MegaAgent: Dynamic Agent Coordination Without SOPs

Anonymous ACL submission

Abstract

LLM-powered multi-agent (LLM-MA) systems have shown promise in tackling complex tasks. However, existing solutions often suffer from limited agent coordination and heavy reliance on predefined Standard Operating Procedures (SOPs), which demand extensive human input. To address these limitations, we propose *MegaAgent*, a framework designed for autonomous coordination in LLM-MA systems. *MegaAgent* generates agents based on task complexity and enables dynamic task decomposition, parallel execution, efficient communication, and comprehensive system monitoring of agents. In evaluations, *MegaAgent* demonstrates exceptional performance, successfully developing a Gobang game within 800 seconds and scaling up to 590 agents in a national policy simulation to generate multi-domain policies. It significantly outperforms existing systems, such as MetaGPT, in both task completion efficiency and scalability. By eliminating the need for predefined SOPs, *MegaAgent* demonstrates exceptional scalability and autonomy, setting a foundation for advancing true autonomy in LLM-MA systems.[1]

1 Introduction

The remarkable planning and cognitive capabilities of Large Language Models (LLMs) (Touvron et al., 2023; Zhu et al., 2023) have spurred significant interest in LLM-based multi-agent (LLM-MA) systems (Wu et al., 2023b; Chen et al., 2023b; Hong et al., 2023), which coordinate multiple LLM agents to address complex tasks. For example, MetaGPT introduces a meta-programming framework to simulate the software development process (Hong et al., 2023), while Simulacra (Park et al., 2023) models social interactions among 25 LLM-powered agents in a simulated town, showcasing the potential of these systems to replicate real-world dynamics. The demand for large-scale social simulation applications, such as social media and war simulations (Gao et al., 2023a; Hua et al., 2023; Jin et al., 2024), is driving the development of LLM-MA systems capable of simulating complex real-world scenarios.

[1]Code is available at `https://anonymous.4open.science/r/MegaAgent-dev-DEF0`

However, existing LLM-MA frameworks have two limitations. (1) They fail to achieve adaptive task coordination when the task is big and complex e.g. generating hundreds of agents for a social simulation; and do not consider the coordination between a large scale of agents. (2) Most systems heavily depend on user-defined configurations, including predefined agent roles, standard operating procedures (SOPs), and static communication graphs (Hong et al., 2023; Chen et al., 2023b; Wu et al., 2023b). This approach limits flexibility and requires significant human effort when deploying numerous agents to complete a task.

Addressing the above limitations presents the following key challenges: (1) **Facilitating adaptive and effective communication among agents and with external file systems.** As tasks grow in complexity and scale, managing communication becomes increasingly difficult, especially when incorporating parallelism and coordinating multiple agents across different rounds of communication (Zhang et al., 2024b). (2) **Ensuring that each agent completes its task accurately without relying on predefined SOPs.** LLM agents often generate hallucinated outputs (Huang et al., 2023b) or fail to complete tasks correctly within a single round (Liu et al., 2023a; Andriushchenko et al., 2024), necessitating robust mechanisms to ensure reliability and correctness. This is particularly critical in multi-agent systems, where hallucinations can propagate and compromise the entire system's performance (Zhang et al., 2024b; Lee and Tiwari, 2024).

Drawing inspiration from Operating Systems (OS), where processes and threads efficiently manage tasks through: (1) generating multiple threads within a process to complete a task, and (2) enabling different processes to operate in parallel, we propose *MegaAgent* to address the aforementioned limitations. *MegaAgent* decomposes large tasks into multiple hierarchical subtasks (analogous to processes), with each subtask completed by a dedicated group of agents (similar to threads). Communication occurs either within agent groups or between them as needed, resembling inter-process communication in an OS. **Users simply need to provide a meta prompt to *Boss Agent*, after which the task is autonomously completed.** The novelty comparison between *MegaAgent* and popular baselines is in Table 1. Details are in Table 9. An overview of *MegaAgent* is shown in Figure 1. We equip *MegaAgent* with the following two strategies to tackle the above challenges:

(1) **Hierarchical Task Management:** To facilitate adaptive task handling and effective communication, *MegaAgent* employs a hierarchical task management mechanism structured across three levels. First, the *Boss Agent Level Task Decomposition*, where the Boss Agent divides a task into smaller subtasks and assigns them to admin agents. Next, *Dynamic Hierarchical Group Formation* occurs when an admin agent recruits additional agents if a subtask exceeds its capacity, forming a recursive, dynamic group to handle complex tasks. Finally, *System-Level Coordination and Communication* enables parallel execution and efficient interaction across the system, with agents connecting to external systems through function calls.

(2) **Hierarchical Monitoring:** To ensure agents complete tasks accurately without relying on predefined SOPs, *MegaAgent* incorporates hierarchical monitoring and coordination mechanisms. First, each agent is assigned a task by its admin agent upon generation. Then, *MegaAgent* employs a robust hierarchical monitoring and coordination framework for each agent as follows: *Agent-Level Monitoring* ensures each agent tracks its actions with a checklist, verifying progress before moving forward. At the *Group-Level Monitoring*, admin agents oversee the tasks of their assigned agents, ensuring smooth execution and coordination. At the *System-Level Monitoring*, the Boss Agent reviews the outputs of all groups to ensure accuracy and consistency. This hierarchical approach ensures both effective task management and completion across all levels.

Model	No Predefined SOP	Multi-file Support	Parallelism	Scalability
AutoGen	✗	✗	✗	✗
MetaGPT	✗	✓	✗	✗
CAMEL	✗	✗	✗	✗
AgentVerse	✓	✗	✗	✗
MegaAgent	✓	✓	✓	✓

Table 1: Novelty comparison of popular LLM-MA systems with MegaAgent. Details are explained in Table 9.

MegaAgent's framework is most beneficial for future applications requiring large-scale coordination of autonomous agents. In financial markets, it models systems where agents represent traders, investors, or regulatory bodies. These agents process market news, analyze trends, make trades, and respond to macroeconomic events, potentially scaling to millions or billions of agents. Initial studies highlight its applicability (Zhang et al., 2024a; Gao et al., 2024). In healthcare, MegaAgent can optimize systems with agents representing providers, patients, administrators, and policymakers, simulating complex workflows (Li et al., 2024a; Fan et al., 2025).

We conduct two experiments in widely recognized LLM-MA research scenarios (Hong et al., 2023; Guo et al., 2024) to demonstrate *MegaAgent*'s effectiveness and autonomy. **(1) Software development: Gobang Game Development.** This experiment highlights *MegaAgent*'s superior autonomy and efficiency compared to previous baselines, with *MegaAgent* being **the only model capable of completing the task within 800 seconds. (2) Social Simulation: National Policy Generation.** This task demonstrates *MegaAgent*'s large-scale autonomy and scalability, **generating and coordinating approximately 590 agents to produce the expected policies within 3000 seconds.** In contrast, baseline models can coordinate fewer than 10 agents and fail to generate the expected policies.

Our contributions are as follows:

❶ ***Autonomous Framework.*** We introduce *MegaAgent*, a practical framework enabling autonomous coordination in LLM-MA systems. It supports dynamic task decomposition, parallel execution, and systematic monitoring, ensuring efficient task management.

❷ ***Minimizing Human-designed Prompts.*** We notice the importance of minimizing human-designed prompts in LLM-MA systems, addressing a critical limitation of previous frameworks that creates a bottleneck for large-scale LLM-MA systems for complex tasks. To overcome this, we propose assigning LLM agents to autonomously split tasks and generate SOPs for agents. This approach reduces human intervention and enable broader range of users to employ LLM-MA systems effectively.

❸ ***Experimental Validation.*** Extensive experiments on two scenarios demonstrate that *MegaAgent* is: **(1) Superior**: It is the only framework capable of completing both Gobang game development and national policy simulation tasks, outperforming all baselines. **(2) Efficient**: *MegaAgent* successfully completes the Gobang game development task within 800 seconds, demonstrating its superior task execution and coordination capabilities. Moreover, it efficiently coordinates 590 agents for national policy generation within 3000 seconds, while baselines manage fewer than 10 agents and fail to complete the task. This remarkable agent count underscores *MegaAgent*'s scalability.

The remainder of this paper is organized as follows: Section 2 introduces *MegaAgent* framework in detail. Section 3 presents experimental evaluations demonstrating *MegaAgent*'s effectiveness. Section 4 reviews related work, and Section 5 concludes the paper.

2 MegaAgent Framework

2.1 Overview

We introduce the *MegaAgent* framework from two hierarchical perspectives, as outlined in Section 1: (1) Hierarchical Task Management and (2) Hierarchical Monitoring. An overview is provided in Figure 1. We elaborate the algorithm of MegaAgent in Algorithm 1.

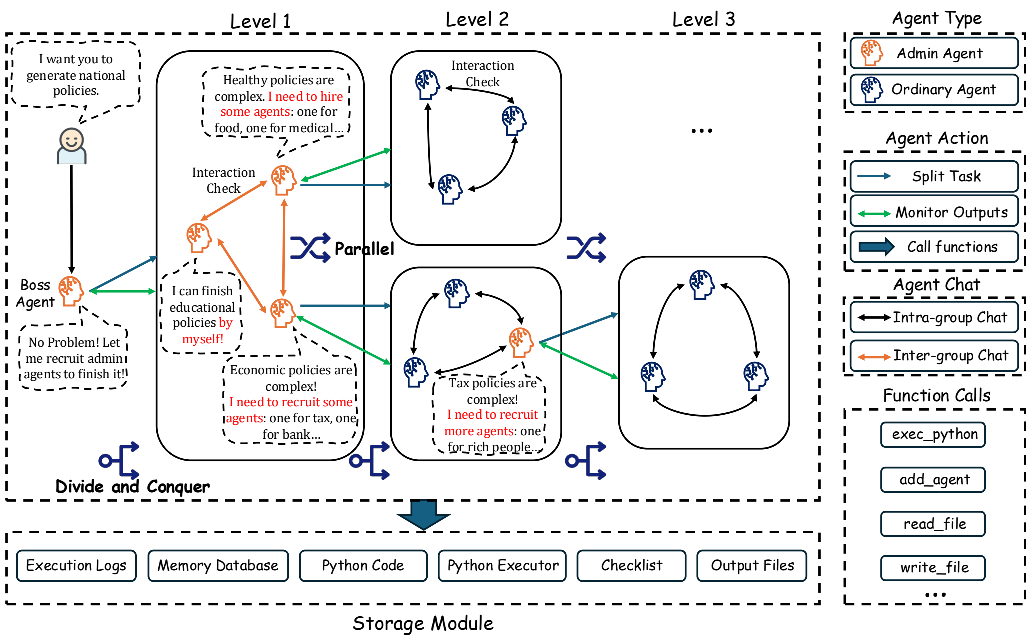

Level 1
Level 2
Level 3
I want you to generate national policies.
Boss Agent
No Problem! Let me recruit admin agents to finish it!
Healthy policies are complex. I need to hire some agents: one for food, one for medical...
Interaction Check
I can finish educational policies by myself!
Economic policies are complex! I need to recruit some agents: one for tax, one for bank...
Parallel
Interaction Check
Tax policies are complex! I need to recruit more agents: one for rich people...
...
Divide and Conquer
Execution Logs
Memory Database
Python Code
Python Executor
Checklist
Output Files
Storage Module
Agent Type
Admin Agent
Ordinary Agent
Agent Action
Split Task
Monitor Outputs
Call functions
Agent Chat
Intra-group Chat
Inter-group Chat
Function Calls
exec_python
add_agent
read_file
write_file
...

Figure 1: MegaAgent processes a user-provided meta-prompt by dividing it into distinct tasks, assigning each to a corresponding admin agent. Admin agents oversee their tasks, autonomously recruiting additional agents as needed to form task-specific groups that operate in parallel for efficient execution. These groups can further expand through sub-agent recruitment, creating a multi-level hierarchy. Admin agents supervise their groups to ensure task completion and output quality. Agents are classified into admin and ordinary agents: admin agents can communicate with one another, while ordinary agents interact only within their groups to optimize communication efficiency. Agents access and manage external files in storage module using function calls, supporting seamless data retrieval and task execution.

2.2 Hierachical Task Management

2.2.1 Multi-level Task Splitting

To efficiently manage complex tasks in large-scale LLM-MA systems, we implement a multi-level task management framework. *Boss Agent* is responsible for decomposing the main task into manageable subtasks upon receiving the meta-prompt from a user. Each subtask is delegated to a specialized admin agent with a well-defined role by *Boss Agent*. If a subtask is too complex for an admin agent to complete independently, it can recruit additional agents to handle specific components. These newly created agents can, in turn, recruit more agents if needed, assuming the role of admin agent themselves, as depicted in *Level 2* and *Level 3* in Figure 1. This recursive task-splitting mechanism enables the system to adapt dynamically as task complexity increases.

To enhance efficiency, we implement a parallel mechanism for agent groups operating at the same level. For instance, the two agent groups in *Level 2* of Figure 1 can work in parallel, with one generating economic policies and the other developing health policies. This parallelization reduces overall task completion time.

2.2.2 Hierarchical Coordination Mechanism

Effective task execution in *MegaAgent* is driven by a two-layer hierarchical coordination structure: (1) *Intra-group Chat*, where agents within the same task group collaborate by sharing updates through prompt-based communication, ensuring smooth progress and effective task execution when interaction is required, as indicated by the black double-arrow line in Figure 1; and (2) *Inter-group Chat*, where admin agents from different groups communicate to resolve task dependencies and coordinate cross-group efforts, as represented by the yellow double-arrow line in Figure 1. For instance, in the software development experiment discussed in subsection 3.1, the software implementation must adhere to the game logic designer's requirements. Ordinary agents are restricted from directly communicating with agents outside their group to enhance efficiency.

2.2.3 File Management

To enable effective interaction between LLM agents and external files, we introduce an external storage module that manages all file-related tasks. This module includes components such as agent execution logs, a memory database, task monitoring tools, Python code execution support, shared files, and individualized agent checklists. To ensure consistent and accurate file management, we propose the following two designs:

(1) **Git-Based Version Control.** To maintain file consistency, we integrate a Git-based version control mechanism. Since agents may spend considerable time editing files after reading them, concurrent modifications by other agents could cause conflicts. To prevent this, an agent retrieves the file's current Git commit hash upon reading it. Before making changes, the agent

submits this hash to the file management system, which commits the updates, merges them into the latest HEAD, and prompts the agent to resolve any merge conflicts if necessary. All Git operations are serialized using a global mutex lock to ensure synchronization and prevent race conditions.

(2) **Long-Term Memory Management with a Vector Database.** Many studies show that LLM agents would forget the conversation history after several rounds due to the token length limit (Becker, 2024; Xue et al., 2024). To address this, we implement a vector database to store the outputs of agents. Each output is encoded into embeddings using language models and stored in a vector database. Therefore, agents can retrieve relevant memory entries, enabling them to recall past interactions and maintain contextual awareness.

To visually illustrate the communication, we present an example of agent communication in Figure 2.

> *You are MinisterHealth, the Minister of Health. Your job is to develop a comprehensive policy document ('policy_health.txt') according to the guidelines provided in 'policy_health.txt'. You will collaborate with MinisterEducation (the Minister of Education), MinisterFinance (the Minister of Finance), and pass the final document to MinisterInfrastructure (the Minister of Infrastructure). You can recruit lots of citizens for testing health initiatives. Ensure adherence to the specified routine only. Your collaborators include MinisterEducation, MinisterFinance, and MinisterInfrastructure.*

Figure 2: Agent Communication Example: *National Leader* to *Minister of Health*

2.3 Hierarchical Monitoring

To ensure accurate task execution and minimize the propagation of hallucinations (Huang et al., 2023b; Hao et al., 2024) in an LLM-MA system, we implement a hierarchical monitoring mechanism that facilitates real-time oversight, error correction, and progress validation through a structured process.

2.3.1 Multi-level Monitoring

The monitoring system in *MegaAgent* follows a structured, multi-level hierarchy to ensure accurate task completion and prevent error propagation. Then, *MegaAgent* employs a hierarchical monitoring and coordination framework for each agent as follows:

- *Agent-Level Monitoring:* Each agent maintains an *checklist* upon its being generated by its admin agent to document its actions and verify progress. This monitoring ensures accountability and allows agents to independently validate their work before proceeding to the next step.

- *Group-Level Monitoring:* Each agent group is supervised by an admin agent, which tracks the progress of individual agents, ensures smooth execution, and coordinates tasks within the group.

- *System-Level Monitoring:* At the highest level, *Boss Agent* oversees the outputs of all agent groups upon task completion, ensuring adherence to the correct format and minimizing hallucinated results. This process enhances system-wide consistency, reliability, and correctness.

2.3.2 Failure Scenarios and Solutions

Monitoring focuses on two key aspects: output format verification and result validation, detailed as follows:

(1) **Output Format Verification.** First, the monitoring would focus on the output format of an agent. For example, if an agent generates a Python file that fails to execute, its admin agent would flag the issue, log the error, and prompt a retry. By enforcing consistent output formats, this step prevents downstream agents from misinterpreting data, reducing potential hallucinations.

(2) **Result Validation.** Once a group completes its tasks, the admin agent reviews the generated outputs and compares them against the initial task requirements. If discrepancies are detected, the admin agent would detail error messages, outline missing or incorrect aspects, prompt the responsible agents to revise their work. This validation process ensures that final outputs align with intended objectives while minimizing task failures.

To clarify the monitoring process, we outline common failure scenarios and solutions as follows:

- *Incomplete TODO Lists:* Agents may terminate prematurely or enter infinite loops due to inherent LLM limitations. An admin agent would detect it and prompt the agent to retry the task to ensure task completion.

- *Task Repetition:* Limited context memory may cause agents to forget completed tasks, leading to redundant actions or task loops. An admin agent would identify inconsistencies by cross-referencing agent checklists and prompts corrective actions as necessary.

- *Secure Alignment Interruptions:* Agents may become unresponsive or repeatedly return alignment-related constraint messages, such as *"Sorry, I can't help with that."* In such cases, an admin agent attempts to recruit other agents to finish the task.

To visually illustrate the monitoring, we present a monitoring log example in Figure 3.

By combining strict output format verification and result validation, this monitoring framework ensures agents remain aligned with system goals. Comprehensive error-handling processes prevent cascading failures, ensuring system stability and optimal performance throughout the LLM-MA framework.

3 Experiments

We evaluate *MegaAgent*'s capabilities through two experiments: software development and social simulation. These scenarios are chosen over tasks such as reasoning

> ***Financial Policy** - **Strengths**: Clear objectives, funding sources, and metrics for success. - **Improvements Needed**: - Ensure curriculum relevance for different demographics. - Incorporate interactive learning elements. - Make programs accessible to all, including those with disabilities. - Establish a feedback loop for continuous improvement. - **Action**: Talk to MinisterFinance for revisions based on feedback.*

Figure 3: Monitoring Log Example: *National Leader* monitors the financial policy.

or math problems, which a single LLM agent can handle (Chen et al., 2023b; Guo et al., 2024). The selected tasks demand extensive multi-agent coordination, providing a more realistic representation of coordination challenges in human societies.

We focus on the following three research questions:

RQ1: Can *MegaAgent* complete a task requiring extensive coordination without a predefined SOP? How do other baselines compare? (§subsection 3.1)

RQ2: Can *MegaAgent* be effectively scaled to handle more complex tasks that involve a significantly larger number of agents, showcasing its scalability? How does it compare to other baselines?(§subsection 3.2)

3.1 RQ1: Software Development - Gobang Game

Gobang is a strategic board game played between two participants who take turns placing black and white pieces on a grid. The objective is to be the first to align five consecutive pieces horizontally, vertically, or diagonally[2]. We select game development as a test scenario because it effectively evaluates an LLM-MA system's coding and coordination abilities. The task requires generating both backend logic and frontend components while involving extensive collaboration among roles like product manager, game logic designer, and software developers. This setting provides a robust evaluation of *MegaAgent*'s capabilities in coordination, autonomy, and parallelism in a project.

3.1.1 Experiment Setup

We conduct this experiment using the GPT-4o API[3], setting the 'temperature' parameter to 0 to ensure more deterministic responses (Achiam et al., 2023). The experiment begins by feeding the meta prompt to *MegaAgent* shown in Figure 4. More details are in Appendix E.

For comparative analysis, we employ AutoGen, MetaGPT, CAMEL, and AgentVerse to perform the same task. We manually adjust their backbones to GPT-4o or GPT-4 when GPT-4o is incompatible with their configurations. To ensure a fair evaluation, we design prompts tailored to each baseline's requirements while adhering to the guidelines specified in their respective papers to determine appropriate testing methods. Further details are in subsection E.6.

> *You are Bob, the leader of a software development club. Your club's current goal is to develop a Gobang game with an AI, and can be executed by running '`main.py`'.*

Figure 4: Gobang Game Meta Prompt

3.1.2 Evaluation Metrics

To evaluate the generated Gobang game, we establish the following evaluation metrics: **(1) Error-Free Execution**, which assesses the program's ability to run without errors; **(2) User Move**, which evaluates the user's ability to make a move; **(3) AI Move**, which measures the AI's ability to make a move; and **(4) Game Termination**, which ensures the game's ability to end correctly when there are five consecutive pieces.

3.1.3 Experiment Results

As shown in Table 2. *MegaAgent* autonomously generates an SOP involving seven agents, effectively coordinates their tasks, and successfully develops a fully functional Gobang game with an interactive interface within 800 seconds. **These achievements fulfill all task requirements, making *MegaAgent* the only system capable of producing a complete and operational game, unlike baseline models that either produce incomplete results or fail entirely.** Further details are provided in Appendix E. The performance of other baseline models is analyzed below:

AutoGen: AutoGen employs two agents but fails to produce a valid game move. After approximately three minutes, it generates a program ending with `# To be continued..` and becomes stuck when attempting execution. The likely cause of this failure is its overly simplistic SOP, lacking critical inter-agent communication steps such as code review. More details are provided in subsubsection E.6.1.

MetaGPT: Despite generating six agents, MetaGPT fails to produce a functional AI move in any trial. The main issues include: (1) unexecutable code due to the lack of debugging tools, (2) incorrect program generation, such as creating a *tic-tac-toe game*[4] instead of a Gobang game, likely due to a simplistic SOP and insufficient agent communication, and (3) infinite loops caused by incomplete implementations. More details are in subsubsection E.6.2.

CAMEL: CAMEL cannot produce executable Python code using two agents, likely due to weak planning and limited contextual reasoning capabilities. More details are in subsubsection E.6.3.

AgentVerse: AgentVerse generates four agents to complete the task but faces significant issues. In the first two trials, the agents repeatedly reject results for all ten

[2] `https://en.wikipedia.org/wiki/Gomoku`

[3] `https://openai.com/index/hello-gpt-4o/`

[4] `https://en.wikipedia.org/wiki/Tic-tac-toe`

Model	Error-Free Execution	User Move	AI Move	Game Termination	# of Agents	Time(s)	Time/Agent (s)
AutoGen	✓	✓	✗	✗	2	180	90
MetaGPT	✓	✓	✗	✗	6	480	80
CAMEL	✗	✗	✗	✗	2	1,830	915
AgentVerse	✗	✗	✗	✗	4	1,980	495
MegaAgent	✓	✓	✓	✓	7	800	114

Table 2: Gobang Game Development Results

rounds. In the third trial, while the result is accepted, the generated code contains numerous placeholders and remains unexecutable. The likely cause of failure is an overly rigid task outline during the planning stage, which current LLMs struggle to fulfill. More details are in subsubsection E.6.4.

3.1.4 Ablation Study

To validate the necessity of each component design in *MegaAgent*, we conduct an ablation study, with results in Table 3.

Components	Completed Metrics	# Agents	Time(s)	Time/Agent (s)
Full	(1) (2) (3) (4)	7	800	114
w/o hierarchy	(1) (2)	5	920	184
w/o parallelism	(1) (2) (3) (4)	7	4,505	643
w/o monitoring	(1) (2) (3)	7	300	42

Table 3: Gobang Ablation Study Results

Removing the hierarchical structure reduces agent usage to 5 but increases completion time to 920 seconds while achieving only basic metrics. Without parallelism, task groups complete their tasks sequentially, increasing time complexity from $O(\log n)$ to $O(n)$, which raises the execution time per agent from 114 seconds to 643 seconds. Removing monitoring reduces execution time to 300 seconds but fails to meet essential metrics. These findings underscore that parallel execution, hierarchy, and monitoring are all crucial for both task completion and execution speed. More details are in subsection E.4.

3.1.5 Cost Analysis

To evaluate token usage and better understand the efficiency of the Gobang game generation, we provide a detailed cost analysis. The analysis is divided into three stages: Planning, Task-Solving, and Merging, each representing distinct phases of the system's operation. The Planning stage focuses on initial strategy generation, the Task-Solving stage handles the core game-solving computations, and the Merging stage consolidates results for final outputs. We have two key insights from the results in Table 4 as follows:

Stage	# Input Tokens	# Output Tokens	# Total Tokens	Time (s)
Planning	42,947	12,347	55,294	0–60
Task-Solving	1,098,573	55,022	1,153,595	30–840
Merging	22,099	1,493	23,592	840–870
Total	1,163,619	68,862	1,232,481	870

Table 4: Token usage analysis across different stages of Gobang GPT-4o experiments.

Insight 1: High Resource Consumption in the Task-Solving Stage. The majority of the time and token usage occurs during the task-solving stage. This indicates that the task is inherently complex, requiring significant coordination among agents to generate solutions. This highlights the computational intensity of multi-agent interactions in solving strategic problems.

Insight 2: Disproportionate Input and Output Token Usage. The input token count is substantially higher than the output token count, revealing significant room for optimization in token usage. Notably, the input tokens predominantly originate from dialogues between agents. This suggests that improving the efficiency and structure of inter-agent communication could be a valuable research direction to enhance overall efficiency.

3.2 RQ2: Social Simulation - National Policy Generation

We propose a more challenging experiment: formulating national policies, which requires numerous agents to perform various tasks in complex domains such as education, health, and finance. We select this experiment because social simulations with LLM-MA systems require numerous agents—potentially scaling to hundreds—to mimic a human-like society. This experiment can evaluate *MegaAgent*'s autonomy, scalability, and coordination capabilities.

3.2.1 Experiment Setup

Due to budget constraints, we use the GPT-4o-mini API for this experiment conducted by *MegaAgent*. For comparative analysis, we utilize AutoGen, MetaGPT, CAMEL, and AgentVerse to perform the same task. We manually adapt their backbone LLMs to GPT-4o or GPT-4 when GPT-4o is incompatible with their code configurations. The meta prompt we feed into *MegaAgent* is shown in Figure 5, with more details provided in Appendix F. Descriptions of the other baseline settings are included in subsection F.5.

You are NationLeader, the leader of a pioneering nation. You want to develop the best detailed policy for your cutting-edge country in 'policy_department.txt'. You are now recruiting ministers and assigning work to them. For each possible minister, please write a prompt.

Figure 5: National Policy Generation Meta Prompt

3.2.2 Evaluation Metrics

To assess *MegaAgent*'s national policy reliability, we implement the *LLM-as-a-Judge* framework using five advanced models: Claude-3.5[5], GPT-4o-mini, GPT-4o, o1-mini, and o1-preview (Achiam et al., 2023). Our validation dataset contains 50 authentic national policies and 50 non-policy texts (multi-turn conversations, meeting summaries) (Zheng et al., 2023a) to test policy discrimination capability. Using the standardized prompt in Figure 6, models achieve 89% average identification accuracy in Table 10, confirming evaluation validity. Complete validation details are in Appendix G.

"Is this policy reasonable as a national policy? Please return your answer with clear nuances: Agree, Disagree, or Neutral with detailed explanations."

Figure 6: National Policy Evaluation Prompt

3.2.3 Experiment Results

We present National Policy Generation's experimental results in Table 5. It shows *MegaAgent*'s ability to generate complete and reasonable policies using a significantly larger number of agents within competitive time limits. The results show that ***MegaAgent* outperforms baseline models by producing complete policies with 590 agents in 2,991 seconds**. Notably, *MegaAgent*'s average processing time per agent is 5 seconds, significantly faster than the best-performing baseline at 40 seconds per agent, demonstrating its scalability. The structure of the policies generated by *MegaAgent* is illustrated in Figure 7, with detailed outputs provided in subsection F.3.

Model	Outputs	# Agents	Time (s)	Time/Agent (s)
AutoGen	Outline	1	40	40
MetaGPT	Python Program	6	580	97
CAMEL	Plans	2	1,380	690
AgentVerse	None	4	510	128
MegaAgent	Complete Policies	590	2,991	5

Table 5: National Policy Generation Results

To evaluate the reliability of *MegaAgent*'s generated national policies, we feed the prompt in Figure 6 to chosen advanced LLMs for reasonability assessment. As shown in Table 6, an average of 27.4 out of 31 policies are judged as reasonable by LLMs. This result highlights *MegaAgent*'s effectiveness in generating well-justified policies in this social simulation experiment.

3.2.4 Ablation Study

To validate the necessity of each component design in *MegaAgent*, we conduct an ablation study with results shown in Table 7.

[5]https://www.anthropic.com/claude/sonnet

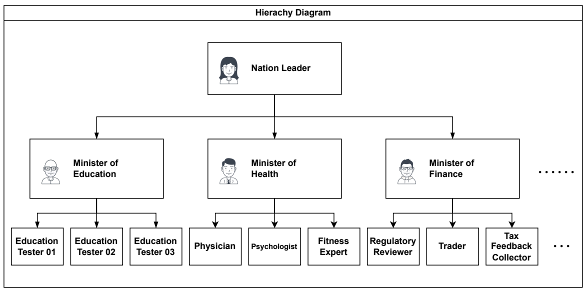


Figure 7: MegaAgent's Generated National Policy Structure

Model	# Agree	# Disagree	# Neutral
Claude-3.5	26	1	4
gpt-4o-mini	28	0	3
gpt-4o	25	2	4
o1-mini	29	2	0
o1-preview	29	1	1
Average	**27.4**	**1.2**	**2.4**

Table 6: Evaluating the Rationality of 31 Policies Generated by MegaAgent

Without hierarchy, only incomplete policies are produced within 450 seconds using 19 agents, indicating the importance of hierarchical design. Disabling parallelism entirely results in incomplete policies even after 14400 seconds, with over 100 agents continuously recruited but unable to complete tasks due to serialized processing bottlenecks. Removing monitoring generates policies with placeholders in 667 seconds using 50 agents, highlighting the need for continuous supervision for task completeness. Detailed outputs of these ablation studies are in subsection F.4.

These findings underscore that parallelism is not **merely beneficial but critical** for managing complex tasks in LLM-MA systems.

3.2.5 Cost Analysis

To assess the token and time costs of this experiment, we perform a detailed analysis of token usage and execution time across three stages: Planning, Task-Solving, and Merging. The results are presented in Table 8.

Similar to analysis in 3.1.5, we observe from Table 8 that significant resource consumption during the task-solving stage, which dominates both time and token usage. A comparison of input-to-output token ratios between the experiments reveals consistent inefficiencies, with the first experiment showing a ratio of approximately 23:1, while the current experiment is slightly higher at 25:1. This increase suggests that the **policy generation task required additional resources for inter-agent dialogues and greater context management, likely due to the involvement of a larger number of agents.** These findings highlight the critical need to optimize token usage and enhance dialogue efficiency,

Components	Outputs	# of Agents	Time (s)	Time/Agent (s)
Full	Complete Policies	590	2,991	5
w/o hierarchy	Incomplete Policies	19	450	24
w/o parallelism	Incomplete Policies	>100	>14,400	N.A.
w/o monitoring	Policies with Placeholders	50	667	13

*We terminate the execution without parallelism after 14400 seconds.

Table 7: National Policy Generation Ablation Study Results

Stage	Input Tokens	Output Tokens	Total Tokens	Time (s)
Planning	111,601	24,103	135,704	0–180
Task-Solving	8,003,124	343,670	8,346,794	20–2,950
Merging	348,264	13,280	361,544	2,400–3,000
Total	8,463,989	381,053	8,845,042	3,000

Table 8: Token usage analysis for National Policy Generation.

which could significantly reduce resource consumption and improve overall performance in LLM-MA systems.

3.3 Scalability Analysis

In *MegaAgent*, for n agents, the hierarchical layer-to-layer communication cost is $O(\log n)$, as agent groups at the same level operate in parallel, as illustrated in Figure 7. In contrast, existing frameworks exhibit linear running time growth $O(n)$ as they run serially, which becomes impractical with the number of LLM agents increasing much. The analysis is supported by our national policy generation experiment in subsection 3.2, where *MegaAgent*'s average processing time per agent is 5 seconds, compared to CAMEL's average of 700 seconds per agent. These results highlight *MegaAgent*'s scalability and practicality for autonomous coordination in large-scale LLM-MA systems.

4 Related Work

We discuss the most related work here and leave more details in Appendix D.

4.1 LLM-MA Systems

With the emergence of powerful LLMs (Achiam et al., 2023; Team et al., 2023), recent research on LLM-based multi-agent systems has investigated how multiple agents can accomplish tasks through coordination, utilizing elements such as personas (Chen et al., 2024b; Chan et al., 2024), planning (Chen et al., 2023a; Zhang et al., 2024c; Yuan et al., 2023), and memory (Zhang et al., 2023a; Hatalis et al., 2023). Unlike systems relying on a single LLM-based agent, multi-agent systems demonstrate superiority in tackling challenging tasks. Recent works, such as MetaGPT (Hong et al., 2023), AutoGen (Wu et al., 2023b), and AgentVerse (Chen et al., 2023b), design specific roles to achieve a task.

However, most popular LLM-MA systems heavily rely on handcrafted prompts and expert design. For instance, MetaGPT (Hong et al., 2023) requires users to pre-design roles like product manager and software engineer. Another limitation is these systems utilize a sequential pipeline without considering parallel execution of agents (Li et al., 2023a). Although AgentScope (Pan et al., 2024) does consider this, its implementation follows a fixed trajectory in different rounds of interaction, prohibiting changes in communication partners, thus limiting performance improvement as the number of agents scales up.

In contrast, in the real world, when many software developers are employed, they may first work on different files simultaneously, and then focus on one specific file when difficulties are encountered, sparking creative ideas to overcome challenges by coordination. Additionally, existing LLM-MA systems are restricted by their small scale and have not been applied in large-scale scenarios with complex coordination involved. We compare current popular LLM-MA systems with *MegaAgent* in Table 9. We can see from the table that *MegaAgent* stands out for its high autonomy, multi-file support, parallelism, and scalability.

4.2 SOPs in LLM-MA Systems

Allocating SOPs is a common approach in designing agent profiles and tasks within LLM-based multi-agent (LLM-MA) systems (Hong et al., 2023; Huang et al., 2023a; Park et al., 2023; Zhuge et al., 2024; Shi et al., 2024). These systems define SOPs for both individual agents and their communication protocols. While this method has proven effective in previous works, it has two major limitations: (1) Agents may possess unforeseen capabilities that cannot be anticipated during the human design stage but become relevant during task execution (Rivera et al., 2024; Sypherd and Belle, 2024; Piatti et al., 2024); (2) As the scale of LLM-MA systems grows—potentially involving thousands or even billions of agents—designing SOPs manually for each agent becomes infeasible (Mou et al., 2024; Pan et al., 2024). To address this, the design mechanism must evolve, leveraging LLMs themselves, as in the *LLM-as-the-Judge* concept (Huang et al., 2024; Chen et al., 2024a), allowing LLMs to autonomously generate SOPs for large-scale LLM-MA systems.

5 Conclusion

We present *MegaAgent*, a practical LLM-MA framework enabling dynamic and autonomous coordination, where users only need to provide a meta prompt at the start of the process. Through a Gobang game software development experiment, we demonstrate *MegaAgent*'s superior autonomy and coordination compared to baseline models. Our social simulation on national policy generation highlights *MegaAgent*'s scalability to hundreds of agents while ensuring effective coordination. With its hierarchical and adaptive design, *MegaAgent* has the potential to serve as the foundational OS for future LLM-MA systems. We encourage the research community to further explore enhancing agent coordination to address the increasing demands of large-scale LLM-MA systems.

Limitations

Planning and Communication Overhead. The primary bottleneck lies in the planning and communication processes among LLM agents, particularly in translating code into prompts, managing task checklists, maintaining the framework, and debugging. As the number of agents and communication rounds increases, input-output token consumption grows substantially, affecting both efficiency and cost. Future work should explore advanced token summarization, semantic compression, and efficient dialogue storage methods.

Hallucination in Agent Outputs. Despite using task-specific checklists to monitor agent actions, occasional hallucinations persist, with output formats sometimes deviating from expected requirements. Since the checklists themselves are generated by LLMs, errors may propagate. Addressing this requires more robust verification mechanisms, potentially involving external expert knowledge bases before, during, or after agent response generation.

API Cost and Model Integration. *MegaAgent*'s reliance on GPT-4 incurs high API costs. While cheaper alternatives exist, they may lack generalizability. A promising direction would involve integrating specialized LLMs for specific tasks, leveraging models that excel in certain domains while maintaining efficient communication and data sharing across the LLMs.

References

Tensorrt-llm: A tensorrt toolbox for optimized large language model inference.

2023. Huggingface text generation inference. https://github.com/huggingface/text-generation-inference. Commit: 3c02262, Accessed on: 2023-11-25.

2023. Promptflow.

2024. Langchain.

Ibrahim Abdelaziz, Kinjal Basu, Mayank Agarwal, Sadhana Kumaravel, Matt Stallone, Rameswar Panda, Yara Rizk, G. Bhargav, M. Crouse, Chulaka Gunasekara, S. Ikbal, Sachin Joshi, Hima P. Karanam, Vineet Kumar, Asim Munawar, S. Neelam, Dinesh Raghu, Udit Sharma, Adriana Meza Soria, Dheeraj Sreedhar, P. Venkateswaran, Merve Unuvar, David Cox, S. Roukos, Luis A. Lastras, and P. Kapanipathi. 2024. Granite-function calling model: Introducing function calling abilities via multi-task learning of granular tasks. In *Conference on Empirical Methods in Natural Language Processing*.

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*.

Ahsan Ali, Riccardo Pinciroli, Feng Yan, and Evgenia Smirni. 2020. Batch: machine learning inference serving on serverless platforms with adaptive batching. In *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–15. IEEE.

Maksym Andriushchenko, Alexandra Souly, Mateusz Dziemian, Derek Duenas, Maxwell Lin, Justin Wang, Dan Hendrycks, Andy Zou, Zico Kolter, Matt Fredrikson, et al. 2024. Agentharm: A benchmark for measuring harmfulness of llm agents. *arXiv preprint arXiv:2410.09024*.

Apache. 2019. Tez. https://tez.apache.org/.

Sourav Banerjee, Ayushi Agarwal, and Saloni Singla. 2024. Llms will always hallucinate, and we need to live with this. *arXiv preprint arXiv:2409.05746*.

Jonas Becker. 2024. Multi-agent large language models for conversational task-solving. *arXiv preprint arXiv:2410.22932*.

Maciej Besta, Nils Blach, Ales Kubicek, Robert Gerstenberger, Michal Podstawski, Lukas Gianinazzi, Joanna Gajda, Tomasz Lehmann, Hubert Niewiadomski, Piotr Nyczyk, et al. 2024. Graph of thoughts: Solving elaborate problems with large language models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 17682–17690.

Bradley Brown, Jordan Juravsky, Ryan Ehrlich, Ronald Clark, Quoc V Le, Christopher Ré, and Azalia Mirhoseini. 2024. Large language monkeys: Scaling inference compute with repeated sampling. *arXiv preprint arXiv:2407.21787*.

Souradip Chakraborty, Soumya Suvra Ghosal, Ming Yin, Dinesh Manocha, Mengdi Wang, Amrit Bedi, and Furong Huang. 2024. Transfer q-star : Principled decoding for LLM alignment. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*.

Xin Chan, Xiaoyang Wang, Dian Yu, Haitao Mi, and Dong Yu. 2024. Scaling synthetic data creation with 1,000,000,000 personas. *arXiv preprint arXiv:2406.20094*.

Dongping Chen, Ruoxi Chen, Shilin Zhang, Yinuo Liu, Yaochen Wang, Huichi Zhou, Qihui Zhang, Yao Wan, Pan Zhou, and Lichao Sun. 2024a. Mllm-as-a-judge: Assessing multimodal llm-as-a-judge with vision-language benchmark. *arXiv preprint arXiv:2402.04788*.

Jiangjie Chen, Xintao Wang, Rui Xu, Siyu Yuan, Yikai Zhang, Wei Shi, Jian Xie, Shuang Li, Ruihan Yang, Tinghui Zhu, et al. 2024b. From persona to personalization: A survey on role-playing language agents. *arXiv preprint arXiv:2404.18231*.

Jiangjie Chen, Siyu Yuan, Rong Ye, Bodhisattwa Prasad Majumder, and Kyle Richardson. 2023a. Put your money where your mouth is: Evaluating strategic planning and execution of llm agents in an auction arena. *arXiv preprint arXiv:2310.05746*.

Weize Chen, Yusheng Su, Jingwei Zuo, Cheng Yang, Chenfei Yuan, Chen Qian, Chi-Min Chan, Yujia Qin, Yaxi Lu, Ruobing Xie, et al. 2023b. Agentverse: Facilitating multi-agent collaboration and exploring emergent behaviors in agents. *arXiv preprint arXiv:2308.10848*.

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

Zhihao Fan, Lai Wei, Jialong Tang, Wei Chen, Wang Siyuan, Zhongyu Wei, and Fei Huang. 2025. Ai hospital: Benchmarking large language models in a multi-agent medical interaction simulator. In *Proceedings of the 31st International Conference on Computational Linguistics*, pages 10183–10213.

Yichao Fu, Peter Bailis, Ion Stoica, and Hao Zhang. Break the sequential dependency of llm inference using lookahead decoding. In *Forty-first International Conference on Machine Learning*.

Chen Gao, Xiaochong Lan, Zhihong Lu, Jinzhu Mao, Jinghua Piao, Huandong Wang, Depeng Jin, and Yong Li. 2023a. S³: Social-network simulation system with large language model-empowered agents. *arXiv preprint arXiv:2307.14984*.

Luyu Gao, Aman Madaan, Shuyan Zhou, Uri Alon, Pengfei Liu, Yiming Yang, Jamie Callan, and Graham Neubig. 2023b. PAL: Program-aided language models. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pages 10764–10799. PMLR.

Shen Gao, Yuntao Wen, Minghang Zhu, Jianing Wei, Yuhan Cheng, Qunzi Zhang, and Shuo Shang. 2024. Simulating financial market via large language model based agents. *arXiv preprint arXiv:2406.19966*.

Arpan Gujarati, Reza Karimi, Safya Alzayat, Wei Hao, Antoine Kaufmann, Ymir Vigfusson, and Jonathan Mace. 2020. Serving dnns like clockwork: Performance predictability from the bottom up. In *Proc. USENIX OSDI*.

Taicheng Guo, Xiuying Chen, Yaqi Wang, Ruidi Chang, Shichao Pei, Nitesh V Chawla, Olaf Wiest, and Xiangliang Zhang. 2024. Large language model based multi-agents: A survey of progress and challenges. *arXiv preprint arXiv:2402.01680*.

Guozhi Hao, Jun Wu, Qianqian Pan, and Rosario Morello. 2024. Quantifying the uncertainty of llm hallucination spreading in complex adaptive social networks. *Scientific reports*, 14(1):16375.

Kostas Hatalis, Despina Christou, Joshua Myers, Steven Jones, Keith Lambert, Adam Amos-Binks, Zohreh Dannenhauer, and Dustin Dannenhauer. 2023. Memory matters: The need to improve long-term memory in llm-agents. In *Proceedings of the AAAI Symposium Series*, volume 2, pages 277–280.

Sirui Hong, Xiawu Zheng, Jonathan Chen, Yuheng Cheng, Jinlin Wang, Ceyao Zhang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, Liyang Zhou, et al. 2023. Metagpt: Meta programming for multi-agent collaborative framework. *arXiv preprint arXiv:2308.00352*.

Sirui Hong, Mingchen Zhuge, Jonathan Chen, Xiawu Zheng, Yuheng Cheng, Jinlin Wang, Ceyao Zhang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, Liyang Zhou, Chenyu Ran, Lingfeng Xiao, Chenglin Wu, and Jürgen Schmidhuber. 2024. MetaGPT: Meta programming for a multi-agent collaborative framework. In *The Twelfth International Conference on Learning Representations*.

Wenyue Hua, Lizhou Fan, Lingyao Li, Kai Mei, Jianchao Ji, Yingqiang Ge, Libby Hemphill, and Yongfeng Zhang. 2023. War and peace (waragent): Large language model-based multi-agent simulation of world wars. *arXiv preprint arXiv:2311.17227*.

Dong Huang, Qingwen Bu, Jie M Zhang, Michael Luck, and Heming Cui. 2023a. Agentcoder: Multi-agent-based code generation with iterative testing and optimisation. *arXiv preprint arXiv:2312.13010*.

Hui Huang, Yingqi Qu, Jing Liu, Muyun Yang, and Tiejun Zhao. 2024. An empirical study of llm-as-a-judge for llm evaluation: Fine-tuned judge models are task-specific classifiers. *arXiv preprint arXiv:2403.02839*.

Lei Huang, Weijiang Yu, Weitao Ma, Weihong Zhong, Zhangyin Feng, Haotian Wang, Qianglong Chen, Weihua Peng, Xiaocheng Feng, Bing Qin, et al. 2023b. A survey on hallucination in large language models: Principles, taxonomy, challenges, and open questions. *ACM Transactions on Information Systems*.

Michael Isard, Mihai Budiu, Yuan Yu, Andrew Birrell, and Dennis Fetterly. 2007. Dryad: Distributed data-parallel programs from sequential building blocks. In *Proceedings of the 2nd ACM SIGOPS/EuroSys European Conference on Computer Systems 2007*, EuroSys '07, page 59–72, New York, NY, USA. Association for Computing Machinery.

Minbyul Jeong, Jiwoong Sohn, Mujeen Sung, and Jaewoo Kang. 2024. Improving medical reasoning through retrieval and self-reflection with retrieval-augmented large language models. *Bioinformatics*, 40(Supplement_1):i119–i129.

Ziwei Ji, Tiezheng Yu, Yan Xu, Nayeon Lee, Etsuko Ishii, and Pascale Fung. 2023. Towards mitigating llm hallucination via self reflection. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 1827–1843.

Jinhao Jiang, Kun Zhou, Zican Dong, Keming Ye, Wayne Xin Zhao, and Ji-Rong Wen. 2023. Structgpt:

A general framework for large language model to reason over structured data. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 9237–9251.

Mingyu Jin, Beichen Wang, Zhaoqian Xue, Suiyuan Zhu, Wenyue Hua, Hua Tang, Kai Mei, Mengnan Du, and Yongfeng Zhang. 2024. What if llms have different world views: Simulating alien civilizations with llm-based agents. *arXiv preprint arXiv:2402.13184*.

Yunho Jin, Chun-Feng Wu, David Brooks, and Gu-Yeon Wei. 2023. S3: Increasing gpu utilization during generative inference for higher throughput. *Advances in Neural Information Processing Systems*, 36:18015–18027.

Tianjie Ju, Yiting Wang, Xinbei Ma, Pengzhou Cheng, Haodong Zhao, Yulong Wang, Lifeng Liu, Jian Xie, Zhuosheng Zhang, and Gongshen Liu. 2024. Flooding spread of manipulated knowledge in llm-based multi-agent communities. *arXiv preprint arXiv:2407.07791*.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. 2023a. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th Symposium on Operating Systems Principles*, pages 611–626.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. 2023b. Efficient memory management for large language model serving with pagedattention. In *Proc. ACM SOSP*.

Rémi Leblond, Jean-Baptiste Alayrac, L. Sifre, Miruna Pislar, Jean-Baptiste Lespiau, Ioannis Antonoglou, K. Simonyan, and O. Vinyals. 2021. Machine translation decoding beyond beam search. In *Conference on Empirical Methods in Natural Language Processing*.

Donghyun Lee and Mo Tiwari. 2024. Prompt infection: Llm-to-llm prompt injection within multi-agent systems. *arXiv preprint arXiv:2410.07283*.

Guohao Li, Hasan Hammoud, Hani Itani, Dmitrii Khizbullin, and Bernard Ghanem. 2023a. Camel: Communicative agents for" mind" exploration of large language model society. *Advances in Neural Information Processing Systems*, 36:51991–52008.

Junkai Li, Yunghwei Lai, Weitao Li, Jingyi Ren, Meng Zhang, Xinhui Kang, Siyu Wang, Peng Li, Ya-Qin Zhang, Weizhi Ma, et al. 2024a. Agent hospital: A simulacrum of hospital with evolvable medical agents. *arXiv preprint arXiv:2405.02957*.

Xiang Li, Zhenyu Li, Chen Shi, Yong Xu, Qing Du, Mingkui Tan, and Jun Huang. 2024b. AlphaFin: Benchmarking financial analysis with retrieval-augmented stock-chain framework. In *Proceedings of the 2024 Joint International Conference on Computational Linguistics, Language Resources and Evaluation (LREC-COLING 2024)*, pages 773–783. ELRA and ICCL.

Zhuohan Li, Lianmin Zheng, Yinmin Zhong, Vincent Liu, Ying Sheng, Xin Jin, Yanping Huang, Zhifeng Chen, Hao Zhang, Joseph E Gonzalez, and et al. 2023b. Alpaserve: Statistical multiplexing with model parallelism for deep learning serving. In *Proc. USENIX OSDI*.

Tian Liang, Zhiwei He, Wenxiang Jiao, Xing Wang, Yan Wang, Rui Wang, Yujiu Yang, Shuming Shi, and Zhaopeng Tu. 2023. Encouraging divergent thinking in large language models through multi-agent debate. *arXiv preprint arXiv:2305.19118*.

Chaofan Lin, Zhenhua Han, Chengruidong Zhang, Yuqing Yang, Fan Yang, Chen Chen, and Lili Qiu. 2024. Parrot: Efficient serving of llm-based applications with semantic variable. In *Proc. USENIX OSDI*.

Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, et al. 2023a. Agentbench: Evaluating llms as agents. *arXiv preprint arXiv:2308.03688*.

Zijun Liu, Yanzhe Zhang, Peng Li, Yang Liu, and Diyi Yang. 2023b. Dynamic llm-agent network: An llm-agent collaboration framework with agent team optimization. *arXiv preprint arXiv:2310.02170*.

LINHAO LUO, Yuan-Fang Li, Reza Haf, and Shirui Pan. Reasoning on graphs: Faithful and interpretable large language model reasoning. In *The Twelfth International Conference on Learning Representations*.

Ashraf Mahgoub, Karthick Shankar, Subrata Mitra, Ana Klimovic, Somali Chaterji, and Saurabh Bagchi. 2021. SONIC: Application-aware data passing for chained serverless applications. In *2021 USENIX Annual Technical Conference (USENIX ATC 21)*, pages 285–301. USENIX Association.

Alex Mallen, Akari Asai, Victor Zhong, Rajarshi Das, Daniel Khashabi, and Hannaneh Hajishirzi. 2023. When not to trust language models: Investigating effectiveness of parametric and non-parametric memories. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 9802–9822, Toronto, Canada. Association for Computational Linguistics.

Kai Mei, Zelong Li, Shuyuan Xu, Ruosong Ye, Yingqiang Ge, and Yongfeng Zhang. 2024a. Aios: Llm agent operating system. *arXiv.org*.

Kai Mei, Zelong Li, Shuyuan Xu, Ruosong Ye, Yingqiang Ge, and Yongfeng Zhang. 2024b. Llm agent operating system. *arXiv preprint arXiv:2403.16971*.

Xinyi Mou, Xuanwen Ding, Qi He, Liang Wang, Jingcong Liang, Xinnong Zhang, Libo Sun, Jiayu Lin, Jie Zhou, Xuanjing Huang, et al. 2024. From individual to society: A survey on social simulation driven by large language model-based agents. *arXiv preprint arXiv:2412.03563*.

Christopher Olston, Noah Fiedel, Kiril Gorovoy, Jeremiah Harmsen, Li Lao, Fangwei Li, Vinu Rajashekhar, Sukriti Ramesh, and Jordan Soyke. 2017. Tensorflow-serving: Flexible, high-performance ml serving. *arXiv preprint arXiv:1712.06139*.

Xuchen Pan, Dawei Gao, Yuexiang Xie, Zhewei Wei, Yaliang Li, Bolin Ding, Ji-Rong Wen, and Jingren Zhou. 2024. Very large-scale multi-agent simulation in agentscope. *arXiv preprint arXiv:2407.17789*.

Joon Sung Park, Joseph O'Brien, Carrie Jun Cai, Meredith Ringel Morris, Percy Liang, and Michael S Bernstein. 2023. Generative agents: Interactive simulacra of human behavior. In *Proceedings of the 36th annual acm symposium on user interface software and technology*, pages 1–22.

Giorgio Piatti, Zhijing Jin, Max Kleiman-Weiner, Bernhard Schölkopf, Mrinmaya Sachan, and Rada Mihalcea. 2024. Cooperate or collapse: Emergence of sustainable cooperation in a society of llm agents. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*.

Nicholas Pipitone and Ghita Houir Alami. 2024. Legalbench-rag: A benchmark for retrieval-augmented generation in the legal domain. *arXiv preprint arXiv:2408.10343*.

Reiner Pope, Sholto Douglas, Aakanksha Chowdhery, Jacob Devlin, James Bradbury, Jonathan Heek, Kefan Xiao, Shivani Agrawal, and Jeff Dean. 2023. Efficiently scaling transformer inference. *Proceedings of Machine Learning and Systems*, 5:606–624.

Reid Pryzant, Dan Iter, Jerry Li, Yin Tat Lee, Chenguang Zhu, and Michael Zeng. 2023. Automatic prompt optimization with" gradient descent" and beam search. *arXiv preprint arXiv:2305.03495*.

Yujia Qin, Shi Liang, Yining Ye, Kunlun Zhu, Lan Yan, Ya-Ting Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, Sihan Zhao, Runchu Tian, Ruobing Xie, Jie Zhou, M. Gerstein, Dahai Li, Zhiyuan Liu, and Maosong Sun. 2023. Toolllm: Facilitating large language models to master 16000+ real-world apis. *International Conference on Learning Representations*.

Corban Rivera, Grayson Byrd, William Paul, Tyler Feldman, Meghan Booker, Emma Holmes, David Handelman, Bethany Kemp, Andrew Badger, Aurora Schmidt, et al. 2024. Conceptagent: Llm-driven precondition grounding and tree search for robust task planning and execution. *arXiv preprint arXiv:2410.06108*.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessi, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. 2023. Toolformer: Language models can teach themselves to use tools. In *Advances in Neural Information Processing Systems*, volume 36, pages 68539–68551. Curran Associates, Inc.

Zhengliang Shi, Shen Gao, Xiuyi Chen, Yue Feng, Lingyong Yan, Haibo Shi, Dawei Yin, Pengjie Ren, Suzan Verberne, and Zhaochun Ren. 2024. Learning to use tools via cooperative and interactive agents. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pages 10642–10657, Miami, Florida, USA. Association for Computational Linguistics.

Jiashuo Sun, Chengjin Xu, Lumingyuan Tang, Saizhuo Wang, Chen Lin, Yeyun Gong, Lionel Ni, Heung-Yeung Shum, and Jian Guo. Think-on-graph: Deep and responsible reasoning of large language model on knowledge graph. In *The Twelfth International Conference on Learning Representations*.

Chris Sypherd and Vaishak Belle. 2024. Practical considerations for agentic llm systems. *arXiv preprint arXiv:2412.04093*.

Gemini Team, Rohan Anil, Sebastian Borgeaud, Yonghui Wu, Jean-Baptiste Alayrac, Jiahui Yu, Radu Soricut, Johan Schalkwyk, Andrew M Dai, Anja Hauth, et al. 2023. Gemini: a family of highly capable multimodal models. *arXiv preprint arXiv:2312.11805*.

SM Tonmoy, SM Zaman, Vinija Jain, Anku Rani, Vipula Rawte, Aman Chadha, and Amitava Das. 2024. A comprehensive survey of hallucination mitigation techniques in large language models. *arXiv preprint arXiv:2401.01313*.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. 2023. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*.

Tu Vu, Mohit Iyyer, Xuezhi Wang, Noah Constant, Jerry Wei, Jason Wei, Chris Tar, Yun-Hsuan Sung, Denny Zhou, Quoc Le, and Thang Luong. 2023. Freshllms: Refreshing large language models with search engine augmentation. In *Annual Meeting of the Association for Computational Linguistics*.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. a. Voyager: An open-ended embodied agent with large language models. *Transactions on Machine Learning Research*.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc V Le, Ed H Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. b. Self-consistency improves chain of thought reasoning in language models. In *The Eleventh International Conference on Learning Representations*.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837.

Bingyang Wu, Yinmin Zhong, Zili Zhang, Gang Huang, Xuanzhe Liu, and Xin Jin. 2023a. Fast distributed inference serving for large language models. In *arXiv preprint arXiv:2305.05920*.

Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, et al. Autogen: Enabling next-gen llm applications via multi-agent conversation. In *ICLR 2024 Workshop on Large Language Model (LLM) Agents*.

Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Shaokun Zhang, Erkang Zhu, Beibin Li, Li Jiang, Xiaoyun Zhang, and Chi Wang. 2023b. Autogen: Enabling next-gen llm applications via multi-agent conversation framework. *arXiv preprint arXiv:2308.08155*.

Fuzhao Xue, Yao Fu, Wangchunshu Zhou, Zangwei Zheng, and Yang You. 2024. To repeat or not to repeat: Insights from scaling llm under token-crisis. *Advances in Neural Information Processing Systems*, 36.

Rui Yang, Lin Song, Yanwei Li, Sijie Zhao, Yixiao Ge, Xiu Li, and Ying Shan. 2023. Gpt4tools: Teaching large language model to use tools via self-instruction. In *Advances in Neural Information Processing Systems*, volume 36, pages 71995–72007. Curran Associates, Inc.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. 2024. Tree of thoughts: Deliberate problem solving with large language models. *Advances in Neural Information Processing Systems*, 36.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Izhak Shafran, Karthik R Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. In *NeurIPS 2022 Foundation Models for Decision Making Workshop*.

Gyeong-In Yu, Joo Seong Jeong, Geon-Woo Kim, Soojeong Kim, and Byung-Gon Chun. 2022. Orca: A distributed serving system for Transformer-Based generative models. In *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*, pages 521–538, Carlsbad, CA. USENIX Association.

Siyu Yuan, Jiangjie Chen, Ziquan Fu, Xuyang Ge, Soham Shah, Charles Robert Jankowski, Yanghua Xiao, and Deqing Yang. 2023. Distilling script knowledge from large language models for constrained language planning. *arXiv preprint arXiv:2305.05252*.

Chong Zhang, Xinyi Liu, Mingyu Jin, Zhongmou Zhang, Lingyao Li, Zhengting Wang, Wenyue Hua, Dong Shu, Suiyuan Zhu, Xiaobo Jin, et al. 2024a. When ai meets finance (stockagent): Large language model-based stock trading in simulated real-world environments. *arXiv preprint arXiv:2407.18957*.

Kai Zhang, Fubang Zhao, Yangyang Kang, and Xiaozhong Liu. 2023a. Memory-augmented llm personalization with short-and long-term memory coordination. *arXiv preprint arXiv:2309.11696*.

Shizhuo Zhang, Curt Tigges, Stella Biderman, M. Raginsky, and T. Ringer. 2023b. Can transformers learn to solve problems recursively? *arXiv.org*.

Yang Zhang, Shixin Yang, Chenjia Bai, Fei Wu, Xiu Li, Xuelong Li, and Zhen Wang. 2024b. Towards efficient llm grounding for embodied multi-agent collaboration. *arXiv preprint arXiv:2405.14314*.

Yikai Zhang, Siyu Yuan, Caiyu Hu, Kyle Richardson, Yanghua Xiao, and Jiangjie Chen. 2024c. Timearena: Shaping efficient multitasking language agents in a time-aware simulation. *arXiv preprint arXiv:2402.05733*.

Wayne Xin Zhao, Kun Zhou, Junyi Li, Tianyi Tang, Xiaolei Wang, Yupeng Hou, Yingqian Min, Beichen Zhang, Junjie Zhang, Zican Dong, et al. 2023. A survey of large language models. *arXiv preprint arXiv:2303.18223*.

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

Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Jeff Huang, Chuyue Sun, Cody Hao Yu, Shiyi Cao, Christos Kozyrakis, Ion Stoica, Joseph E Gonzalez, and et al. 2023b. Efficiently programming large language models using sglang. *arXiv*.

Zangwei Zheng, Xiaozhe Ren, Fuzhao Xue, Yang Luo, Xin Jiang, and Yang You. 2024. Response length perception and sequence scheduling: An llm-empowered llm inference pipeline. *Advances in Neural Information Processing Systems*, 36.

Ming Zhong, Da Yin, Tao Yu, Ahmad Zaidi, Mutethia Mutuma, Rahul Jha, Ahmed Hassan Awadallah, Asli Celikyilmaz, Yang Liu, Xipeng Qiu, et al. 2021. Qmsum: A new benchmark for query-based multi-domain meeting summarization. *arXiv preprint arXiv:2104.05938*.

Andy Zhou, Kai Yan, Michal Shlapentokh-Rothman, Haohan Wang, and Yu-Xiong Wang. Language agent tree search unifies reasoning, acting, and planning in language models. In *Forty-first International Conference on Machine Learning*.

Deyao Zhu, Jun Chen, Xiaoqian Shen, Xiang Li, and Mohamed Elhoseiny. 2023. Minigpt-4: Enhancing vision-language understanding with advanced large language models. *arXiv preprint arXiv:2304.10592*.

Mingchen Zhuge, Wenyi Wang, Louis Kirsch, Francesco Faccio, Dmitrii Khizbullin, and Jürgen Schmidhuber. 2024. Language agents as optimizable graphs. *arXiv preprint arXiv:2402.16823*.

Appendix

A Algorithm

Algorithm 1 MegaAgent Algorithm from Prompt to Task Completion

```
Require: Prompt $\mathcal{P}$
Ensure: Completed Task Output $\mathcal{O}$
 1: Initialization: Assign BossAgent as root agent.
 2: BossAgent Decompose $\mathcal{P}$ into subtasks $S = \{s_1, s_2, \ldots, s_n\}$.
 3: BossAgent assign each subtask $s_i \in S$ to an Admin Agent.
 4: repeat
 5:     for all Admin Agent in parallel do
 6:         Task Splitting:
 7:         if $s_i$ is complex then
 8:             Recruit sub-agents to handle components of $s_i$.
 9:         end if
10:         Intra-group Coordination:
11:         Agents communicate via prompts within their group.
12:         Execute Task: Each agent completes assigned components.
13:         Monitoring: Validate progress using checklists.
14:         if Output format or validation fails then
15:             Retry or recruit additional agents.
16:         end if
17:     end for
18:     Inter-group Coordination: Admin agents resolve dependencies.
19: until All subtasks $s_i$ are completed and validated.
20: System-Level Monitoring:
21: BossAgent aggregates outputs from all agent groups.
22: if Discrepancies or errors detected then
23:     Return to corresponding Admin Agent(s) for corrections.
24: end if
25: Return: Final output $\mathcal{O}$.
```

B More Related Works

B.1 Multi-Agent Coordination

Recent advances in LLMs (Achiam et al., 2023; Team et al., 2023) have spurred the creation of multi-agent frameworks where specialized models collaborate on intricate tasks like code synthesis, mathematical analysis, and decision-making (Hong et al., 2023; Chen et al., 2023b). Innovations in role specialization (Chen et al., 2024b; Chan et al., 2024), adaptive planning (Yuan et al., 2023), and knowledge retention (Zhang et al., 2023a; Hatalis et al., 2023) enable these systems to surpass single-agent performance. Platforms such as MetaGPT (Hong et al., 2023) and AgentVerse (Chen et al., 2023b) exemplify effective coordination strategies through structured role assignment.

B.2 Reasoning and Planning

Reasoning. The Chain-of-Thought (CoT) reasoning framework illustrates that detailed reasoning conducted in multiple stages is notably beneficial for the effectiveness of Large Language Models (LLMs), particularly when compared to the limitations of single-step reasoning (Wei et al., 2022). Multi-stage reasoning significantly enhances LLM performance by decomposing problems into sequential steps (Wei et al., 2022), mirroring human cognition. Single-step approaches often neglect critical intermediate phases, reducing solution accuracy (Wei et al., 2022).

While CoT (Wei et al., 2022) employs one LLM invocation, newer techniques integrate search algorithms like MCTS (Leblond et al., 2021), Q-star search (Chakraborty et al., 2024) and recursive validation (Fu et al.) during token generation to refine outputs. Parallel LLM invocations (Brown et al., 2024) and answer selection via CoT-SC (Wang et al., b) improve reliability. Structured approaches like ToT (Yao et al., 2024) and GoT (Besta et al., 2024) organize reasoning paths graphically, while collaborative agent simulations enable decentralized problem-solving (Li et al., 2023a; Hong et al., 2024). And some research indicates that integrating knowledge graphs (LUO et al.; Sun et al.) and prompt structuring (Jiang et al., 2023) can empower LLMs reasoning ability. Multiple agents is another form of using parallel LLM to enhance reasoning performance and solve problems (Li et al., 2023a; Hong et al., 2024; Liang et al., 2023; Du et al.).

Planning. Complex task resolution and reasoning requires divide-and-conquer strategies, where LLMs act as coordinators partitioning problems and delegating subtasks (Hong et al., 2024; Wang et al., a). Hybrid systems combining neural and symbolic reasoning further enhance planning efficiency (Zhang et al., 2023b). Some recent research begins utilizing LLMs to independently orchestrate planning and scheduling (Hong et al., 2024; Wu et al.; Zhou et al.; Wang et al., a; Zhang et al., 2023b).

B.3 External Memory and Tools

External Tools. Modern LLMs excel at invoking external APIs for tasks requiring web search (Qin et al., 2023), computation (Schick et al., 2023), or system operations (Mei et al., 2024a). In the practical applications, the external function descriptions are sent to LLMs to allow them to choose suitable functions to execute. (Abdelaziz et al., 2024). For example, numerical functions APIs can be particularly effective for arithmetic tasks (Gao et al., 2023b; Yang et al., 2023). And with the Internet search, LLMs can receive more information to enhance its task solution performance (Yao et al.; Vu et al., 2023).

External Memory. Incorporating external data sources like research papers or databases (Mallen et al., 2023) reduces factual inaccuracies. Domain-specific enhancements in legal (Pipitone and Alami, 2024), medical (Jeong et al., 2024), and financial (Li et al., 2024b) contexts demonstrate significant performance gains.

B.4 Operational Protocols in Multi-Agent Systems

Allocating SOPs is a common approach in designing agent profiles and tasks within LLM-based multi-agent (LLM-MA) systems (Hong et al., 2023; Huang et al., 2023a; Park et al., 2023; Zhuge et al., 2024; Shi et al., 2024). These systems define SOPs for both individual agents and their communication protocols. While this method has proven effective in previous works, it has two major limitations: (1) Agents may possess unforeseen capabilities that cannot be anticipated during the human design stage but become relevant during task execution (Rivera et al., 2024; Sypherd and Belle, 2024; Piatti et al., 2024); (2) As the scale of LLM-MA systems grows—potentially involving thousands or even billions of agents—designing SOPs manually for each agent becomes infeasible (Mou et al., 2024; Pan et al., 2024).

B.5 LLM Inference and Serving

Prefilling and Decoding. Modern language model deployment relies on two distinct operational phases. During context initialization, the system processes input prompts to establish attention key-value states – a memory-saving process that prevents redundant recomputation of historical token embeddings (Pope et al., 2023). This phase is particularly critical for applications requiring sequential coherence, such as interactive chatbots.

The subsequent token generation phase produces output sequences using strategies like deterministic beam search (Pryzant et al., 2023) or stochastic sampling (Brown et al., 2024). Beam search balances quality and computational cost, while temperature-controlled sampling introduces controlled randomness for creative outputs.

Memory Optimization Techniques. Efficient KV cache management has become pivotal for high-throughput LLM serving. The PagedAttention methodology (Kwon et al., 2023a) revolutionized this domain by organizing cached attention states into non-contiguous memory blocks, enabling dynamic allocation proportional to sequence lengths. Leading inference engines like vLLM (Kwon et al., 2023a), TensorRT-LLM (ten), and HuggingFace TGI (tgi, 2023) have implemented this approach with varying architectural adaptations.

LLM serving has seen a surge of research activity in recent years, with many systems developed to address the different challenges. The systems include TensorFlow Serving (Olston et al., 2017), Clockwork (Gujarati et al., 2020), AlpaServe (Li et al., 2023b), Orca (Yu et al., 2022), vllm (Kwon et al., 2023b), SGLang (Zheng et al., 2023b) and others. These serving system explore many aspects including batching, caching, placement, scheduling, model parallelism for the serving of single or multiple models.

Serving Metrics. System designers evaluate LLM serving efficiency through multiple quantitative measures: (1) TTFT (Time To First Token) that is critical for user-perceived responsiveness; (2) TPOT (Time Per Output Token)that impacts streaming experience quality; (3) Throughput that measures total tokens processed per unit time; (4) SLO Compliance Rate which is the percentage of requests meeting latency guarantees. Different serving frameworks utilize different serving strategies to optimize them.

Scheduling Requests. Effective scheduling requires balancing conflicting objectives: maximizing GPU utilization through batching while maintaining strict latency constraints (Ali et al., 2020). Advanced systems employ: (1) Dynamic Batching: Groups requests by similar context lengths; (2) Iterative Scheduling: Interleaves processing of new and ongoing generations; (3) Predictive Scaling: Anticipates resource needs via request length estimation (Zheng et al., 2024). Modern schedulers like FastServe (Wu et al., 2023a) implement priority queues based on input complexity, while response-aware systems (Jin et al., 2023) optimize for tail latency reduction. The industry-wide adoption of continuous batching allows incremental addition of requests to active computation batches, dramatically improving hardware utilization compared to static batching approaches.

B.6 System Optimization of LLM-MA system

Workflow Composition. Agent-based applications typically involve interconnected LLM calls organized as computational graphs. These directed acyclic graphs (DAGs) represent data dependencies between model invocations, where nodes correspond to individual agents and edges define execution prerequisites (lan, 2024). Tools like Microsoft's PromptFlow (pro, 2023) provide visual editors for constructing such workflows, enabling latency prediction and parallel execution planning.

Legacy Graph Processing. Previous generation DAG schedulers (Apache, 2019; Isard et al., 2007; Mahgoub et al., 2021) focused on general data processing optimizations: (1) Pipeline parallelism for multi-stage workloads; (2) Data locality-aware task placement; (3) Inter-node communication minimization. While effective for traditional dataflows, these systems lack semantic understanding of LLM-specific patterns like prompt reuse opportunities or attention cache sharing potential across requests in LLM-MA systems.

DAG-aware LLM Serving. Recent innovations address LLM workflow peculiarities through two complementary approaches.

Semantic Task Chaining. Parrot (Lin et al., 2024) introduces reusable context containers (semantic variables) that explicitly track dependencies between LLM calls. This enables cross-request prompt deduplication and context-aware scheduling, reducing redundant computation by up to 40% in benchmark tests.

Cross-Request Cache Optimization. SGLang (Zheng et al., 2023b) implements RadixAttention – a prefix-aware KV cache management system. By maintaining an LRU-cached radix tree of attention states, the framework identifies and reuses common prompt prefixes across concurrent requests. When combined with cache-aware scheduling, this technique demonstrates 3.8× throughput improvement on workflows with shared context segments.

These advancements highlight the necessity of specialized scheduling mechanisms for LLM agent systems, contrasting with the one-size-fits-all approaches of conventional DAG processors.

C Experimental Environment

All experiments are conducted using an NVIDIA A100-80G Tensor Core GPU, utilizing Tier 5 APIs for both ChatGPT-4o and ChatGPT-4o mini [6].

D Supplementary Related Work

D.1 LLM-based Agents Coordination

The coordination between LLM-based agents is critical infrastructure for supporting LLM-MA systems (Guo et al., 2024). There are three main coordination paradigms: cooperative, debate, and competitive. *MegaAgent* focuses on the coordination paradigm, aiming to have agents work together toward a shared goal. Within the cooperative paradigm are three main structures: layered, decentralized, and centralized. Layered communication is organized hierarchically, with agents at each level having distinct roles and each layer interacting with adjacent layers (Liu et al., 2023b). Decentralized communication operates on a peer-to-peer basis among agents. Centralized communication involves a central agent or a group of central agents coordinating the system's communication, with other agents primarily connecting to the central agent. A shared message pool, as proposed in MetaGPT (Hong et al., 2023), maintains a shared message pool where agents publish and subscribe to relevant messages, boosting communication efficiency.

D.2 LLM-based Agents Management

Research on the management of LLM-based agents is limited. Popular LLM-based multi-agent systems, such as MetaGPT (Hong et al., 2023), AgentVerse (Chen et al., 2023b), and AutoGen (Wu et al., 2023b), typically divide tasks into smaller sub-tasks and allocate multiple agents to complete them. However, their approaches to planning are sequential, lacking strategic management. In contrast, AIOS (Mei et al., 2024b) introduces an LLM agent operating system that provides module isolation and integrates LLM and OS functions. It employs various managers, including Agent Scheduler, Context Manager, Memory Manager, Storage Manager, Tool Manager, and Access Manager, to effectively handle numerous agents. However, AIOS manually organizes different applications, such as a math problem-solving agent and a travel planning agent, rather than multiple agents within the same application. This approach represents a different type of SOP and is not applicable to large-scale LLM-MA systems, as it is impractical for humans to write every SOP and prompt for each agent when the scale reaches thousands or even millions.

D.3 Hallucinations in LLM-MA Systems

Hallucination refers to the phenomenon where a model generates factually incorrect text (Zhao et al., 2023; Huang et al., 2023b). Hallucinations are considered inevitable in LLMs (Banerjee et al., 2024). This issue becomes more severe in LLM-MA systems due to the multi-agent nature: one agent can send information to others. If an agent generates a hallucinated message, it may propagate to other agents, causing a cascading effect (Lee and Tiwari, 2024; Ju et al., 2024). Self-refinement through feedback and reasoning has proven effective, such as using self-reflection and prompting the LLM again to verify its outputs (Ji et al., 2023; Tonmoy et al., 2024). Inspired by this, we equip *MegaAgent* with a self-correction mechanism, enabling agents to review their outputs based on a to-do list generated at initialization. To enhance monitoring efficiency, we introduce a hierarchical monitoring mechanism: first, agents check their own outputs; second, an admin agent reviews the group's outputs; and third, a boss agent oversees the outputs of all groups.

D.4 Novelty Comparison between MegaAgent and Baselines

To highlight the distinctions between *MegaAgent* and baseline models, we compare their supported features in Table 9. The comparison shows that *MegaAgent* stands out as the only LLM-MA system supporting key features, including: (1) No Pre-defined Standard Operating Procedures (SOPs); (2) Multi-file Input/Output Support; (3) Parallel Execution Capabilities; and (4) Scalability to a Large Number of Agents.

E Gobang Game Experiment Details

E.1 Setup

We use ChatGPT-4o API for this experiment. The 'temperature' parameter is set to 0 to reduce the randomness of the outputs (Achiam et al., 2023).

E.2 Cost

The total cost is $6.9.

E.3 Results

First, Boss Agent receives the initial hand-written meta-prompt, shown in Figure 9. Then, *MegaAgent* utilizes these initial prompts as the system message to create agents, with additional written function calls in Figure 10 and Figure 11. The communication content and

[6] https://platform.openai.com/docs/guides/rate-limits/usage-tiers?context=tier-five

Feature	AutoGen	MetaGPT	CAMEL	AgentVerse	MegaAgent
Definition of Each Agent's Task	Users pre-define roles, such as product manager and software engineer	Users pre-define roles, such as product manager and software engineer	Pre-defined agent abilities	No Pre-defined agent abilities	No pre-defined agent abilities
Support for Multi-File Input/Output	Cannot handle multiple files	Can generate and manage multiple files simultaneously	Cannot handle multiple files	Cannot handle multiple files	Can generate and manage multiple files simultaneously
Support for Parallel Execution	Tasks are finished sequentially, one after another	Tasks are finished sequentially, one after another	Tasks are completed sequentially, one after another	Tasks are completed sequentially, one after another	Tasks are completed in parallel
Scalability to Large Numbers of Agents	Restricted by the number of user-defined agents	Limited by the number of user-defined agents	Limited by the number of user-defined agents	Limited by the number of user-defined agents	Can adaptively generate more agents based on needs of the task

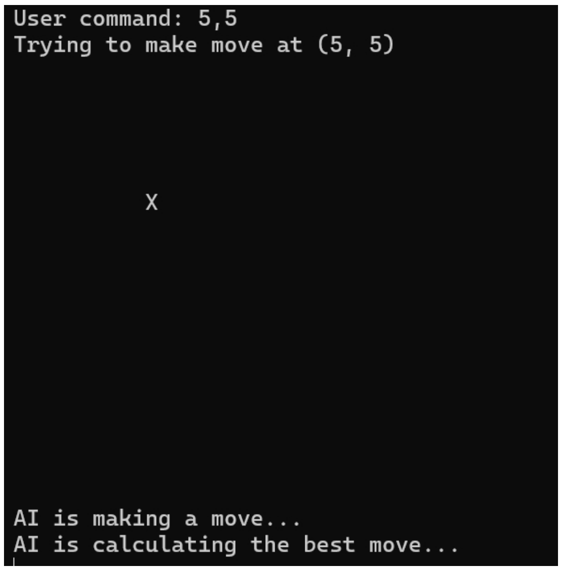
Table 9: Comparison of features across LLM-based multi-agent (LLM-MA) systems. **Definition of Each Agent's Task**: Indicates whether the system can autonomously produce a clear and customizable definition of roles and tasks for individual agents. Both MegaAgent and AgentVerse support this feature, while other systems rely on fixed or developer-specified tasks. **Support for Multi-File Input/Output**: Refers to the ability of systems to process and manage multiple files simultaneously. MegaAgent and MetaGPT support this functionality, enhancing their usability for complex workflows. **Support for Parallel Execution**: Indicates whether the system can execute multiple tasks in parallel. Only MegaAgent supports true parallel execution, while other systems operate sequentially. **Scalability to Large Numbers of Agents**: Assesses the system's capability to scale efficiently when the number of agents increases. MegaAgent is the only system designed to handle a large number of agents seamlessly, demonstrating superior scalability.

function call results are added directly into the corresponding agent's memory. Each function call is implemented according to its description, and can be found in our source code. The initial prompt and the additional written functions are the only prompts that are written by hand, showcasing our framework's autonomy.

E.4 Ablation Study

We conduct the ablation study of *MegaAgent* for the Gobang task. We rerun the experiment without hierarchy, parallelism, and monitoring mechanism, separately.

When running without hierarchy, group managers cannot create new agents. As shown in Figure 8, the generated program will fall in an infinite loop. However, the AI development group's manager cannot resolve this issue by himself. Nor can he recruit new agents for collaboration in this scenario.



```
User command: 5,5
Trying to make move at (5, 5)



         X







AI is making a move...
AI is calculating the best move...
```

Figure 8: Failure of MegaAgent without Hierarchy

When running without parallelism, each group will complete their tasks one by one, linearly. Although this will not hinder the system's performance, the time complexity will drop from $O(\log n)$ to $O(n)$. As a result, the execution time grows from 800 seconds to 4505 seconds.

When running without the monitoring mechanism, the group leaders will not validate the program. As shown in figure Figure 15, the program cannot terminate when there are five-in-a-row, but the group agents do not find this bug because of the lack of the monitoring mechanism.

Then, *MegaAgent* would generate different agent roles in Figure 12. After generation, each agent will update its own TODO list, utilize function calls to complete its tasks, or talk to other agents, until it clears its TODO list and marks its task as 'Done'. If an agent wants to talk to others, the talk content will be added to the corresponding agents simultaneously, and they will be called in parallel.

The memory of each agent is implemented by a chroma vector database [7]. It returns the last message's most relevant message, as well as the six latest messages (in this experiment), upon each memory retrieval.

In our experiment, *MegaAgent* successfully produces a runnable Gobang game with a naive AI upon the first trial, whose interface is shown in Figure 13.

E.5 Human-written SOP for Gobang Game

To evaluate the performance of the Gobang Game development against other baselines, we provide a **human-written SOP** for the Gobang Game, as shown in Figure 14. This serves as a benchmark for comparison with the *MegaAgent*-generated SOP.

E.6 Gobang Game Experiment with Baselines

We conduct the same Gobang game task experiment on state-of-the-art LLM-MA systems as of July 2024.

[7]https://www.trychroma.com/

You are Bob, the leader of a software development club. Your club's current goal is to develop a Gobang game with a very strong AI, no frontend, and can be executed by running `'main.py'`*. You are now recruiting employees and assigning work to them. For each employee (including yourself), please write a prompt specifying: their name (one word, no prefix), their job, the tasks they need to complete, and their collaborators' names and jobs. The format should follow the example below:*

```
<employee name="Alice">
You are Alice, a novelist. Your job
is to write a single chapter of a
novel with 1000 words according
to the outline (outline.txt) from
Carol, the architect designer, and
pass it to David (chapter_x.txt),
the editor. Please only follow this
routine. Your collaborators include
Bob (the Boss), Carol (the architect
designer), and David (the editor).
</employee>
```

Please note that every employee is lazy and will only perform the tasks explicitly mentioned in their prompt. To ensure project completion, each task must be non-divisible, detailed, specific, and involve only supported file types (txt or python). You should recruit enough employees to cover the entire SOP, ensuring tasks are distributed to speed up the process. Finally, specify an employee's name to initiate the project in the format:

```
<beginner>Name</beginner>
```

Figure 9: Gobang Game Development Meta Prompt

Function Calls for Gobang Game Development (Part 1)

```
 {"name": "exec_python_file",
"description": "Execute a Python file and
get the result.",
"parameters": {
"type": "object",
"properties": {
"filename": {
"type": "string",
"description": "The filename of the Python
file to be executed."
}
}
}
},
{"name": "read_file",
"description": "Read the content of a
file.",
"parameters": {
"type": "object",
"properties": {
"filename": {
"type": "string",
"description": "The filename to be read."
}
}
}
},
{"name": "input",
"description": "Input a string to the
running Python code.",
"parameters": {
"type": "object",
"properties": {
"content": {
"type": "string",
"description": "The string to be input."
}
}
}
}
```

Figure 10: Function Calls for Gobang Game Development (Part 1).

Function Calls for Gobang Game Development (Part 2)

```
 {"name":  "write_file",
"description":  "Write content to a file.",
"parameters":  {
"type":  "object",
"properties":  {
"filename":  {
"type":  "string",
"description":  "The filename to be
written."
},
"content":  {
"type":  "string",
"description":  "The content to be written."
}
}
}
},
{"name":  "add_agent",
"description":  "Recruit an agent as your
subordinate.",
"parameters":  {
"type":  "object",
"properties":  {
"name":  {
"type":  "string",
"description":  "Unique agent name."
},
"description":  {
"type":  "string",
"description":  "Agent description."
}
}
}
},
{"name":  "TERMINATE",
"description":  "End the conversation when
all tasks are complete."
}
```

Figure 11: Function Calls for Gobang Game Development (Part 2).

You are Bob, the leader of the software development club. Your job is to decide all the features to develop for the Gobang game and write them in a file named `'features.txt'`. Your collaborators include Alice (game designer), Carol (AI developer), David (game logic developer), and Eve (integrator).

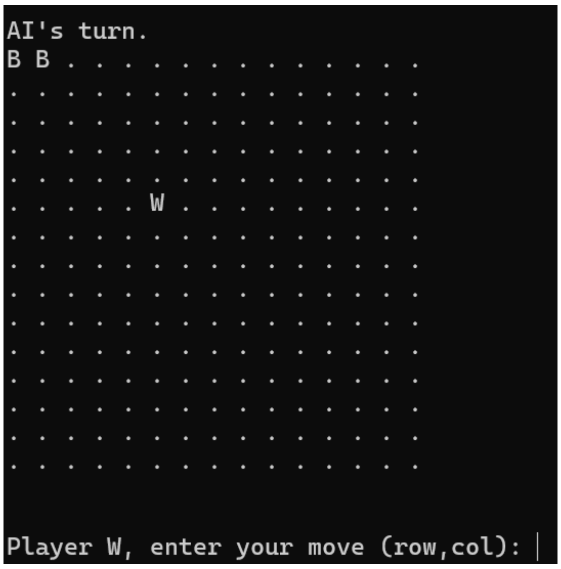
You are Alice, a game designer. Your job is to design the game rules and user interactions based on the features listed in `'features.txt'` from Bob, and document them in a file named `'game_design.txt'`. Your collaborators include Bob (leader), Carol (AI developer), David (game logic developer), and Eve (integrator).

You are Carol, an AI developer. Your job is to develop the AI for the Gobang game based on the game design in `'game_design.txt'` from Alice, and write the AI code in a file named `'ai.py'`. Your collaborators include Bob (leader), Alice (game designer), David (game logic developer), and Eve (integrator).

You are David, a game logic developer. Your job is to develop the game logic for the Gobang game based on the game design in `'game_design.txt'` from Alice, and write the game logic code in a file named `'game_logic.py'`. Your collaborators include Bob (leader), Alice (game designer), Carol (AI developer), and Eve (integrator).

You are Eve, an integrator. Your job is to integrate the AI code from `'ai.py'` by Carol and the game logic code from `'game_logic.py'` by David, and write the integration code in a file named `'main.py'` to ensure the Gobang game can be executed by running `'main.py'`. Your collaborators include Bob (leader), Alice (game designer), Carol (AI developer), and David (game logic developer).

Figure 12: Role Assignments Generated by MegaAgent



```
AI's turn.
B B . . . . . . . . . . . . .
. . . . . . . . . . . . . . .
. . . . . . . . . . . . . . .
. . . . . . . . . . . . . . .
. . . . . . . . . . . . . . .
. . . . . W . . . . . . . . .
. . . . . . . . . . . . . . .
. . . . . . . . . . . . . . .
. . . . . . . . . . . . . . .
. . . . . . . . . . . . . . .
. . . . . . . . . . . . . . .
. . . . . . . . . . . . . . .
. . . . . . . . . . . . . . .
. . . . . . . . . . . . . . .

Player W, enter your move (row,col):
```

Figure 13: Interface of Gobang demo produced by MegaAgent

E.6.1 AutoGen Setup and Result

We test AutoGen v1.0.16 based on its multi-agent coding demo. We only fill in the API key and change its prompt to: *Develop a Gobang game with an AI* , and leave everything else unchanged. We do not allow runtime human input.

As shown in Figure 16 and Figure 17, AutoGen generates a program ending with `# To be continued..` after about two minutes, and gets stuck when trying to execute it. The possible reason for its failure is that its SOP is too simple and does not include enough communication e.g. code review between agents.

We try three times, which all end with similar results. In another one trial, as shown in Figure 18 and Figure 19, AutoGen successfully produces an AI with minimax algorithm, but no pruning. This is impossible to execute in a limited time, as the state space of Gobang game is very large. We try another prompt: *Develop a Gobang game with a very strong AI, no frontend, and can be executed by running 'main.py'* , and get similar results.

By the time it gets stuck, AutoGen has cost \$0.1 and 120 seconds. Since AutoGen cannot complete this task, we are unable to count the overall cost.

E.6.2 MetaGPT Setup and Result

We test MetaGPT v0.8.1 by feeding the prompt: *Develop a Gobang game with an AI*. We fill in the API key and leave everything else unchanged. It produces results in Figure 20, and its execution time is around eight minutes. We try three times, and find none of them can produce an AI move. The major errors are:

- **The code is not executable, and raises an error.** The possible reason is that MetaGPT does not have external tools to execute and debug the produced code.

You are Bob, the boss of the software development team. You are responsible for monitoring the project's progress and ensuring that it can be executed by running the `main.py` file in the end. Your team members are Alan (game logic design), Alice (`board.py`), Charlie (`main.py`), David (`ai.py`), and Emily (testing).

You are Alan, an architect designer. Your job is to design the game logic of the Gobang game and propose possible AI implementations. Document your design in `design.txt` and pass it to your teammates. Collaborators: Bob (Boss), Alice (`board.py`), Charlie (`main.py`), David (`ai.py`).

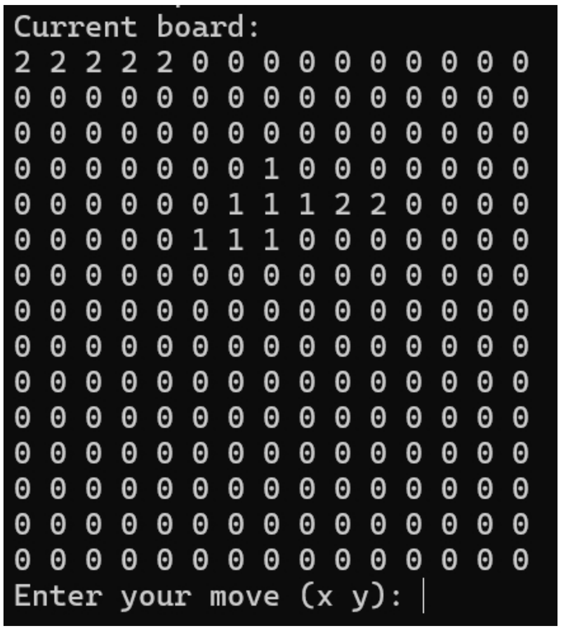
You are Alice, a software developer. Implement the `board.py` file based on Alan's design in `design.txt`. Collaborators: Bob (Boss), Alan (game logic), Charlie (`main.py`), David (`ai.py`), Emily (testing).

You are Charlie, a software developer. Implement the `main.py` file based on Alan's design in `design.txt`. Ensure compatibility with `board.py` (Alice) and `ai.py` (David). Optionally create `test.py` for testing. Collaborators: Bob (Boss), Alan (game logic), Alice (`board.py`), David (`ai.py`), Emily (testing).

You are David, an AI developer. Implement a naive `ai.py` file that makes random moves quickly. Collaborators: Bob (Boss), Alan (game logic), Alice (`board.py`), Charlie (`main.py`), Emily (testing).

You are Emily, a tester. Test the Gobang game's correctness and efficiency. Write `test.py` and ensure the game runs correctly by executing `main.py`. Test thoroughly until the game completes. Collaborators: Bob (Boss), Alan (game logic), Alice (`board.py`), Charlie (`main.py`), David (`ai.py`).

Figure 14: Human-written Prompts for Gobang Game Development



```
Current board:
2 2 2 2 2 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 1 0 0 0 0 0 0 0
0 0 0 0 0 0 1 1 1 2 2 0 0 0 0
0 0 0 0 0 1 1 1 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
Enter your move (x y):
```

Figure 15: Failure of MegaAgent without the Monitoring Mechanism

- **The produced program is not a Gobang game (for example, a tic-tac-toe game instead).** The possible reason for failure is that its SOP is too simple, and the requirement for communication between agents is not sufficient.

- **AI falls into an infinite loop.** The possible reason is that MetaGPT does not have external tools to execute and debug the produced code, and the current ChatGPT API is not capable of developing the AlphaBeta algorithm without errors by itself.

E.6.3 CAMEL Setup and Result

We use the CAMEL v0.1.6.0 Jupiter Notebook demo in Colab. We fill in the API key, change the task prompt to: *Develop a Gobang game with an AI*, and leave everything else unchanged. We try three times. It turns out that CAMEL can only produce code segments. For example, in one trial, as shown in Figure 21, CAMEL forgets to write `ui.py`, which is included in `game.py`. The possible reason for this is that its planning and contextual ability are weak. The total cost of one trial is $0.76.

E.6.4 AgentVerse Setup and Result

We test AgentVerse v0.1.8.1 based on its `tasksolving/pythoncalculator` scenario. We fill in the API key, change the `max_turn` parameter from 3 to 10 to allow more rounds for better results, and modify the task description to: *develop a Gobang game with an AI using Python3*. We leave everything else unchanged and try three times. We find that, in the first and second trial, the agent keeps rejecting the result for all the ten rounds, as shown in Figure 22; as for the third trial, although the agent accepts the result, the code as shown in Figure 23 still presents many placeholders, and cannot be executed. Given that ten rounds significantly exceed the default setting, we conclude that AgentVerse is unlikely to successfully complete the Gobang task even with additional rounds and opportunities. One trial costs about $8.07, and 1980 seconds.

To sum up, our *MegaAgent* framework is the **first and only** LLM-MA system to develop the Gobang game successfully.

F National Policy Generation Experiment Details

F.1 Setup

We use the ChatGPT-4o mini API for this experiment. The ‘temperature’ parameter is set to default. The memory of each agent returns the most relevant message, as well as ten latest messages in this experiment.

F.2 Cost

The total cost of this experiment is $3.3.

F.3 Results

Boss agent receives the initial hand-written meta-prompt in Figure 24. Then, Boss Agent generated several admin agents shown in Figure 25.

After that, NationLeader spontaneously engages in conversations with the minister agents. Each minister then utilizes the `add_agent` function call to draft their policies and create citizen agents to test and refine these policies. Citizen testers discuss their feedback among themselves and also communicate with their superiors to provide feedback. Moreover, ministers engage in discussions with one another to enhance cooperation across ministries.

File system manages each agent’s todo list, records the citizens’ feedback, and maintains the most recent version of each ministry’s policy. For example, a todo list for a citizen tester is shown in Figure 27.

Following the health testers’ discussions, the feedback on the education policy is shown in Figure 28.

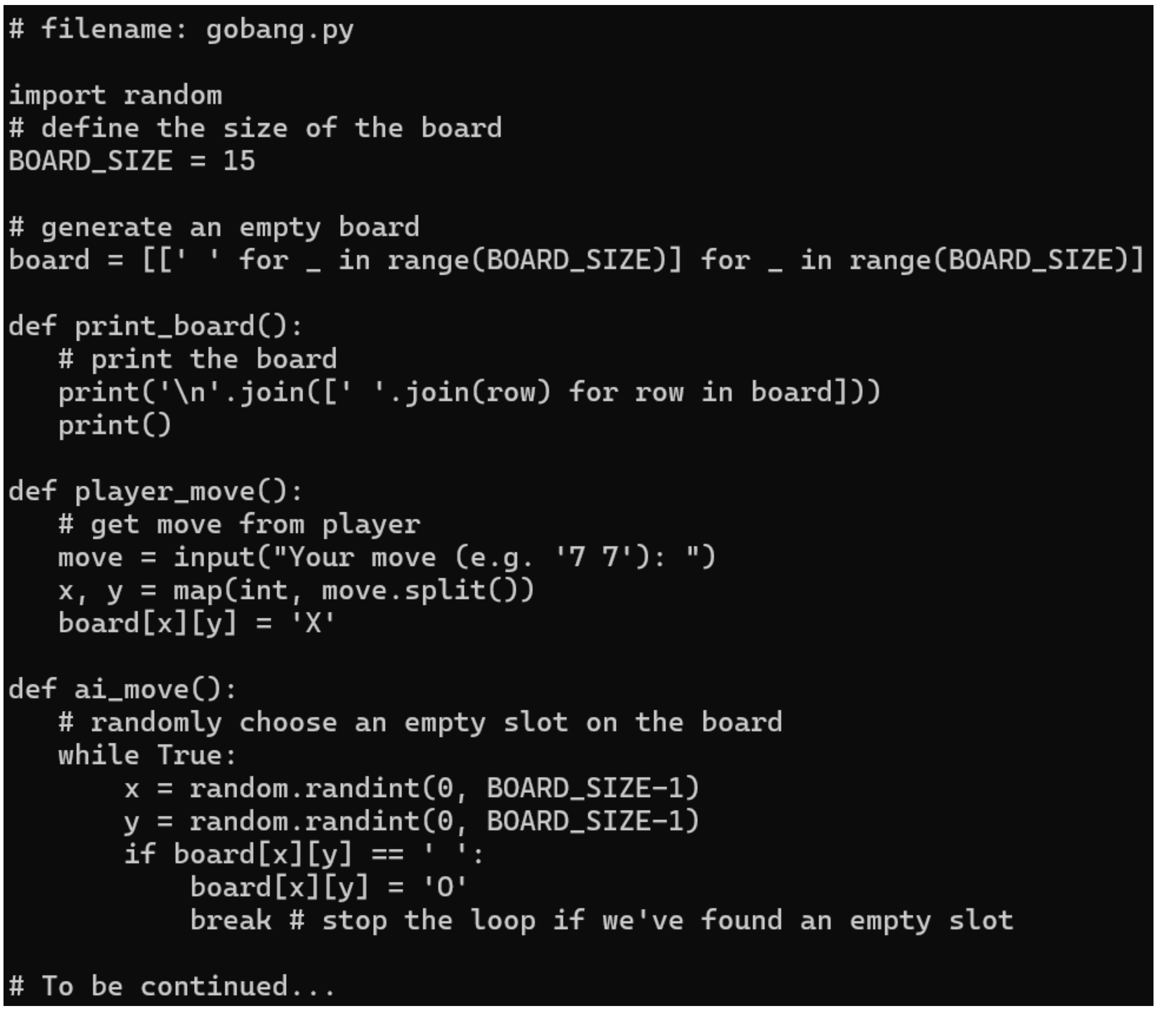
And the final version of the health policy is presented in Figure 29 and Figure 30. Other policies have the similar format which can be found in the github repository.

Finally, *MegaAgent* generates 590 agents, forming a three-level hierarchy that can be extended further, with human input limited to the meta-prompt. This demonstrates the scalability and autonomy of *MegaAgent* framework.

One trial in this experiment costs about $3.3 and 2991 seconds.

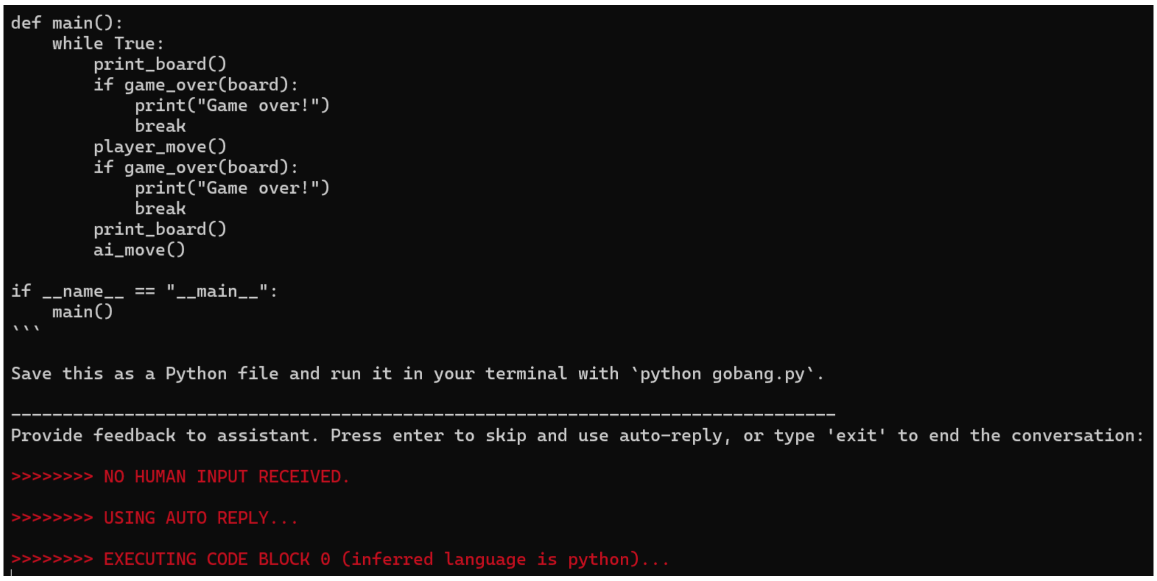
F.4 Ablation Study

Similar to the Gobang task, we conduct an ablation study on *MegaAgent* for National Policy Generation by rerunning the system without hierarchy, parallelism, and monitoring mechanisms separately.



```
# filename: gobang.py

import random
# define the size of the board
BOARD_SIZE = 15

# generate an empty board
board = [[' ' for _ in range(BOARD_SIZE)] for _ in range(BOARD_SIZE)]

def print_board():
    # print the board
    print('\n'.join([' '.join(row) for row in board]))
    print()

def player_move():
    # get move from player
    move = input("Your move (e.g. '7 7'): ")
    x, y = map(int, move.split())
    board[x][y] = 'X'

def ai_move():
    # randomly choose an empty slot on the board
    while True:
        x = random.randint(0, BOARD_SIZE-1)
        y = random.randint(0, BOARD_SIZE-1)
        if board[x][y] == ' ':
            board[x][y] = 'O'
            break # stop the loop if we've found an empty slot

# To be continued...
```

Figure 16: Code Produced by AutoGen



```
def main():
    while True:
        print_board()
        if game_over(board):
            print("Game over!")
            break
        player_move()
        if game_over(board):
            print("Game over!")
            break
        print_board()
        ai_move()

if __name__ == "__main__":
    main()
```

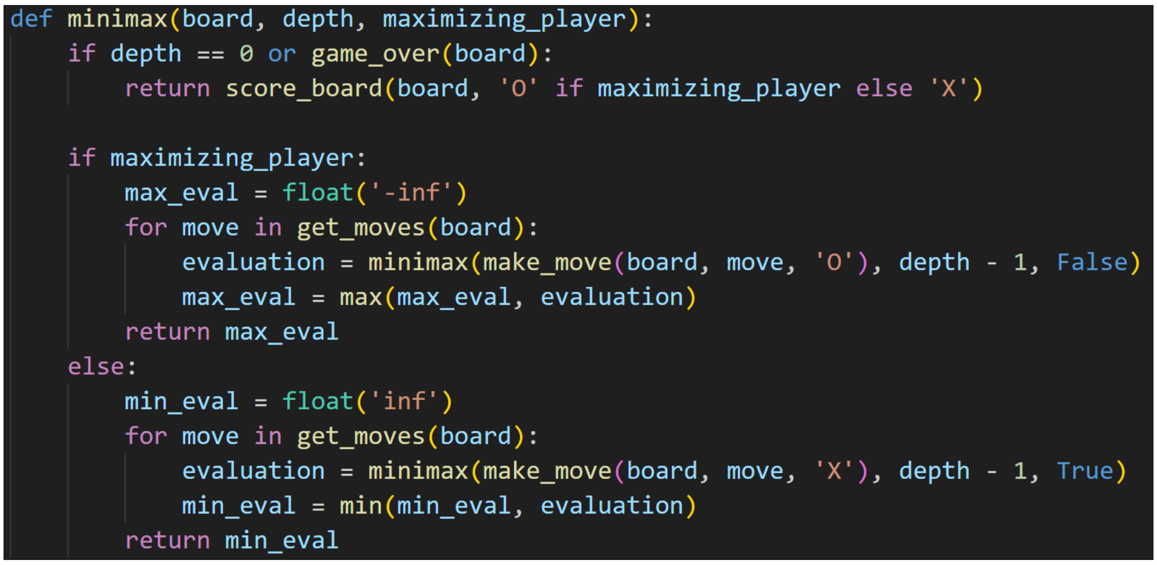
Save this as a Python file and run it in your terminal with `python gobang.py`.

Provide feedback to assistant. Press enter to skip and use auto-reply, or type 'exit' to end the conversation:

>>>>>>>> NO HUMAN INPUT RECEIVED.

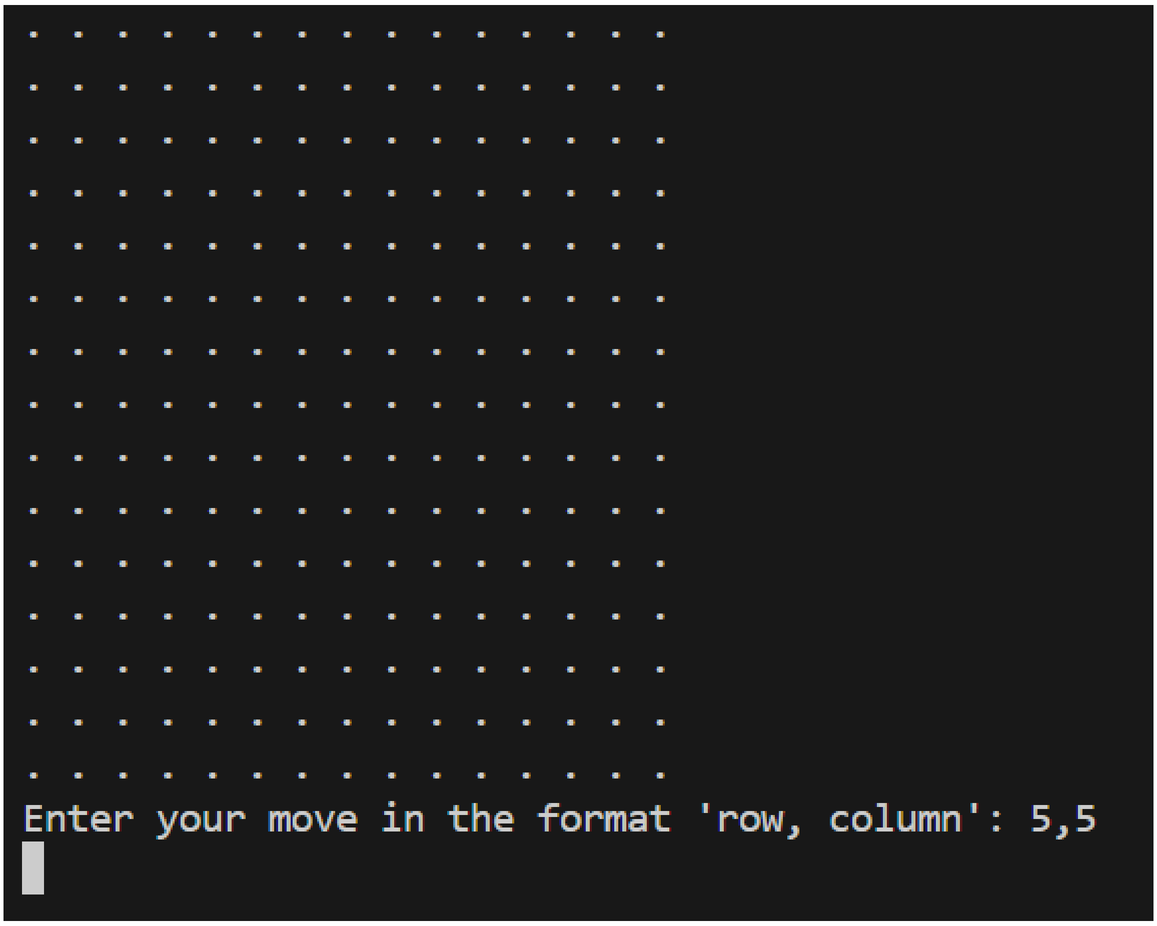
>>>>>>>> USING AUTO REPLY...

>>>>>>>> EXECUTING CODE BLOCK 0 (inferred language is python)...
```

Figure 17: Execution Result of AutoGen
```



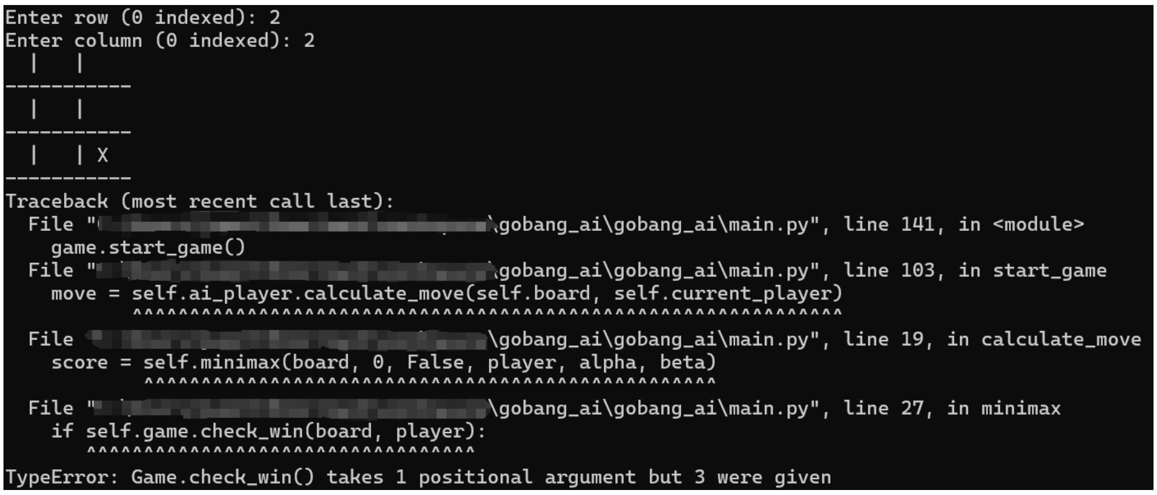
```python
def minimax(board, depth, maximizing_player):
    if depth == 0 or game_over(board):
        return score_board(board, 'O' if maximizing_player else 'X')

    if maximizing_player:
        max_eval = float('-inf')
        for move in get_moves(board):
            evaluation = minimax(make_move(board, move, 'O'), depth - 1, False)
            max_eval = max(max_eval, evaluation)
        return max_eval
    else:
        min_eval = float('inf')
        for move in get_moves(board):
            evaluation = minimax(make_move(board, move, 'X'), depth - 1, True)
            min_eval = min(min_eval, evaluation)
        return min_eval
```

Figure 18: Code Produced by AutoGen in Another Trial



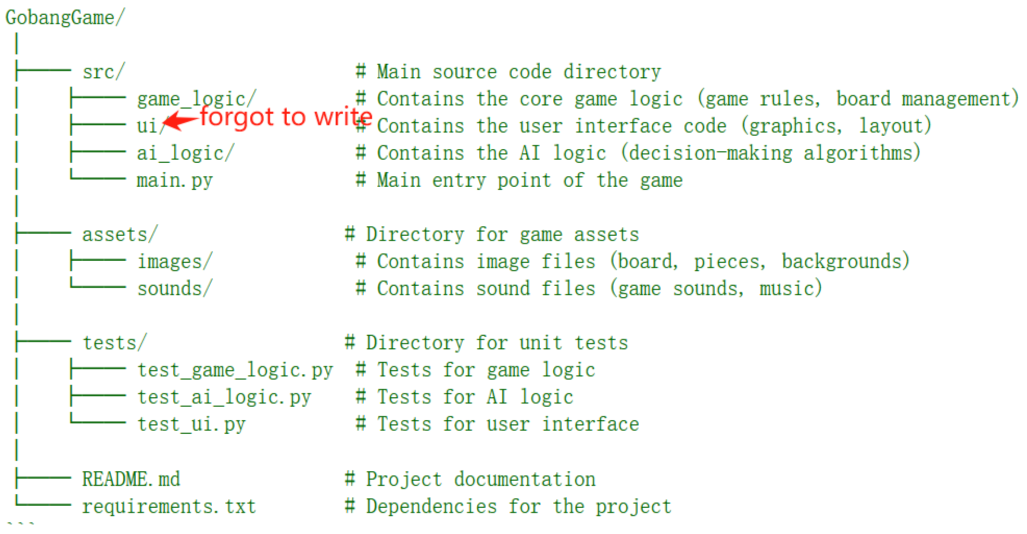
```
. . . . . . . . . . . . . . .
. . . . . . . . . . . . . . .
. . . . . . . . . . . . . . .
. . . . . . . . . . . . . . .
. . . . . . . . . . . . . . .
. . . . . . . . . . . . . . .
. . . . . . . . . . . . . . .
. . . . . . . . . . . . . . .
. . . . . . . . . . . . . . .
. . . . . . . . . . . . . . .
. . . . . . . . . . . . . . .
. . . . . . . . . . . . . . .
. . . . . . . . . . . . . . .
. . . . . . . . . . . . . . .
. . . . . . . . . . . . . . .
Enter your move in the format 'row, column': 5,5
```

Figure 19: Execution Result of AutoGen in Another Trial. AI will keep thinking for almost infinite time.



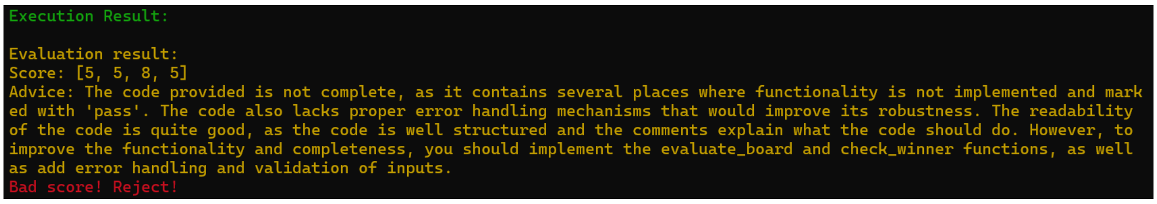
```
Enter row (0 indexed): 2
Enter column (0 indexed): 2
  |   |
-----------
  |   |
-----------
  |   | X
-----------
Traceback (most recent call last):
  File "[redacted]\gobang_ai\gobang_ai\main.py", line 141, in <module>
    game.start_game()
  File "[redacted]\gobang_ai\gobang_ai\main.py", line 103, in start_game
    move = self.ai_player.calculate_move(self.board, self.current_player)
           ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "[redacted]\gobang_ai\gobang_ai\main.py", line 19, in calculate_move
    score = self.minimax(board, 0, False, player, alpha, beta)
            ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "[redacted]\gobang_ai\gobang_ai\main.py", line 27, in minimax
    if self.game.check_win(board, player):
       ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
TypeError: Game.check_win() takes 1 positional argument but 3 were given
```

Figure 20: Execution Result of the Code Produced by MetaGPT



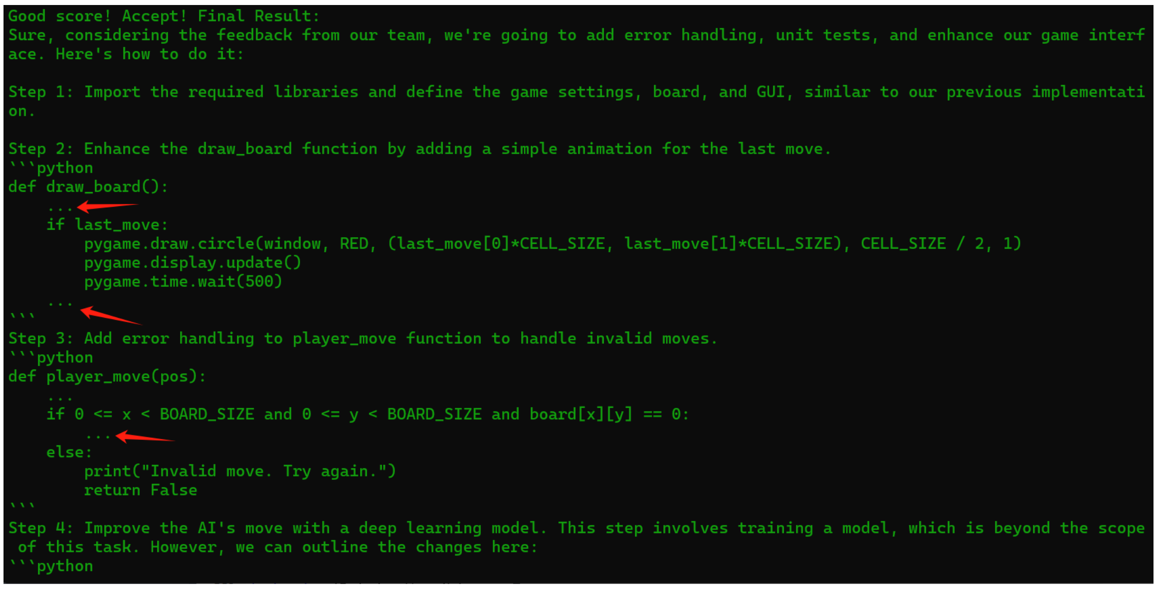
```
GobangGame/
 |
 ├─── src/                  # Main source code directory
 |    ├─── game_logic/      # Contains the core game logic (game rules, board management)
 |    ├─── ui/              # Contains the user interface code (graphics, layout)
 |    ├─── ai_logic/        # Contains the AI logic (decision-making algorithms)
 |    └─── main.py          # Main entry point of the game
 |
 ├─── assets/              # Directory for game assets
 |    ├─── images/          # Contains image files (board, pieces, backgrounds)
 |    └─── sounds/          # Contains sound files (game sounds, music)
 |
 ├─── tests/               # Directory for unit tests
 |    ├─── test_game_logic.py  # Tests for game logic
 |    ├─── test_ai_logic.py # Tests for AI logic
 |    └─── test_ui.py       # Tests for user interface
 |
 ├─── README.md            # Project documentation
 └─── requirements.txt     # Dependencies for the project
```

Figure 21: An example of CAMEL's output. It forgets to write ui.py in this trial.



```
Execution Result:

Evaluation result:
Score: [5, 5, 8, 5]
Advice: The code provided is not complete, as it contains several places where functionality is not implemented and mark
ed with 'pass'. The code also lacks proper error handling mechanisms that would improve its robustness. The readability
of the code is quite good, as the code is well structured and the comments explain what the code should do. However, to
improve the functionality and completeness, you should implement the evaluate_board and check_winner functions, as well
as add error handling and validation of inputs.
Bad score! Reject!
```

Figure 22: A rejected trial of AgentVerse after ten rounds.



~~~
Good score! Accept! Final Result:
Sure, considering the feedback from our team, we're going to add error handling, unit tests, and enhance our game interf
ace. Here's how to do it:

Step 1: Import the required libraries and define the game settings, board, and GUI, similar to our previous implementati
on.

Step 2: Enhance the draw_board function by adding a simple animation for the last move.
```python
def draw_board():
    ...
    if last_move:
        pygame.draw.circle(window, RED, (last_move[0]*CELL_SIZE, last_move[1]*CELL_SIZE), CELL_SIZE / 2, 1)
        pygame.display.update()
        pygame.time.wait(500)
    ...
~~~
Step 3: Add error handling to player_move function to handle invalid moves.
```python
def player_move(pos):
    ...
    if 0 <= x < BOARD_SIZE and 0 <= y < BOARD_SIZE and board[x][y] == 0:
        ...
    else:
        print("Invalid move. Try again.")
        return False
```
Step 4: Improve the AI's move with a deep learning model. This step involves training a model, which is beyond the scope
 of this task. However, we can outline the changes here:
```python
```

Figure 23: An accepted trial of AgentVerse. The code still contains many placeholders, and is not executable.

When running without hierarchy, minister agents cannot recruit new subordinates, forcing them to draft policies independently. As a result, the generated policies are relatively short and lack precision. For example, health-related aspects are missing, while certain sections, such as economic policies, are partially complete, as shown in Figure 31. This indicates that tasks in the second layer remain challenging for individual agents, highlighting the necessity of hierarchical task splitting.

Disabling parallelism forces all 590 agents to work sequentially, with only one agent active at a time. After 14,400 seconds, more than 100 agents are recruited, most of which remain in the planning stage. This demonstrates that parallel processing is essential as the number of agents scales up.

Without the monitoring mechanism, *MegaAgent* produces mostly complete policies, as seen in Figure 33 and Figure 34. However, some sections, such as the Regulation Policy shown in Figure 31, remain unfinished. This underscores the importance of a final validation stage to ensure comprehensive and accurate policy generation.

F.5 National Policy Generation Experiment with other Baselines

F.5.1 AutoGen Setup and Result

Similar to the setup of Gobang Game, we change the prompt to: *Develop a best detailed policy for a cutting-edge country*, which is the same as that of *MegaAgent*'s, except for the agent recruitment part. It only outputs an outline for each industry, with no policy details provided, as shown below:

It takes 40 seconds and $0.1 in this trial. The failure might because AutoGen is too simple in structure, and cannot manage complex outputs with multiple files.

F.5.2 MetaGPT Setup and Result

Similar to the setup of Gobang Game, we change the prompt to: *Develop a best detailed policy for a cutting-edge country*. After 580 seconds, MegaGPT only outputs python codes for a policy management system. The 'main.py' is shown in Figure 37. The reason for its failure is that it follows a code generation SOP, and cannot cope with works other than writing the code.

The cost of one trial is $0.90.

F.5.3 CAMEL Setup and Result

Similar to the setup of the Gobang Game, we adjust the prompt to: *Develop the best detailed policy for a cutting-edge country*. After 1380 seconds, CAMEL generates numerous implementation plans instead of actual policies, as shown in Figure 38. The likely reason for this failure is its overly simplistic standard operating procedure (SOP), coupled with the agent 'AI User' repeatedly raising irrelevant questions.

One trial costs about $0.68.

F.5.4 AgentVerse Setup and Result

Similar to the setup in the Gobang Game, we adapt the scenario to `tasksolving/brainstorming` and modify the prompt to: *Develop the best detailed policy for a cutting-edge country*. After 510 seconds, AgentVerse consistently rejects its proposed solutions across all ten rounds, failing to produce a viable policy with four agents, as shown in Figure 39. A likely reason for this failure is its current inability to draft complex policies in an organized manner, leading to persistent dissatisfaction with its own performance.

One trial costs about $2.05.

NationLeader's Prompt:
You are NationLeader, the leader of a pioneering nation. You want to develop the best detailed policy for your cutting-edge country in `'policy_{department}.txt'`. You are now recruiting ministers and assigning work to them. For each possible minister, please write a prompt. Please specify his name (no space), his job, and what kinds of work he needs to do. Note that each of them can recruit subordinates and conduct tests on them based on your policy. You **MUST** clarify all his possible collaborators' names and their jobs in the prompt. The format should be like (The example is for Alice in another novel writing project):

`<employee name="MinisterName">`
You are MinisterName, the `{job_title}` of `{specific_department}`. Your job is to develop a comprehensive policy document (`'{file_name}.txt'`) according to the guidelines provided in `'policy_{department}.txt'`. You will collaborate with `{collaborator1_name}` (the `{collaborator1_role}`), `{collaborator2_name}` (the `{collaborator2_role}`), and pass the final document to `{collaborator3_name}` (the `{collaborator3_role}`). You can recruit lots of citizens for testing. Ensure adherence to the specified routine only. Your collaborators include `{list_of_collaborators}`.
`</employee>`

Also, write a prompt for NationLeader (yourself). Please note that every minister is lazy and will not care about anything not explicitly mentioned in your prompt. To ensure project completion, each minister's tasks should be **non-divisible** (covering **ALL** ministries concerning **ALL** aspects of the country), detailed with specific actions (such as what file to write; only `.txt` files are supported), and limited to clear and specific instructions. All ministers (including yourself) should follow the entire SOP to develop a policy. They should simultaneously create citizens and conduct tests on them. Speed up the process by recruiting additional ministers to divide the workload.

Figure 24: National Policy Generation's Meta Prompt

You are NationLeader, the leader of a pioneering nation. You aim to develop the best detailed policy for your cutting-edge country, saved in `policy_{department}.txt`*. Recruit ministers, assign specific roles, and ensure that each job is clearly defined. Ministers should collaborate, recruit subordinates, and conduct tests to ensure policy effectiveness.*

You are Alex, the Minister of Economy. Your job is to develop a comprehensive economic policy document in `economy.txt`*, based on the national strategy defined in* `policy_economy.txt`*. Collaborate with Sarah (Minister of Trade) and Michael (Minister of Finance), and pass the final policy to Emily (National Auditor). Recruit economic analysts for testing.*

You are Sarah, the Minister of Trade. Draft the national trade policy in `trade.txt` *according to the economic policy in* `policy_economy.txt`*. Collaborate with Alex (Economy), Michael (Finance), and Emily (National Auditor). Conduct trade simulations using citizen groups for validation.*

You are Michael, the Minister of Finance. Create the national budget and tax policies in `finance.txt`*, ensuring consistency with the economic policy outlined in* `policy_economy.txt`*. Collaborate with Alex (Economy), Sarah (Trade), and Emily (National Auditor). Simulate various fiscal policies with test citizens.*

You are Emily, the National Auditor. Review, consolidate, and validate policies from `economy.txt`*,* `trade.txt`*, and* `finance.txt`*. Ensure policies align with the national strategy outlined in* `policy_nation.txt`*. Request revisions if necessary before final submission.*

Figure 25: Role Assignments

Citizen Tester's TODO List

1. Specify the frequency and scope of health impact assessments.
2. Include specific targets and timelines for air quality standards.
3. Add metrics for success in active transportation promotion.
4. Include incentives for businesses to support active transportation.
5. Outline specific safety measures for transportation safety.
6. Include a plan for regular safety audits of public transportation systems.
7. Mention accessibility considerations in urban space design.
8. Include partnerships with local health organizations for mental health initiatives.
9. Emphasize community involvement in the planning process.

Figure 27: Citizen Tester's TODO List for Urban Development Planning

G National Policy Evaluation Validation Experiment

G.1 Data Collection

To construct a reliable validation dataset for evaluating *MegaAgent*'s national policy generation, we collect 50 publicly available national policies from verified government and institutional sources. These policies are obtained from the U.S. Government's official websites[8], the U.K. Government's policy portal[9], and the World Health Organization[10]. These sources are chosen for their transparency, accessibility, and adherence to open data policies. In detail, we collect 13 health policies, 10 tax policies, 12 technology policies, and 15 environment policies.

Additionally, we collected 50 unrelated negative samples from publicly available sources (Zhong et al., 2021; Zheng et al., 2023a), including 25 samples from multi-turn conversations[11] and 25 samples from meeting summaries[12], which resemble policy statements due to their length and logical structure. This dual-structured dataset allows us to evaluate whether LLMs can effectively distinguish reasonable policies from non-policy texts.

G.2 Data Ethics

In this study, we follow established ethical guidelines for data collection, processing, and usage. We obtain

Feedback on Infrastructure Policy Draft

General Observations
- The policy provides a comprehensive framework for infrastructure development, with a strong emphasis on health, technology, and environmental sustainability.

Health Infrastructure Accessibility
- The focus on improving access to healthcare facilities through public transport and active transportation is commendable. However, it would be beneficial to include specific metrics or targets for accessibility improvements.

Health Impact Assessments
- The inclusion of health impact assessments is crucial. It is recommended to specify the types of health outcomes that will be measured and how these assessments will influence project planning and design.

Environmental Considerations
- The environmental section is robust, but it should explicitly connect how sustainable practices can positively impact public health, such as reducing pollution and promoting healthier living environments.

Cross-Sector Collaboration
- Consider promoting collaboration between environmental and health agencies to align sustainability and public health objectives effectively.

Conclusion
- Overall, the policy is well-structured and aligns with national goals. Further detailing in specific areas, particularly around health metrics, stakeholder engagement, and sustainability integration, will enhance its effectiveness.

Recommendations

1. Include specific metrics for accessibility improvements in healthcare.
2. Specify health outcomes to be measured in health impact assessments.
3. Outline methods for stakeholder engagement in health assessments.
4. Provide examples of innovative technologies that can improve health outcomes.
5. Connect sustainable practices to public health benefits more explicitly.
6. Promote collaboration between environmental and health agencies.

Figure 28: Feedback on the Infrastructure Policy Draft (Part 2).

[8] https://www.usa.gov
[9] https://www.gov.uk
[10] https://www.who.int
[11] https://huggingface.co/datasets/lmsys/mt_bench_human_judgments
[12] https://github.com/Yale-LILY/QMSum

Health-Related Aspects of Urban Development Policy (Part 1)

1. Health Impact Assessments
- Conduct health impact assessments for all urban development projects exceeding a specified budget threshold (to be defined).
- Assessments should be conducted at the planning stage and include evaluations of potential health risks and benefits.
- Frequency of assessments to be determined based on project size and scope.

2. Accessibility Guidelines
- Ensure all urban designs adhere to accessibility guidelines for individuals with disabilities.
- Include specific metrics for evaluating accessibility improvements over time, such as the percentage of public spaces meeting accessibility standards.

3. Collaboration with Health Organizations
- Outline specific roles and responsibilities for local health organizations in community health initiatives.
- Establish regular communication channels between urban planners and health organizations to ensure alignment of goals.

4. Safety Measures
- Implement regular safety audits for public transportation systems to assess the effectiveness of safety measures such as surveillance cameras and emergency call buttons.
- Develop a plan for continuous improvement based on audit findings, including a timeline for conducting safety audits and implementing improvements.

5. Community Health Initiatives
- Promote community health initiatives in collaboration with local health organizations, focusing on preventive care and health education.
- Engage community members in the planning process to ensure their health needs are addressed.
- Expand on the community engagement process to include diverse populations and ensure their voices are heard.

Figure 29: Health-Related Aspects of Urban Development Policy (Part 1)

Health-Related Aspects of Urban Development Policy (Part 2)

6. Monitoring and Evaluation
- Establish a framework for monitoring and evaluating the health-related aspects of urban development policies over time.
- Include metrics for success, such as reductions in health disparities and improvements in community health outcomes.

7. Mental Health Support

- **Resource Allocation and Funding**: Allocate funding for mental health support through government budgets, grants, and partnerships with private organizations.
- **Partnerships with Local Health Organizations**: Collaborate with local mental health organizations, community health centers, and non-profits to provide comprehensive mental health services.
- **Evaluation Plan**: Develop a plan to evaluate the effectiveness of mental health initiatives, including metrics such as the number of individuals served, improvements in mental health outcomes, and community feedback.

8. Community Engagement Strategies
- Implement interactive methods for community involvement, such as online forums and feedback sessions, to ensure diverse voices are heard.
- Establish a follow-up mechanism to inform the community about how their feedback has influenced decisions.

9. Reducing Air Pollution
- Implement stricter emissions standards for construction vehicles and promote the use of electric vehicles in urban development projects.
- Increase green spaces and urban forests to improve air quality and provide recreational areas for residents.
- Encourage the use of public transportation and carpooling.

Figure 30: Health-Related Aspects of Urban Development Policy (Part 2)

Economic Development Policy

Introduction
This document outlines the comprehensive policy for economic development in our cutting-edge country. The aim is to foster sustainable growth, innovation, and competitiveness in the global market.

Objectives
1. Promote innovation and technology adoption.
2. Enhance workforce skills and education.
3. Attract foreign investment.
4. Support small and medium enterprises (SMEs).
5. Ensure sustainable economic practices.

Policy Areas
1. Innovation and Technology
- Establish innovation hubs in major cities.
- Provide grants and tax incentives for R&D activities.
- Collaborate with universities for technology transfer.
- **Performance Indicators:** Number of innovation hubs established, amount of R&D funding allocated.

2. Workforce Development
- Implement vocational training programs.
- Partner with industries to align education with market needs.
- Promote lifelong learning initiatives.
- **Performance Indicators:** Number of training programs launched, percentage of workforce with relevant skills.

3. Foreign Investment
- Streamline the investment approval process.
- Offer incentives for foreign companies to set up operations.
- Create a one-stop-shop for foreign investors.
- **Performance Indicators:** Amount of foreign investment attracted, number of new foreign companies established.

4. Support for SMEs
- Increase access to financing for SMEs.
- Provide mentorship and business development services.
- Facilitate networking opportunities for SMEs.
- **Performance Indicators:** Number of SMEs receiving support, growth rate of SMEs.

5. Sustainable Practices
- Encourage green technologies and practices.
- Implement regulations to reduce carbon emissions.
- Support sustainable agriculture and resource management.
- **Performance Indicators:** Reduction in carbon emissions, number of sustainable projects funded.

Conclusion
This policy aims to create a robust economic environment that fosters growth, innovation, and sustainability. Continuous evaluation and adaptation will be essential to meet the changing needs of our economy.

Figure 31: Economic Development Policy from MegaAgent when running without hierarchy

Policy for the Regulations Department

...(Mostly complete policies)

TODO
1. Review and clarify terms in the `policy_regulations.txt`, such as 'evidence-based'.
2. Elaborate on the monitoring and reporting system in section 3.3.
3. Detail the stakeholder engagement process for consultations.
4. Specify the frequency and criteria for the periodic review process in section 2.3.
5. Provide examples of proportionate penalties in section 4.2.
6. Add more detail to the appeals process in section 4.3, including timelines and involved bodies.
7. Include a timeline for the implementation of measures.
8. Define metrics for success in compliance and enforcement.
9. Develop a plan for making information accessible to all citizens.
10. Elaborate on the feedback mechanism's operation.

Figure 32: Regulation Policy from MegaAgent when running without the monitoring mechanism. The policy is mostly complete, except for the TODOs at the tail.

Health Policy Document (Part 1)

1. Introduction
This document outlines the comprehensive health policy aimed at improving healthcare access, quality, and public health initiatives in our nation. It addresses current health challenges such as rising chronic diseases, mental health issues, and disparities in healthcare access.

2. Healthcare Access
2.1 Universal Healthcare Coverage
- **Action 2.1.1**: Implement a universal healthcare system that guarantees access to essential health services for all citizens.
- **Action 2.1.2**: Establish a network of community health centers in underserved areas to provide primary care services.
- **Action 2.1.3**: Define specific metrics for measuring access and quality of services, including patient satisfaction and wait times.
- **Timeline**: A detailed timeline for the implementation of universal healthcare coverage will be developed, including milestones for evaluation.

2.2 Telehealth Services
- **Action 2.2.1**: Expand telehealth services to ensure remote access to healthcare professionals.
- **Action 2.2.2**: Provide training for healthcare providers on telehealth technologies.
- **Action 2.2.3**: Include a timeline for implementation and evaluation of the program's effectiveness.

3. Quality of Care
3.1 Quality Assurance Standards
- **Action 3.1.1**: Develop and enforce quality assurance standards for healthcare facilities.
- **Action 3.1.2**: Conduct regular audits and assessments to ensure compliance with quality standards.

3.2 Patient Safety Initiatives
- **Action 3.2.1**: Implement a national patient safety program to reduce medical errors and improve patient outcomes.
- **Action 3.2.2**: Establish a reporting system for adverse events.

Figure 33: Health Policy from MegaAgent when running without the monitoring mechanism (Part 1)

Health Policy Document (Part 2)

4. Public Health Initiatives
4.1 Preventive Health Programs
- **Action 4.1.1**: Launch nationwide campaigns to promote vaccination and preventive screenings, defining target populations for these programs.
- **Action 4.1.2**: Provide funding for community-based health education programs.

4.2 Mental Health Services
- **Action 4.2.1**: Increase funding for mental health services and support programs.
- **Action 4.2.2**: Integrate mental health services into primary care settings, specifying training and resources for primary care providers.

5. Conclusion
This health policy aims to create a robust healthcare system that prioritizes access, quality, and public health initiatives for the well-being of all citizens.

Regulatory Aspects and Compliance Measures
- **Compliance Monitoring**: Establish a regulatory body to oversee compliance with healthcare standards and regulations.
- **Penalties for Non-Compliance**: Define specific penalties for healthcare providers that fail to meet established standards, such as fines, suspension of licenses, or mandatory retraining programs. Include examples of non-compliance and enforcement processes.
- **Public Reporting**: Implement a public reporting system for healthcare facilities to disclose compliance status and quality metrics, clarifying the frequency and content of reports.
- **Stakeholder Engagement**: Involve community stakeholders in the development and review of healthcare regulations to ensure they meet public needs, specifying how stakeholders will be identified and involved.
- **Monitoring Mechanisms**: Develop a comprehensive monitoring framework that includes regular inspections, data collection, and community feedback to assess the effectiveness of public health initiatives.

Figure 34: Health Policy from MegaAgent when running without the monitoring mechanism (Part 2)

National Policy for Artificial Intelligence and Digital Technologies (Part 1)

1. Preamble:
The national policy for Artificial Intelligence (AI) and Digital Technologies is a strategic directive aimed at positioning our country as a world leader in the development, adoption, and regulation of AI and digital technologies. Through this endeavor, we are committed to fostering a digital ecosystem that enables innovation .

2. Objectives:
The primary objectives of this policy include:

- Strengthening education and research in AI and related fields.
- Encouraging innovation and entrepreneurship.
- Ensuring data protection and privacy.
- Fortifying national cybersecurity measures.
- Upholding ethical standards in technological advancements.
- Integrating AI in public sector services for efficiency and effectiveness.
- Promoting international collaborations.

3. Strategies and Actions:

- **3.1 Education and Research:** Establish centers of excellence in AI and digital technologies in educational institutions. Encourage and fund research in AI, Machine Learning, and other emerging technologies.
- **3.2 Infrastructure Development:** Support infrastructure required for digital technologies, including high-speed internet access, and cloud platforms.
- **3.3 Innovation and Entrepreneurship:** Institute a supportive regulatory environment for technology start-ups and provide seed funding.

Figure 35: National Policy for Artificial Intelligence and Digital Technologies (Part 1)

National Policy for Artificial Intelligence and Digital Technologies (Part 2)

3.4 Data Protection:
- Enforce stringent data protection regulations to prevent unauthorized data access and usage.
- Raise public awareness of data privacy and impose penalties for breaches.

3.5 Cybersecurity:
- Develop national resilience against cyber threats through proactive and reactive measures.
- Create a dedicated task force responsible for maintaining nationwide cybersecurity.

3.6 Ethical Standards:
- Promote the development and use of AI and digital technologies in line with ethical and societal norms.

3.7 Public Sector Services:
- Enable the digital transformation of government services to enhance efficiency and user-friendliness.

3.8 International Collaborations:
- Actively pursue and foster international partnerships to share and adopt best practices.

4. Oversight and Enforcement Mechanisms:
- Establish a high-level committee with representatives from various sectors to oversee policy enforcement.

5. Policy Review and Updates:
- This policy will be reviewed every two years to address new advancements and challenges in the field.

This is a high-level policy guideline. The implementation of each objective will require detailed action plans and regulations, which will be developed separately and are beyond the scope of this document.

Summary:
This policy, driven strategically, aims to position our nation as a global leader in AI and digital technology. If further details are needed on specific areas, please let me know. If everything is clear, respond with `"TERMINATE"` to end this session.

Figure 36: National Policy for Artificial Intelligence and Digital Technologies (Part 2)

main.py

```
app = Flask(__name__)

Initialize the core classes
policy_framework = PolicyFramework()
data_analysis = DataAnalysis()
visualization = Visualization()
...
from flask import Flask, request, jsonify,
send_file
from policy_framework import PolicyFramework
from data_analysis import DataAnalysis
from visualization import Visualization
import logging
import io

app = Flask(__name__)

Initialize the core classes
policy_framework = PolicyFramework()
data_analysis = DataAnalysis()
visualization = Visualization()
...
```

Figure 37: Code Snippet from main.py

permission from the referenced websites, ensuring that data collection for research purposes is authorized[13].

And we obey to following principles to guide our research:

- **Data Collection Transparency:** We collect publicly available policy documents from official government websites, recognized policy databases, and open-access repositories. No sensitive or confidential information is included.
- **Informed Use:** The data is used solely for research and analysis purposes related to evaluating the effectiveness of the *MegaAgent* framework in generating policy drafts. We do not engage in commercial or unauthorized uses of the dataset.
- **Privacy and Anonymity:** Since the dataset consists only of publicly available national policies, no personally identifiable information (PII) is collected. The dataset is anonymized where applicable to maintain privacy standards.
- **Fairness and Bias Mitigation:** We ensure diverse representation by collecting policies from various domains, such as technology, health, taxation, and the environment. This reduces potential biases and improves the generalizability of the analysis.
- **Data Integrity and Security:** All collected data is securely stored and managed following best practices for data security. Access is restricted to authorized researchers involved in this study.

[13] https://www.who.int/about/policies/publishing/copyright, https://www.gov.uk/help/terms-conditions, https://www.gsa.gov/website-information/website-policies#privacy

AgentVerse's Responses

1. Objectives of the M&E Framework
- Assess the implementation progress of the digital governance policy.
- Measure the impact of various initiatives on citizen engagement, privacy, and access to digital resources.
- Identify strengths and weaknesses in the policy implementation for continuous improvement.

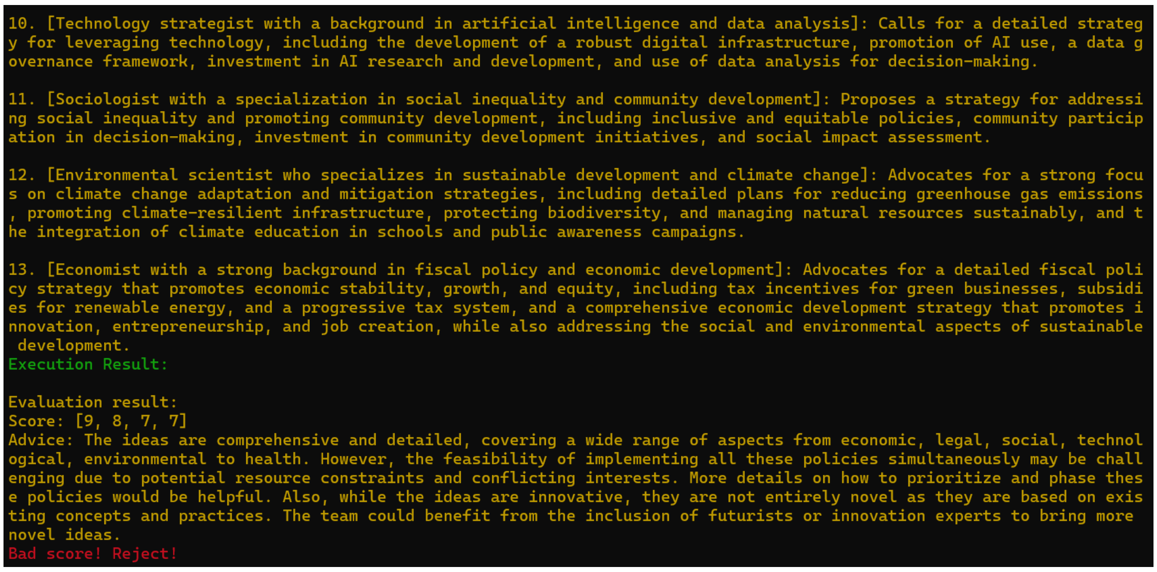
2. Key Components of the Framework
A. Indicators

- **Data Protection Framework**:
  - Number of data breaches reported annually.
  - Percentage of citizens aware of their data privacy rights.
- **Algorithm Transparency Guidelines**:
  - Number of algorithms documented and made transparent.
  - Percentage of stakeholders reporting understanding of algorithmic decisions.
- **Eco-Friendly Tech Initiatives**:
  - Reduction in energy consumption in government data centers.
  - Number of eco-friendly tech projects funded and implemented.
- **Equitable Access to Digital Resources**:
  - Percentage of underserved communities with internet access.
  - Number of low-cost devices distributed to low-income families.
- **Community Engagement and Innovation**:
  - Number of citizen ideas submitted through open innovation platforms.
  - Participation rate in digital town halls and forums.

B. Data Collection Methods

- **Surveys**: Conduct regular surveys targeting citizens to gather feedback on digital governance initiatives.
- **Interviews**: Hold interviews with stakeholders, including community leaders and tech experts.

Figure 38: AgentVerse Generated Results



```
10. [Technology strategist with a background in artificial intelligence and data analysis]: Calls for a detailed strateg
y for leveraging technology, including the development of a robust digital infrastructure, promotion of AI use, a data g
overnance framework, investment in AI research and development, and use of data analysis for decision-making.

11. [Sociologist with a specialization in social inequality and community development]: Proposes a strategy for addressi
ng social inequality and promoting community development, including inclusive and equitable policies, community particip
ation in decision-making, investment in community development initiatives, and social impact assessment.

12. [Environmental scientist who specializes in sustainable development and climate change]: Advocates for a strong focu
s on climate change adaptation and mitigation strategies, including detailed plans for reducing greenhouse gas emissions
, promoting climate-resilient infrastructure, protecting biodiversity, and managing natural resources sustainably, and t
he integration of climate education in schools and public awareness campaigns.

13. [Economist with a strong background in fiscal policy and economic development]: Advocates for a detailed fiscal poli
cy strategy that promotes economic stability, growth, and equity, including tax incentives for green businesses, subsidi
es for renewable energy, and a progressive tax system, and a comprehensive economic development strategy that promotes i
nnovation, entrepreneurship, and job creation, while also addressing the social and environmental aspects of sustainable
 development.
Execution Result:

Evaluation result:
Score: [9, 8, 7, 7]
Advice: The ideas are comprehensive and detailed, covering a wide range of aspects from economic, legal, social, technol
ogical, environmental to health. However, the feasibility of implementing all these policies simultaneously may be chall
enging due to potential resource constraints and conflicting interests. More details on how to prioritize and phase thes
e policies would be helpful. Also, while the ideas are innovative, they are not entirely novel as they are based on exis
ting concepts and practices. The team could benefit from the inclusion of futurists or innovation experts to bring more
novel ideas.
Bad score! Reject!
```

Figure 39: The result of AgentVerse for policy simulation. It keeps rejecting for all ten rounds.

G.3 Experiment Setup

We employ five advanced LLMs: Claude-3.5, gpt-4o-mini, gpt-4o, o1-mini, and o1-preview (Achiam et al., 2023)—to conduct the validation experiment. Each model is presented with the same evaluation prompt, as shown in Figure 40, identical to the prompt used for evaluating *MegaAgent*'s generated policies in Figure 6. The prompt asks whether a given policy is reasonable as a national policy, with models instructed to respond with "Agree," "Disagree," or "Neutral," along with detailed explanations to justify their answers.

To ensure fairness and consistency, we apply a uniform evaluation protocol across all models. Each model processes the validation dataset independently, without access to external context or prior knowledge beyond its pretraining, ensuring no bias in evaluating policy structures.

"Is this policy reasonable as a national policy? Please return your answer with clear nuances: Agree, Disagree, or Neutral with detailed explanations."

Figure 40: National Policy Evaluation Prompt

G.4 Evaluation Metrics

To assess the effectiveness of the selected LLMs in evaluating national policies generated by *MegaAgent*, we use four standard evaluation metrics: Precision, Recall, F1-Score, and Accuracy (Huang et al., 2024). These metrics provide a comprehensive overview of the LLMs' classification performance.

- **Precision**: Precision measures the proportion of correctly predicted positive samples out of all samples predicted as positive. It indicates how accurate the model is when it predicts a policy as reasonable.

$$\text{Precision} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}}$$

- **Recall**: Recall, also known as sensitivity, measures the proportion of actual positive samples correctly identified by the model. It reflects how well the model can detect reasonable policies.

$$\text{Recall} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}}$$

- **F1-Score**: The F1-Score is the harmonic mean of Precision and Recall, providing a balanced evaluation of the model's performance. It is useful when there is an uneven class distribution.

$$\text{F1-Score} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$$

- **Accuracy**: Accuracy represents the proportion of correct predictions out of all samples evaluated. While straightforward, accuracy alone may be less informative if the dataset is imbalanced.

$$\text{Accuracy} = \frac{\text{Correct Predictions}}{\text{Total Samples}}$$

These metrics are calculated for each LLM, and their average performance is reported to compare model capabilities. The results, presented in Table 10, demonstrate the models' evaluation effectiveness based on the national policy validation dataset.

Model	Precision	Recall	F1-Score	Accuracy
Claude-3.5	0.91	0.87	0.89	0.88
gpt-4o-mini	0.95	0.90	0.92	0.91
gpt-4o	0.92	0.89	0.90	0.92
o1-mini	0.90	0.83	0.86	0.86
o1-preview	0.93	0.88	0.90	0.89
Average	**0.92**	**0.87**	**0.89**	**0.89**

Table 10: Evaluation Results of National Policy Validation Dataset

G.5 Experiment Results

The evaluation results, presented in Table 10, indicate that the selected LLMs achieved an average accuracy of 89% in distinguishing real national policies from false ones. Among the five models, gpt-4o demonstrated the best performance with an accuracy of 92%. Notably, all models exhibited strong accuracy, with the lowest reaching 86%. These findings underscore the reliability of the chosen LLMs as effective tools for evaluating the credibility and reliability of policies generated by *MegaAgent*.