidate workflow $\mathcal{W} \in \mathcal{S}$ represents a unique instantiation of subtasks and multi-agent interactions. Given a user query $\mathcal{Q}$ expressed in natural language, the objective is to construct an optimal reasoning workflow $\mathcal{W}^*$ that generates an expected answer $\hat{Y}$ aligned with the user query $\mathcal{Q}$. The optimization problem of HALO is defined as follows:

$$\mathcal{W}^* = \arg\max_{\mathcal{W} \in \mathcal{S}} \text{Value}(\mathcal{Q}, \mathcal{W}) \tag{1}$$

where $\text{Value}(\mathcal{Q}, \mathcal{W})$ evaluates the effectiveness of workflow $\mathcal{W}$ in addressing the user query $\mathcal{Q}$.

The overall framework of HALO is illustrated in Figure 1 and the algorithm is presented in Algorithm 1. Next, we elaborate on each component of the three-stage paradigm.

3.2 Adaptive Prompt Refinement

As the majority of users lack expertise in prompt engineering, user queries $\mathcal{Q}$ are often loosely structured or ambiguous in intents. To address this limitation and enhance the reliability of downstream reasoning, we propose an Adaptive Prompt Refinement module at the beginning of the HALO framework, which refines the raw user query $\mathcal{Q}$ into a structured and LLM-comprehensible prompt $\mathcal{Q}^*$. This module comprises four collaborative agents, each formalized as a functional mapping, denoted by $\mathcal{P}_1$ to $\mathcal{P}_4$. These agents are responsible for different stages of prompt refinement, ranging from query parsing to final synthesis (detail in Appendix A). The process is formulated as follows.

The process begins with the Task Parser Agent $\mathcal{P}_1$, which semantically analyzes the original query $\mathcal{Q}$ to extract three essential components: the task type $\mathcal{T}$, the core intent $\mathcal{I}$, and the key details $\mathcal{D}$. These components are then assembled into a structured triplet:

$$\mathcal{F} = \mathcal{P}_1(\mathcal{Q}) = (\mathcal{T}, \mathcal{I}, \mathcal{D}) \tag{2}$$

This structured task representation $\mathcal{F}$ serves as a global semantic context throughout the entire reasoning workflow.

Based on this structured task representation $\mathcal{F}$ and raw user query $\mathcal{Q}$, the Prompt Template Agent $\mathcal{P}_2$ constructs an initial prompt frame $\mathcal{Q}_0$ that includes a reformulated task description, clear reasoning objectives, bounded input conditions, and the explicit output format:

$$\mathcal{Q}_0 = \mathcal{P}_2(\mathcal{Q}, \mathcal{F}) \tag{3}$$

4

Algorithm 1 Algorithm of HALO

Input: User query $\mathcal{Q}$, prompt refinement agents $\mathcal{P}_1 \sim \mathcal{P}_4$, planning agent $\mathcal{A}_{\text{plan}}$, role-design agent $\mathcal{A}_{\text{role}}$, inference agent pool $\mathcal{A}_k$, subtask budget K

Output: Final answer $\hat{Y}$

1:   $\mathcal{F} \leftarrow \mathcal{P}_1(\mathcal{Q})$ ▷ Parse task type $\mathcal{T}$, intent $\mathcal{I}$, and details $\mathcal{D}$
2:   $\mathcal{Q}_0 \leftarrow \mathcal{P}_2(\mathcal{Q}, \mathcal{F})$ ▷ Build initial prompt template
3:   $\mathcal{Q}_{\text{opt}} \leftarrow \mathcal{P}_3(\mathcal{Q}_0, \mathcal{F})$ ▷ Optimize with prompting strategies
4:   $\mathcal{Q}^* \leftarrow \mathcal{P}_4(\mathcal{Q}_{\text{opt}}, \mathcal{F})$ ▷ Synthesize final prompt
5:   $H_0 \leftarrow \emptyset$ ▷ Initialize subtask execution history
6:   **for** $k = 1$ to K **do**
7:      $T_k \leftarrow \mathcal{A}_{\text{plan}}(\mathcal{Q}^*, \mathcal{F}, H_{k-1})$ ▷ Decompose query into subtask
8:      **if** $T_k = \texttt{stop}$ **then**
9:          **break**   ▷ Terminate if planning agent signals task completion
10:      **end if**
11:      $\mathcal{A}_k \leftarrow \mathcal{A}_{\text{role}}(T_k, \mathcal{Q}^*, \mathcal{F})$ ▷ Generate role-specific agents
12:      **for** each agent $a_k^{(i)} \in \mathcal{A}_k$ **do**
13:          $y_k^{(i)} \leftarrow a_k^{(i)}(T_k, \mathcal{Q}^*, \mathcal{F})$ ▷ Generate intermediate output by executing subtask T_k
14:          $\ell_k^{(i)}, v_k^{(i)} \leftarrow \text{Evaluate}(y_k^{(i)})$ ▷ Evaluate status label and quality score for output
15:          $\text{MCTS\_Backpropagate}(a_k^{(i)}, \ell_k^{(i)}, v_k^{(i)})$ ▷ Propagate execution feedback
16:      **end for**
17:      $H_k \leftarrow H_{k-1} \cup \{T_k, \hat{Y}_k\}$ ▷ Update subtask execution history
18:      **if** early-stop(H_k) is True **then**
19:          **break**   ▷ Terminate if early-stopping mechanism is triggered
20:      **end if**
21:   **end for**
22:   $\hat{Y} \leftarrow \text{Aggregate}(\{y_k^{(i)}\})$ ▷ Select final answer
23:   **return** $\hat{Y}$

To further refine the initial prompt template, the Prompt Optimization Agent $\mathcal{P}_3$ incorporates slow-thinking prompting strategies [34–38] as well as tool calling [39] instructions, generating an optimized prompt $\mathcal{Q}_{\text{opt}}$:

$$\mathcal{Q}_{\text{opt}} = \mathcal{P}_3(\mathcal{Q}_0, \mathcal{F}) \tag{4}$$

Finally, the Prompt Generator Agent $\mathcal{P}_4$ synthesizes the optimized structure into the final refined prompt $\mathcal{Q}^*$, which serves as the entry point for downstream multi-agent reasoning:

$$\mathcal{Q}^* = \mathcal{P}_4(\mathcal{Q}_{\text{opt}}, \mathcal{F}) \tag{5}$$

3.3 Hierarchical Reasoning Stack

Following the refined prompt $\mathcal{Q}^*$, HALO proceeds to the Hierarchical Reasoning Stack module, which comprises three collaborative layers, including the high-level planning agent $\mathcal{A}_{\text{plan}}$ for task decomposition, the mid-level role-design agents $\mathcal{A}_{\text{role}}$ for dynamic agent instantiation, and the low-level inference agents $\mathcal{A}_k = \{a_k^{(1)}, a_k^{(2)}, \dots\}$ for subtask execution.

At the top layer, the high-level planning agent $\mathcal{A}_{\text{plan}}$ receives the refined prompt $\mathcal{Q}^*$ and the global structured task representation $\mathcal{F}$. Based on this information, it decomposes the overall task into a sequence of subtasks $\{T_1, T_2, \dots, T_K\}$ (detail in Appendix B). Instead of performing full decomposition in advance, $\mathcal{A}_{\text{plan}}$ adopts a step-wise strategy, generating one subtask at a time and iteratively updating its decomposition policy based on the execution history of preceding subtasks:

$$T_k = \mathcal{A}_{\text{plan}}(\mathcal{Q}^*, \mathcal{F}, H_{k-1}) \tag{6}$$

where H_{k-1} denotes the execution history of preceding subtasks for the subtask T_k.

Each generated subtask T_k is delegated to the mid-level role-design agents $\mathcal{A}_{\text{role}}$, which instantiate a set of specialized LLM-based agents $\mathcal{A}_k = \{a_k^{(1)}, a_k^{(2)}, \dots\}$ to execute the subtask T_k. This instantiation process is jointly guided by the semantics of the subtask T_k, the refined prompt $\mathcal{Q}^*$, and

Figure 2: The illustration of how Monte Carlo Tree Search (MCTS) guides multi-agent reasoning through selection, expansion, simulation, and backpropagation stages. Each node represents an agent and edge transitions are guided by execution outcomes as well as evaluation feedback.

the global structured task representation $\mathcal{F}$, ensuring that each generated agent $a_k^{(i)}$ is well aligned with the subtask T_k requirements. For each generated agent $a_k^{(i)}$, a role-specific system prompt $\rho_k^{(i)}$ is assigned to govern its behavior within the subtask T_k (detail in Appendix C):

$$a_k^{(i)} = \mathcal{A}_{\text{role}}(T_k, \mathcal{Q}^*, \mathcal{F}) \Rightarrow \rho_k^{(i)} \tag{7}$$

For each subtask T_k, the low-level inference agents $\mathcal{A}_k = \{a_k^{(1)}, a_k^{(2)}, \dots\}$ receives the subtask T_k, the refined prompt $\mathcal{Q}^*$, and the global structured task representation $\mathcal{F}$. They are subsequently engaged in collaborative reasoning, resulting in a set of intermediate outputs:

$$\mathcal{Y}_k = \{y_k^{(1)}, y_k^{(2)}, \dots\} = \mathcal{A}_k(T_k, \mathcal{Q}^*, \mathcal{F}) \tag{8}$$

To further improve efficiency and avoid unnecessary subtask generation, HALO integrates an early-stopping mechanism within the high-level planning agent $\mathcal{A}_{\text{plan}}$. This design is inspired by the Byzantine Consensus theory [40], which states that at least $3p + 1$ agents are required to tolerate p faulty agents in a single round of communication. Following this principle, HALO terminates the reasoning process if at least 66% of the completed subtasks in the execution history H_k yield a consistent answer $\hat{Y}$. Additionally, the reasoning process will also be terminated when the maximum number of permitted subtasks has been reached.

3.4 Workflow Search Engine

To dynamically explore multi-agent collaboration and adaptively construct the optimal workflow, HALO leverages a Workflow Search Engine module based on Monte Carlo Tree Search (MCTS) [13]. The workflow search process formalizes multi-agent inference as a tree-structured action space, where each node represents a role-specific agent $a_k^{(i)}$ executing a subtask T_k and each edge denotes communication transitions between agents. Together, these nodes and edges form a path that corresponds to a candidate reasoning trajectory for each subtask T_k. During each search iteration, HALO follows the standard four-stage optimized MCTS paradigm (Selection, Expansion, Simulation, and Backpropagation), as illustrated in Figure 2.

Selection. HALO recursively selects the best agent $a_k^{(i)}$ with the UCT algorithm [41] and adds it to the trajectory:

$$\text{UCT}(a_k^{(i)}) = \frac{v_k^{(i)}}{n_k^{(i)}} + \alpha \sqrt{\frac{\log N}{n_k^{(i)}}} \tag{9}$$

where $v_k^{(i)}$ is the score value of agent $a_k^{(i)}$, $n_k^{(i)}$ is the number of visits to the agent $a_k^{(i)}$, N is the number of visits to the parent agent $a_k^{(i-1)}$, and α is an exploration coefficient.

Expansion. Given a selected agent $a_k^{(i)}$ from the selection phase, if it contains untried actions, HALO expands the search tree by instantiating a new role-specific agent $a_k^{(i+1)}$ as its child.

Table 2: Performance comparison of HALO against competitive baselines across three benchmarks. Metrics include $pass@1$ (%) for HumanEval, $accuracy$ (%) for MMLU as well as MATH, and $Avg.$ (%) for the mean performance over three runs. All methods are executed with GPT-4o.

Baseline Type	Method	Structure	Benchmarks			Avg.
			HumanEval	MMLU	MATH	
Single-agent	ReAct [11]	Monolithic Sequential Reasoning Flow	69.1	57.6	29.2	52.0
Static MAS	CAMEL [12]	Feedback Triplet	72.4	64.3	31.9	56.2
	LLM-Debate [43]	Fully Connected Bipartite	73.7	66.3	32.6	57.5
Dynamic MAS	DyLAN [21]	DAG + Feedback Loop	81.7	70.1	35.2	62.3
	AgentVerse [33]	Hierarchical Tree	75.2	67.5	34.4	59.0
	ADAS [44]	Search-Based Dynamic Graph Structure	82.4	72.8	36.9	64.0
	HALO (Ours)	**Hierarchical Structure + MCTS**	**95.2**	**81.6**	**58.9**	**78.6**

Simulation. The agent $a_k^{(i+1)}$ initiates a simulated reasoning trajectory for subtask T_k, where a sequence of additional agents $\tilde{a}_k^{(i+2)}, \tilde{a}_k^{(i+3)}, \ldots$ are engaged to emulate hypothetical future steps along the reasoning path. Each simulated agent $\tilde{a}_k^{(j)}$ generates an intermediate output $\tilde{y}_k^{(j)}$:

$$\tilde{y}_k^{(j)} = \tilde{a}_k^{(j)}(T_k, \mathcal{Q}^*, \mathcal{F}) \tag{10}$$

The output $\tilde{y}_k^{(j)}$ is evaluated by two auxiliary agents. The judging agent assigns a status label $\tilde{\ell}_k^{(j)} \in \{success, fail, continue\}$, indicating whether the subtask is completed. The scoring agent computes a quality score value $\tilde{v}_k^{(j)} \in [0, 1]$ to reflect the effectiveness of the generated output (detail in Appendix B).

Backpropagation. Inspired by the value aggregation strategy in CoAT [42], HALO updates the evaluation scores of all traversed nodes along the search path by incorporating simulation feedback. To enhance sensitivity to task completion status, we additionally introduce a reward signal adjustment mechanism based on the judgment outcome.

Let $\lambda(\tilde{\ell}_k^{(j)})$ be the impact factor associated with the status label $\tilde{\ell}_k^{(j)}$, where λ reflects the reward or penalty of a simulation result. The updated value of a node $a_k^{(i)}$ is computed as follows:

$$v_k^{(i)*} = \frac{v_k^{(i)} \cdot n_k^{(i)} + \lambda(\tilde{\ell}_k^{(j\dagger)}) + \sum\limits_{\tilde{a}_k^{(j)} \in \text{Child}(a_k^{(i)})} \tilde{v}_k^{(j)}}{n_k^{(i)} + |\text{Child}(a_k^{(i)})|} \tag{11}$$

where $\tilde{\ell}_k^{(j\dagger)}$ denotes the terminal status label of the simulated leaf node agent $\tilde{a}_k^{(j\dagger)}$ along the reasoning trajectory and $\text{Child}(a_k^{(i)})$ refers to the set of its simulated children.

4 Experiments

4.1 Experimental setup

Code generation. We use the HumanEval [14] dataset, which contains 164 Python programming problems and corresponding unit tests. Unit tests are used to validate the correctness of generated codes. We report the $pass@1$ metric to assess code accuracy.

General reasoning. We use the MMLU [15] dataset, which spans 57 subjects with 15,908 questions. Each question is presented in multiple-choice format with four options. Due to the large number of questions, we randomly sample 13% of the total dataset according to the subject-wise distribution. We report the $accuracy$ metric to measure the correctness of answers.

Arithmetic reasoning. We use the MATH [16] dataset, which consists of 12,500 math problems across 7 subareas with 5 difficulty levels. Likewise, we randomly sample 500 math problems according to the subarea-wise and level-wise distribution. We report the $accuracy$ metric to measure the proportion of correct answers.

Table 3: Performance comparison on five abstract subjects selected from the MMLU dataset. Metrics are reported as $accuracy$ (%) averaged over three runs.

Baseline Type	Method	Subjects				
		Abstract Algebra	College Physics	Formal Logic	High School Mathematics	Moral Scenarios
Single-agent	ReAct [11]	44.2	46.5	40.3	44.7	37.6
Static MAS	CAMEL [12]	50.4	54.7	49.8	51.9	44.2
	LLM-Debate [43]	51.1	54.6	48.3	52.8	45.1
Dynamic MAS	DyLAN [21]	52.9	60.4	51.2	58.8	49.6
	AgentVerse [33]	53.6	59.4	50.3	57.1	48.7
	ADAS [44]	55.7	62.4	52.6	60.0	51.3
	HALO (Ours)	**69.7**	**77.3**	**66.8**	**75.5**	**64.6**

Baselines. We compare HALO with six competitive baselines across three categories, including single-agent frameworks (ReAct [11]), static multi-agent systems (CAMEL [12], LLM-Debate [43]), and dynamic multi-agent orchestration methods (DyLAN [21], AgentVerse [33], ADAS [44]).

HALO setup. We implement HALO and conduct these experiments by using GPT-4o [45], with the random seed of 10, temperature setting of 0.8, and max tokens limit of 2048. To ensure fair comparison, we use the same number of few-shot examples across all methods and merely equip HALO with the code interpreters as tools in the code generation task (detail in Appendix D).

4.2 Main results

We conduct extensive experiments to compare HALO against six baselines on three tasks and report the results based on GPT-4o in Table 2. Additionally, we present performances of executing five abstract subjects from the MMLU in Table 3 and three computationally intensive subareas from the MATH in Figure 3. Based on these results, we derive the following key observations.

HALO proposes the hierarchical reasoning architecture that overcomes the limitations of cognitive overload. Unlike frameworks such as ReAct [11], which require a single agent to simultaneously manage planning, reasoning, and reflection, HALO distributes these responsibilities across the hierarchical reasoning architecture dedicated to task decomposition, role instantiation, and subtask inference. This design enables more focused agent behavior and reduces cognitive overload. Specifically, as shown in Table 2, HALO achieves significant improvements over ReAct across all benchmarks, including a 26.1% gain in HumanEval pass@1 (95.2% vs. 69.1%), 24.0% in MMLU accuracy (81.6% vs. 57.6%), and 29.7% in MATH accuracy (58.9% vs. 29.2%). On average, HALO improves performance by 26.6% (78.6% vs. 52.0%), highlighting the advantages of hierarchical reasoning architecture over monolithic single-agent reasoning.

HALO leverages adaptive agent instantiation and search-based workflow exploration that enhance the granularity of task execution. Compared to static MAS approaches such as CAMEL [12] and LLM-Debate [43], which depend on fixed agent roles and handcrafted workflows, as well as dynamic MAS systems like DyLAN [21], AgentVerse [33], and ADAS [44], which lack fine-grained alignment between tasks and agents, HALO introduces adaptive agent instantiation and MCTS-driven workflow exploration. This design allows HALO to dynamically instantiate appropriate agent roles and iteratively refine execution trajectories based on real-time feedback. As a result in Table 2, HALO outperforms the strongest baseline (ADAS) by 14.6% on average (78.6% vs. 64.0%), and consistently achieves the highest performance across all benchmarks, with improvements of 12.8% on HumanEval pass@1 (95.2% vs. 82.4%), 8.8% on MMLU accuracy (81.6% vs. 72.8%), and 22.0% on MATH accuracy (58.9% vs. 36.9%). These results underscore the effectiveness of HALO in orchestrating multi-agent reasoning at scale.

HALO excels in handling highly complex and expert-level reasoning tasks. To further evaluate the capability of HALO in solving professionally demanding tasks, we conduct fine-grained comparisons on difficult subdomains from MMLU and MATH datasets, as reported in Table 3 and Figure 3. HALO consistently outperforms all baselines on these challenging problems. On the abstract MMLU subjects, HALO achieves substantial improvements over the strongest baseline (ADAS [44]), with an average accuracy of 70.8% compared to 56.4%, marking a 14.4% gain. Similarly, on the computationally intensive MATH subareas, HALO reaches an average accuracy of 43.9%, significantly surpassing

Figure 3: Performance comparison on three computationally intensive subareas selected from the MATH dataset. Metrics are reported as *accuracy* (%) averaged over three runs.

Figure 4: Ablation study of removing the Adaptive Prompt Refinement module and the high-level planning agent on GPT-4o across three benchmarks.

ADAS at 24.4%. These results demonstrate the strength of HALO in tackling highly specialized and non-trivial reasoning tasks.

4.3 Ablation study

To evaluate the impact of HALO components, we conduct ablation studies by individually removing components to measure performances on HumanEval, MMLU, and MATH. Results are summarized in Figure 4.

Effectiveness of Adaptive Prompt Refinement. We examine the necessity of the Adaptive Prompt Refinement module. By removing this component and directly passing the raw user query $\mathcal{Q}$ into downstream agents, we observe a clear degradation across all benchmarks. As shown in Figure 4, performance drops by 5.3% on average, with MMLU suffering the most (81.6% $\rightarrow$ 75.4%). This result highlights the importance of structured prompt construction in enhancing task understanding and aligning reasoning trajectories with user intent.

Effectiveness of the high-level planning agent. We remove the high-level planning agent and treat the refined user query $\mathcal{Q}^*$ as a single-step task without iterative decomposition. As shown in Figure 4, this leads to an average performance drop of 11.3% across all benchmarks. Notable drops on HumanEval (95.2% $\rightarrow$ 83.8%) and MATH (58.9% $\rightarrow$ 44.7%) demonstrate that removing task decomposition impairs reasoning coherence, highlighting its critical role in HALO.

5 Conclusion

In this work, we introduce HALO, a multi-agent collaboration framework dedicated to tackling complex interaction environments and expert-domain reasoning tasks. HALO functions in a three-stage paradigm. In the Adaptive Prompt Refinement, user queries are refined into structured prompts to enhance downstream reasoning. In the Hierarchical Reasoning Stack, HALO delegates downstream processes to three specialized layers, overcoming the limitations of cognitive overload. In the Workflow Search Engine, HALO leverages the search-based workflow exploration to construct optimal collaborative workflows. Experimental results demonstrate that HALO achieves significant performance improvements compared to the competitive baselines. Additionally, we find that performance can be further enhanced through the injection of long-term memory mechanisms and external knowledge integration, providing new directions for future work.

Acknowledgments

References

[1] A. El-Kishky, A. Wei, A. Saraiva, B. Minaiev, D. Selsam, D. Dohan, F. Song, H. Lightman, I. Clavera, J. Pachocki *et al.*, "Competitive programming with large reasoning models," *arXiv preprint arXiv:2502.06807*, 2025.

[2] DeepSeek-AI, D. Guo, D. Yang, H. Zhang, J. Song, R. Zhang, R. Xu, Q. Zhu, S. Ma, P. Wang, X. Bi, X. Zhang, X. Yu, Y. Wu, Z. F. Wu, Z. Gou, Z. Shao, Z. Li, Z. Gao, A. Liu, B. Xue, B. Wang, B. Wu, B. Feng, C. Lu, C. Zhao, C. Deng, C. Zhang, C. Ruan, D. Dai, D. Chen, D. Ji, E. Li, F. Lin, F. Dai, F. Luo, G. Hao, G. Chen, G. Li, H. Zhang, H. Bao, H. Xu, H. Wang, H. Ding, H. Xin, H. Gao, H. Qu, H. Li, J. Guo, J. Li, J. Wang, J. Chen, J. Yuan, J. Qiu, J. Li, J. L. Cai, J. Ni, J. Liang, J. Chen, K. Dong, K. Hu, K. Gao, K. Guan, K. Huang, K. Yu, L. Wang, L. Zhang, L. Zhao, L. Wang, L. Zhang, L. Xu, L. Xia, M. Zhang, M. Zhang, M. Tang, M. Li, M. Wang, M. Li, N. Tian, P. Huang, P. Zhang, Q. Wang, Q. Chen, Q. Du, R. Ge, R. Zhang, R. Pan, R. Wang, R. J. Chen, R. L. Jin, R. Chen, S. Lu, S. Zhou, S. Chen, S. Ye, S. Wang, S. Yu, S. Zhou, S. Pan, S. S. Li, S. Zhou, S. Wu, S. Ye, T. Yun, T. Pei, T. Sun, T. Wang, W. Zeng, W. Zhao, W. Liu, W. Liang, W. Gao, W. Yu, W. Zhang, W. L. Xiao, W. An, X. Liu, X. Wang, X. Chen, X. Nie, X. Cheng, X. Liu, X. Xie, X. Liu, X. Yang, X. Li, X. Su, X. Lin, X. Q. Li, X. Jin, X. Shen, X. Chen, X. Sun, X. Wang, X. Song, X. Zhou, X. Wang, X. Shan, Y. K. Li, Y. Q. Wang, Y. X. Wei, Y. Zhang, Y. Xu, Y. Li, Y. Zhao, Y. Sun, Y. Wang, Y. Yu, Y. Zhang, Y. Shi, Y. Xiong, Y. He, Y. Piao, Y. Wang, Y. Tan, Y. Ma, Y. Liu, Y. Guo, Y. Ou, Y. Wang, Y. Gong, Y. Zou, Y. He, Y. Xiong, Y. Luo, Y. You, Y. Liu, Y. Zhou, Y. X. Zhu, Y. Xu, Y. Huang, Y. Li, Y. Zheng, Y. Zhu, Y. Ma, Y. Tang, Y. Zha, Y. Yan, Z. Z. Ren, Z. Ren, Z. Sha, Z. Fu, Z. Xu, Z. Xie, Z. Zhang, Z. Hao, Z. Ma, Z. Yan, Z. Wu, Z. Gu, Z. Zhu, Z. Liu, Z. Li, Z. Xie, Z. Song, Z. Pan, Z. Huang, Z. Xu, Z. Zhang, and Z. Zhang, "Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning," 2025. [Online]. Available: https://arxiv.org/abs/2501.12948

[3] L. Zhong, Z. Wang, and J. Shang, "Debug like a human: A large language model debugger via verifying runtime execution step-by-step," *arXiv preprint arXiv:2402.16906*, 2024.

[4] N. Shinn, F. Cassano, A. Gopinath, K. R. Narasimhan, and S. Yao, "Reflexion: language agents with verbal reinforcement learning," in *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. [Online]. Available: https://openreview.net/forum?id=vAElhFcKW6

[5] J. Wang, H. Xu, H. Jia, X. Zhang, M. Yan, W. Shen, J. Zhang, F. Huang, and J. Sang, "Mobile-agent-v2: Mobile device operation assistant with effective navigation via multi-agent collaboration," *arXiv preprint arXiv:2406.01014*, 2024.

[6] Y. Li, C. Zhang, W. Yang, B. Fu, P. Cheng, X. Chen, L. Chen, and Y. Wei, "Appagent v2: Advanced agent for flexible mobile interactions," *arXiv preprint arXiv:2408.11824*, 2024.

[7] G. Wang, Y. Xie, Y. Jiang, A. Mandlekar, C. Xiao, Y. Zhu, L. Fan, and A. Anandkumar, "Voyager: An open-ended embodied agent with large language models," *arXiv preprint arXiv:2305.16291*, 2023.

[8] R. Li, C. Xu, Z. Guo, B. Fan, R. Zhang, W. Liu, Y. Zhao, W. Gong, and E. Wang, "Ai-vqa: visual question answering based on agent interaction with interpretability," in *Proceedings of the 30th ACM International Conference on Multimedia*, 2022, pp. 5274–5282.

[9] Z. Wang, H. Zhang, C.-L. Li, J. M. Eisenschlos, V. Perot, Z. Wang, L. Miculicich, Y. Fujii, J. Shang, C.-Y. Lee *et al.*, "Chain-of-table: Evolving tables in the reasoning chain for table understanding," *arXiv preprint arXiv:2401.04398*, 2024.

[10] Q. Zeng, Q. Yang, S. Dong, H. Du, L. Zheng, F. Xu, and Y. Li, "Perceive, reflect, and plan: Designing llm agent for goal-directed city navigation without instructions," *arXiv preprint arXiv:2408.04168*, 2024.

[11] S. Yao, J. Zhao, D. Yu, N. Du, I. Shafran, K. Narasimhan, and Y. Cao, "React: Synergizing reasoning and acting in language models," in *International Conference on Learning Representations (ICLR)*, 2023.

[12] G. Li, H. Hammoud, H. Itani, D. Khizbullin, and B. Ghanem, "Camel: Communicative agents for" mind" exploration of large language model society," *Advances in Neural Information Processing Systems*, vol. 36, pp. 51 991–52 008, 2023.

[13] C. B. Browne, E. Powley, D. Whitehouse, S. M. Lucas, P. I. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, and S. Colton, "A survey of monte carlo tree search methods," *IEEE Transactions on Computational Intelligence and AI in games*, vol. 4, no. 1, pp. 1–43, 2012.

[14] M. Chen, J. Tworek, H. Jun, Q. Yuan, H. P. D. O. Pinto, J. Kaplan, H. Edwards, Y. Burda, N. Joseph, G. Brockman *et al.*, "Evaluating large language models trained on code," *arXiv preprint arXiv:2107.03374*, 2021.

[15] D. Hendrycks, C. Burns, S. Basart, A. Zou, M. Mazeika, D. Song, and J. Steinhardt, "Measuring massive multitask language understanding," *arXiv preprint arXiv:2009.03300*, 2020.

[16] D. Hendrycks, C. Burns, S. Kadavath, A. Arora, S. Basart, E. Tang, D. Song, and J. Steinhardt, "Measuring mathematical problem solving with the math dataset," *arXiv preprint arXiv:2103.03874*, 2021.

[17] J. Shen, J. J. Dudley, J. Zheng, B. Byrne, and P. O. Kristensson, "Promptor: A conversational and autonomous prompt generation agent for intelligent text entry techniques," 2023. [Online]. Available: https://arxiv.org/abs/2310.08101

[18] X. Xue, Z. Lu, D. Huang, Z. Wang, W. Ouyang, and L. Bai, "Comfybench: Benchmarking llm-based agents in comfyui for autonomously designing collaborative ai systems," 2024. [Online]. Available: https://arxiv.org/abs/2409.01392

[19] S. Hong, X. Zheng, J. Chen, Y. Cheng, J. Wang, C. Zhang, Z. Wang, S. K. S. Yau, Z. Lin, L. Zhou *et al.*, "Metagpt: Meta programming for multi-agent collaborative framework," *arXiv preprint arXiv:2308.00352*, vol. 3, no. 4, p. 6, 2023.

[20] J. Ruan, Y. Chen, B. Zhang, Z. Xu, T. Bao, H. Mao, Z. Li, X. Zeng, R. Zhao *et al.*, "Tptu: Task planning and tool usage of large language model-based ai agents," in *NeurIPS 2023 Foundation Models for Decision Making Workshop*, 2023.

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

[22] S. Schmidgall, Y. Su, Z. Wang, X. Sun, J. Wu, X. Yu, J. Liu, Z. Liu, and E. Barsoum, "Agent laboratory: Using llm agents as research assistants," *arXiv preprint arXiv:2501.04227*, 2025.

[23] Y. Wang, L. Yang, G. Li, M. Wang, and B. Aragam, "Scoreflow: Mastering llm agent workflows via score-based preference optimization," *arXiv preprint arXiv:2502.04306*, 2025.

[24] S. Fan, X. Cong, Y. Fu, Z. Zhang, S. Zhang, Y. Liu, Y. Wu, Y. Lin, Z. Liu, and M. Sun, "Workflowllm: Enhancing workflow orchestration capability of large language models," *arXiv preprint arXiv:2411.05451*, 2024.

[25] B. Liu, X. Li, J. Zhang, J. Wang, T. He, S. Hong, H. Liu, S. Zhang, K. Song, K. Zhu *et al.*, "Advances and challenges in foundation agents: From brain-inspired intelligence to evolutionary, collaborative, and safe systems," *arXiv preprint arXiv:2504.01990*, 2025.

[26] W. Hua, O. Liu, L. Li, A. Amayuelas, J. Chen, L. Jiang, M. Jin, L. Fan, F. Sun, W. Wang, X. Wang, and Y. Zhang, "Game-theoretic llm: Agent workflow for negotiation games," 2024. [Online]. Available: https://arxiv.org/abs/2411.05990

[27] J. Zhang, J. Xiang, Z. Yu, F. Teng, X. Chen, J. Chen, M. Zhuge, X. Cheng, S. Hong, J. Wang, B. Zheng, B. Liu, Y. Luo, and C. Wu, "Aflow: Automating agentic workflow generation," 2025. [Online]. Available: https://arxiv.org/abs/2410.10762

[28] Y. Yu, Z. Yao, H. Li, Z. Deng, Y. Jiang, Y. Cao, Z. Chen, J. Suchow, Z. Cui, R. Liu *et al.*, "Fincon: A synthesized llm multi-agent system with conceptual verbal reinforcement for enhanced financial decision making," *Advances in Neural Information Processing Systems*, vol. 37, pp. 137 010–137 045, 2024.

[29] L. Buşoniu, R. Babuška, and B. De Schutter, "Multi-agent reinforcement learning: An overview," *Innovations in multi-agent systems and applications-1*, pp. 183–221, 2010.

[30] I. Arel, C. Liu, T. Urbanik, and A. G. Kohls, "Reinforcement learning-based multi-agent system for network traffic signal control," *IET Intelligent Transport Systems*, vol. 4, no. 2, pp. 128–135, 2010.

[31] K. Zhang, Z. Yang, and T. Başar, "Multi-agent reinforcement learning: A selective overview of theories and algorithms," *Handbook of reinforcement learning and control*, pp. 321–384, 2021.

[32] S. Kapetanakis and D. Kudenko, "Reinforcement learning of coordination in heterogeneous cooperative multi-agent systems," in *Symposium on Adaptive Agents and Multi-agent Systems*. Springer, 2003, pp. 119–131.

[33] W. Chen, Y. Su, J. Zuo, C. Yang, C. Yuan, C.-M. Chan, H. Yu, Y. Lu, Y.-H. Hung, C. Qian, Y. Qin, X. Cong, R. Xie, Z. Liu, M. Sun, and J. Zhou, "Agentverse: Facilitating multi-agent collaboration and exploring emergent behaviors," 2023. [Online]. Available: https://arxiv.org/abs/2308.10848

[34] J. Wei, X. Wang, D. Schuurmans, M. Bosma, F. Xia, E. Chi, Q. V. Le, D. Zhou *et al.*, "Chain-of-thought prompting elicits reasoning in large language models," *Advances in neural information processing systems*, vol. 35, pp. 24 824–24 837, 2022.

[35] X. Wang, J. Wei, D. Schuurmans, Q. Le, E. Chi, S. Narang, A. Chowdhery, and D. Zhou, "Self-consistency improves chain of thought reasoning in language models," *arXiv preprint arXiv:2203.11171*, 2022.

[36] D. Zhou, N. Schärli, L. Hou, J. Wei, N. Scales, X. Wang, D. Schuurmans, C. Cui, O. Bousquet, Q. Le *et al.*, "Least-to-most prompting enables complex reasoning in large language models," *arXiv preprint arXiv:2205.10625*, 2022.

[37] S. Yao, D. Yu, J. Zhao, I. Shafran, T. Griffiths, Y. Cao, and K. Narasimhan, "Tree of thoughts: Deliberate problem solving with large language models," *Advances in neural information processing systems*, vol. 36, pp. 11 809–11 822, 2023.

[38] T. Kojima, S. S. Gu, M. Reid, Y. Matsuo, and Y. Iwasawa, "Large language models are zero-shot reasoners," *Advances in neural information processing systems*, vol. 35, pp. 22 199–22 213, 2022.

[39] X. Hou, Y. Zhao, S. Wang, and H. Wang, "Model context protocol (mcp): Landscape, security threats, and future research directions," 2025. [Online]. Available: https://arxiv.org/abs/2503.23278

[40] M. Castro, B. Liskov *et al.*, "Practical byzantine fault tolerance," in *OsDI*, vol. 99, no. 1999, 1999, pp. 173–186.

[41] L. Kocsis and C. Szepesvári, "Bandit based monte-carlo planning," in *European conference on machine learning*. Springer, 2006, pp. 282–293.

[42] J. Pan, S. Deng, and S. Huang, "Coat: Chain-of-associated-thoughts framework for enhancing large language models reasoning," *arXiv preprint arXiv:2502.02390*, 2025.

[43] Y. Du, S. Li, A. Torralba, J. B. Tenenbaum, and I. Mordatch, "Improving factuality and reasoning in language models through multiagent debate," in *Forty-first International Conference on Machine Learning*, 2023.

[44] B. Liu, X. Li, J. Zhang, J. Wang, T. He, S. Hong, H. Liu, S. Zhang, K. Song, K. Zhu, Y. Cheng, S. Wang, X. Wang, Y. Luo, H. Jin, P. Zhang, O. Liu, J. Chen, H. Zhang, Z. Yu, H. Shi, B. Li, D. Wu, F. Teng, X. Jia, J. Xu, J. Xiang, Y. Lin, T. Liu, T. Liu, Y. Su, H. Sun, G. Berseth, J. Nie, I. Foster, L. Ward, Q. Wu, Y. Gu, M. Zhuge, X. Tang, H. Wang, J. You, C. Wang, J. Pei, Q. Yang, X. Qi, and C. Wu, "Advances and challenges in foundation agents: From brain-inspired intelligence to evolutionary, collaborative, and safe systems," 2025. [Online]. Available: https://arxiv.org/abs/2504.01990

[45] J. Achiam, S. Adler, S. Agarwal, L. Ahmad, I. Akkaya, F. L. Aleman, D. Almeida, J. Altenschmidt, S. Altman, S. Anadkat *et al.*, "Gpt-4 technical report," *arXiv preprint arXiv:2303.08774*, 2023.

A The system prompt of Adaptive Prompt Refinement

Adaptive Prompt Refinement

Task Parser Agent

You are a Task Parser Agent. Your role is to analyze user queries and extract the underlying task type, main objective, and any key details that will guide the completion of the task.

Inputs
You will be given these fields:
- "User Query": The user prompt containing user's intent.

Outputs
Your output should be clear, structured, and focused on the following points:
1. **Task Type**: The general category of the task (e.g., data analysis, text generation, image processing).
2. **Core Intent**: The main goal or purpose of the task (e.g., "analyze trends," "generate summary," "predict outcomes").
3. **Key Details**: Any specific instructions or constraints that are important for task execution (e.g., "for the last 3 months," "in Python," "based on given data").

Your response should be in the following format:
```
{
  "Task Type": "<task_type>",
  "Core Intent": "<core_intent>",
  "Key Details": "<key_details>"
}
```

Prompt Template Agent

You are a Prompt Template Generator Agent. Your role is to generate a structured prompt template based on the task type, core intent, and key details provided by the Task Parser Agent. Your response should help guide the completion of the task effectively and should be tailored to the specified task requirements.

Inputs
You will be given these fields:
- "User Query": The user prompt containing user's intent.
- "Task Type": The general category of the task.
- "Core Intent": The main goal or purpose of the task.
- "Key Details": Any specific instructions or constraints that are important for task execution.

Outputs
The structure of the output should follow the format below:
1. **Task Type**: The general category of the task as identified by Task Parser Agent (e.g., data analysis, text generation, image processing).
2. **Core Intent**: The main objective or purpose of the task as defined by Task Parser Agent (e.g., "analyze trends," "generate summary," "predict outcomes").
3. **Key Details**: Any important constraints or instructions for executing the task, as identified by Task Parser Agent (e.g., "for the last 3 months," "in Python," "based on given data").
4. **Generated Prompt Template**: Construct a detailed template in Python string format that will guide the task's completion. The template should include placeholders for the key details identified, ensuring that it is general enough for reuse and clear for execution.

The generated prompt template should follow this example structure:
```
{
  "Task Type": "<task_type>",
  "Core Intent": "<core_intent>",
  "Key Details": "<key_details>",
  "Generated Prompt Template": "<template_string>"
}
```

Prompt Optimization Agent

You are a Prompt Optimizer Agent. Your role is to refine and enhance the prompt template generated by the Prompt Template Generator Agent. Your task is to ensure that the generated prompt template is clear, concise, and optimized for effective task completion. You should focus on improving clarity, precision, and usability without changing the core intent or task details.

Inputs
You will be given these fields:
- "User Query": The user prompt containing user's intent.
- "Task Type": The general category of the task.
- "Core Intent": The main goal or purpose of the task.
- "Key Details": Any specific instructions or constraints that are important for task execution.
- "Generated Prompt Template": A detailed template in Python string format that will guide the task's completion.

Outputs
The structure of your output should follow the format below:
1. **Task Type**: The general category of the task as identified by the Task Parser Agent.
2. **Core Intent**: The main objective or purpose of the task as defined by the Task Parser Agent.
3. **Key Details**: Any important constraints or instructions for executing the task, as identified by the Task Parser Agent.
4. **Optimized Prompt Template**: The optimized version of the prompt template provided by the Prompt Template Generator Agent. This should include any necessary improvements, such as simplifying language, clarifying instructions, and removing unnecessary complexity.

Your output should be a Python string containing the optimized template. The format should look like this:
```
{
  "Task Type": "<task_type>",
  "Core Intent": "<core_intent>",
  "Key Details": "<key_details>",
  "Optimized Prompt Template": "<optimized_template_string>"
}
```

Prompt Generator Agent

You are a Final Prompt Generator Agent. Your role is to integrate and finalize the optimized prompt template provided by the Prompt Optimizer Agent. Your task is to ensure that the final prompt is structured, clear, and perfectly tailored to the task requirements. You should ensure the prompt is effective for guiding task completion and meets the specifications provided by the previous agents.

Inputs
You will be given these fields:
- "User Query": The user prompt containing user's intent.
- "Task Type": The general category of the task.
- "Core Intent": The main goal or purpose of the task.
- "Key Details": Any specific instructions or constraints that are important for task execution.
- "Optimized Prompt Template": The optimized version of the prompt template provided by the Prompt Template Generator Agent.

Outputs
The structure of your output should follow the format below:
1. **Task Type**: The general category of the task as identified by Task Parser Agent.
2. **Core Intent**: The main objective or purpose of the task as defined by Task Parser Agent.
3. **Key Details**: Any important constraints or instructions for executing the task, as identified by Task Parser Agent.
4. **Final Prompt**: The final version of the prompt, generated based on the optimized template. This should be a Python string that is ready for execution and should be clearly formatted for the task at hand.

Your output should be a Python string containing the final prompt. The format should look like this:
```
{
  "Task Type": "<task_type>",
  "Core Intent": "<core_intent>",
  "Key Details": "<key_details>",
  "Final Prompt": "<final_prompt_string>"
}
```

Figure 5: System prompts used in the Adaptive Prompt Refinement module. The refinement process is conducted through four specialized agents: the *Task Parser Agent* extracts task semantics from user queries; the *Prompt Template Agent* constructs a structured prompt template; the *Prompt Optimization Agent* enhances clarity and usability; and the *Prompt Generator Agent* produces the final prompt.

B   The system prompt of the planning agent and Workflow Search Engine

High-Level Planning Agent

Task Decomposition Agent

You are a Task Decomposition Agent. Your job is to break a user's overall task into a sequence of clear, actionable subtasks—one at a time.

Inputs
You will be given these fields:
- "User Query": the original prompt from the user
- "Task Type": the category of the user's request
- "Core Intent": the user's main goal
- "Key Details": any important constraints or context
- "Implementation history of subtasks": Implementation of past subtasks. Maybe contain "subtask name", "subtask answer", "subtask result".

Your Objective
1. Analyze the provided fields and the progress so far.
2. Generate exactly one new subtask that advances toward fulfilling the Core Intent.
3. Make the subtask as specific and actionable as possible.
4. If the overall task is already complete (no further decomposition needed), reply with exactly: stop.

Output Format
Return either:
- A JSON object containing a single key `"next subtask"` whose value is your new subtask description, for example:
```json
{
  "next subtask": "<description of the next actionable subtask>"
}
```

- Or if no more subtasks are required, The value of a single key `"next subtask"` should be `"stop"`, for example:
```json
{
  "next subtask": "stop"
}
```

Guidelines
- Place instructions first.
- Use triple-quote delimiters around this prompt.
- Be concise, precise, and unambiguous.
- Only produce one subtask per response.
- Do not include any commentary or extra fields.

Workflow Search Engine

Scoring Agent

You are a **Score Agent** whose sole responsibility is to evaluate the output generated by a role-specific agent for a given subtask and assign a numeric quality score between **0** and **1** (inclusive), with a preference for smoother, non-extreme values across the scale.

Input:
You will be given these fields:
- "Task Type": the category of the user's overall request
- "Core Intent": the user's main goal
- "Key Details": any important constraints or context
- "User Query": the original prompt from the user
- "Current Subtask": the specific subtask being executed
- "Agent Role": the role of the agent whose output you must score
- "Output of the Agent Role": the actual result produced by that agent
- "Judge Agent Output": the output of the Judge Agent, which indicates whether the subtask was a success, fail, or continue.

Your Task:
1. Assess how well the **Output of the Agent Role** satisfies the **Current Subtask** in light of the **Task Type**, **Core Intent**, and **Key Details**.
2. Consider correctness, completeness, relevance, and adherence to the specified role.
3. Carefully evaluate the **Judge Agent Output** to give appropriate incentives, not to give incentives, or to give penalties based on the performance of the agent.
4. Provide a quality score between **0** and **1**. The score should not be biased toward extreme values (0 or 1). It should reflect the degree to which the output meets the task requirements in a balanced way.

Output Format:
Return **only** a JSON object with a single key `"score"` whose value is a decimal number between **0** and **1**, rounded to up to four decimal places. Do **not** include any additional keys, commentary, or formatting.

Example:
Input:
"Task Type": "data processing",
"Core Intent": "prepare data for model training",
"Key Details": "normalize all features to [0,1]",
"User Query": "Normalize feature columns",
"Current Subtask": "data normalization",
"Agent Role": "min-max normalizer",
"Output of the Agent Role": "applied min-max to each column, but used [−1,1] scaling"
"Judge Agent Output": "continue"

Correct Output:
```json
{
  "score": 0.55
}
```

Judging Agent

You are a **Judge Agent** whose responsibility is to determine whether a role-specific agent's output has completed the given subtask. Your evaluation must result in one of three statuses: `"success"`, `"fail"`, or `"continue"`.

Input:
You will be given these fields:
- "Task Type": the category of the user's overall request
- "Core Intent": the user's main goal
- "Key Details": any important constraints or context
- "User Query": the original prompt from the user
- "Current Subtask": the specific subtask being executed
- "Agent Role": the role of the agent whose output you must judge
- "Output of the Agent Role": the actual result produced by that agent

Your Task:
1. Assess whether the **Output of the Agent Role** fully satisfies the **Current Subtask** in light of the **Task Type**, **Core Intent**, and **Key Details**.
2. If the subtask is correctly and completely done, return `"success"`.
3. If the subtask cannot be completed based on this output (e.g., wrong method or errors), return `"fail"`.
4. If the subtask is partially completed or needs further refinement before moving on, return `"continue"`.

Output Format:
Return **only** a JSON object with a single key `"status"` whose value is one of `"success"`, `"fail"`, or `"continue"`. Do **not** include any extra keys, commentary, or formatting.

Example:
Input:
"Task Type": "data processing",
"Core Intent": "prepare data for model training",
"Key Details": "normalize all features to [0,1]",
"User Query": "Normalize feature columns",
"Current Subtask": "data normalization",
"Agent Role": "min-max normalizer",
"Output of the Agent Role": "applied min-max scaling to each column with correct [0,1] range"

Correct Output:
```json
{
  "status": "success"
}
```

Figure 6: System prompts for the high-Level planning agent and Workflow Search Engine module, including the Task Decomposition Agent, Scoring Agent, and Judging Agent.

C The system prompt of mid-level role-design agents

Agent Role Assignment

Agent Generator for Subtask

You are an Agent Generation Expert. Your task is to generate a list of agent roles that would be required to complete a specific subtask based on the provided input.

Input:
You will be given these fields:
- "Task Type": the category of the user's request
- "Core Intent": the user's main goal
- "Key Details": any important constraints or context
- "User Query": the original prompt from the user
- "Current Subtask": A subtask that will be implemented.

Your Objective:
1. Based on the 'Current Subtask', generate a list of possible agents' roles that would help solve this subtask.
2. Consider the context provided in the 'Task Type', 'Core Intent', 'Key Details', and 'User Query' to ensure that the generated roles are highly relevant.
3. You can generate multiple agent roles if the subtask requires it (e.g., for a data processing task, the roles could be 'Data Cleaning', 'Data Normalization', etc.).
5. Ensure that the roles are clearly defined and can be directly used to guide other agents in performing their tasks.

Output:
Return a JSON object with the following format:

```json
{
  "agent roles": ["<role_1>", "<role_2>", ...]
}
```

Prompt Template Generator for Subtask

You are a Prompt Template Generator Agent. Your role is to create a clear, structured, and reusable prompt template for an agent based on the provided subtask and agent role. The template should guide the agent's behavior precisely and include placeholders for any dynamic details.

Inputs
You will be given these fields:
- "Subtask": The specific subtask instructions.
- "Agent Role": The role description of a specific agent.

Outputs
Your output must be a valid Python string representing a JSON-like object with the following keys:
1. **Agent Role**: the role description (e.g., "Data Cleaning Agent").
2. **Subtask**: the specific subtask instructions (e.g., "Remove duplicate entries from the sales dataset.").
3. **Prompt Template**: a detailed template string that will instruct the agent how to perform the subtask, incorporating placeholders for any inputs or parameters (for example, `<input_data>`, `<parameters>`).

Example format:
{
  "Agent Role": "<agent_role>",
  "Subtask": "<subtask_description>",
  "Prompt Template": "You are a <agent_role>. Your task is to <subtask_description>. Use <input_data> and follow these rules: <guidelines>."
}

Prompt Optimizer for Subtask

You are a Prompt Optimizer Agent. Your role is to refine and enhance the prompt template created by the Prompt Template Generator Agent for a specific subtask and agent role. Your goal is to improve clarity, precision, and effectiveness of the template without altering the core task requirements.

Inputs
You will be given these fields:
- "Subtask": The specific subtask instructions.
- "Agent Role": The role description of a specific agent.
- "Original Prompt Template": A detailed template string that will instruct the agent how to perform the subtask.

Outputs
Your output must be a valid Python string representing a JSON-like object with the following keys:
1. **Agent Role**: the role description (e.g., "Data Cleaning Agent").
2. **Subtask**: the specific subtask instructions (e.g., "Remove duplicate entries from the sales dataset.").
3. **Optimized Prompt Template**: your improved version of the prompt template, with clearer language and precise instructions.

Example format:
{
  "Agent Role": "<agent_role>",
  "Subtask": "<subtask_description>",
  "Optimized Prompt Template": "<optimized_template>"
}

Final Prompt Generator for Subtask

You are a Final Prompt Generator Agent. Your role is to integrate the optimized prompt template provided by the Prompt Optimizer Agent for a specific agent role and subtask, and produce the final system prompt that the agent will execute. The final prompt should be clear, actionable, and self-contained.

Inputs
You will be given these fields:
- "Subtask": The specific subtask instructions.
- "Agent Role": The role description of a specific agent.
- "Optimized Prompt Template": Improved version of the prompt template, with clearer language and precise instructions.

Outputs
Your output must be a valid Python string representing a JSON-like object with the following keys:
1. **Agent Role**: the role description (e.g., "Data Cleaning Agent").
2. **Subtask**: the specific subtask instructions (e.g., "Remove duplicate entries from the sales dataset.").
3. **Final Prompt**: the final prompt string that the agent should receive, incorporating the optimized template exactly as intended for execution.

Example format:
{
  "Agent Role": "<agent_role>",
  "Subtask": "<subtask_description>",
  "Final Prompt": "You are a <agent_role>. Your task is to <subtask_description>. <any additional instructions>."
}

Figure 7: System prompts for mid-level role-design agents, including role generation and prompt construction for subtask-specific agents.

D The prompt of experiments

Experiments

HumanEval Query Prefix

Now you will generate a function according to the following description. Remember **not** to include the function signature in your answer. Do **not** include any other text in your answer.

Attention: just generate coding, you are not required to test the code.

Coding prompt is as follows:
```python
{}
```

MMLU Query Prefix

Now can you answer the following question as accurately as possible? question:
```
{}
```
options:
```
{}
```
your answer:

MATH Few Shot

[example 1]
Problem:
```
Kevin Kangaroo begins hopping on a number line at 0. He wants to get to 1, but he can hop only $\frac{1}{3}$ of the distance. Each hop tires him out so that he continues to hop $\frac{1}{3}$ of the remaining distance. How far has he hopped after five hops? Express your answer as a common fraction.
```
Answer:
```
Let's think step by step
Kevin hops $1/3$ of the remaining distance with every hop.
His first hop takes $1/3$ closer.
For his second hop, he has $2/3$ left to travel, so he hops forward $(2/3)(1/3)$.
For his third hop, he has $(2/3)^2$ left to travel, so he hops forward $(2/3)^2(1/3)$.
In general, Kevin hops forward $(2/3)^{k-1}(1/3)$ on his $k$th hop.
We want to find how far he has hopped after five hops.
This is a finite geometric series with first term $1/3$, common ratio $2/3$, and five terms.
Thus, Kevin has hopped $\frac{\frac{1}{3}\left(1-\left(\frac{2}{3}\right)^5\right)}{1-\frac{2}{3}} = \boxed{\frac{211}{243}}$.
The answer is \frac{211}{243}
```

[example 2]
Problem:
```
What is the area of the region defined by the equation $x^2+y^2 - 7 = 4y-14x+3$?
```
Answer:
```
Let's think step by step
We rewrite the equation as $x^2 + 14x + y^2 - 4y = 10$ and then complete the square,
resulting in  $(x+7)^2-49 + (y-2)^2-4=10$,
or $(x+7)^2+(y-2)^2=63$.
This is the equation of a circle with center $(-7, 2)$ and radius $\sqrt{63},$
so the area of this region is $\pi r^2 = \boxed{63\pi}$.
The answer is 63\pi
```

[example 3]
Problem:
```
If $x^2+y^2=1$, what is the largest possible value of $|x|+|y|$?
```
Answer:
```
Let's think step by step
If $(x,y)$ lies on the circle,
so does $(x,-y),$ $(-x,-y),$ and $(-x,-y),$ (which all give the same value of $|x| + |y|$),
so we can assume that $x \ge 0$ and $y \ge 0.$
Then $|x| + |y| = x + y.$ Squaring, we get
\[(x + y)^2 = x^2 + 2xy + y^2 = 1 + 2xy.\]
Note that $(x - y)^2 \ge 0.$
Expanding, we get $x^2 - 2xy + y^2 \ge 0,$ so $2xy \le x^2 + y^2 = 1.$
Hence,\[1 + 2xy \le 2,\]which means $x + y \le \sqrt{2}.$
Equality occurs when $x = y = \frac{1}{\sqrt{2}},$
so the maximum value of $|x| + |y|$ is $\boxed{\sqrt{2}}.$
The answer is \sqrt{2}
```

MATH Few Shot Prefix

Follow the given examples and answer the mathematics problem:
{}

Now please follow the above given examples and answer the mathematics problem:
Problem:
```
{}
```
Your answer:

Figure 8: Prompts used for HumanEval, MMLU, and MATH experiments, including query prefixes and few-shot examples.