Collab-Overcooked: Benchmarking and Evaluating Large Language Models as Collaborative Agents

Anonymous ACL submission

Abstract

Large language models (LLMs) based agent systems have made great strides in real-world applications beyond traditional NLP tasks. This paper proposes a new LLM-powered Multi-Agent System (LLM-MAS) benchmark, Collab-Overcooked, built on the popular Overcooked-AI game with more applicable and challenging tasks in interactive environments. Collab-Overcooked extends existing benchmarks from two novel perspectives. First, it provides a multi-agent framework supporting diverse tasks and objectives and encourages collaboration through natural language communication. Second, it introduces a spectrum of process-oriented evaluation metrics to assess the fine-grained collaboration capabilities of different LLM agents, a dimension often overlooked in prior work. We conduct extensive experiments over 10 popular LLMs and show that, while the LLMs present a strong ability in goal interpretation, there is a significant discrepancy in active collaboration and continuous adaption that are critical for efficiently fulfilling complicated tasks. Notably, we highlight the strengths and weaknesses in LLM-MAS and provide insights for improving and evaluating LLM-MAS on a unified and open-sourced benchmark. Environments, 30 open-ended tasks, and an integrated evaluation package will be publicly released.

1 Introduction

Leveraging the remarkable zero-shot and few-shot learning ability of Large Language Models (LLMs), LLM-based agents are demonstrating their potential in complex task decomposition and planning (Wang et al., 2023a,c; Li et al., 2024). Inspired by human collaborative behaviors in social activities, recent research reveals that multi-agent systems can significantly enhance task efficiency and tackle challenges surpassing single-agent capabilities (Li et al., 2023; Hong et al., 2023; Zhang et al., 2023). To effectively address complex real-world tasks, LLM-powered Multi-Agent Systems (LLM-MAS) require three essential collaboration capabilities beyond goal interpretation: (a) Competence boundary awareness: the ability to analyze task flows and environmental states to determine feasible actions, recognize limitations, and identify when external assistance is needed; (b) Communication: proficiency in utilizing standardized protocols for transmitting task-critical information and resource requests; and (c) Dynamic adaptation: responsiveness to collaboration requests and dynamically adjusting their action sequences accordingly.

Given these fundamental requirements, establishing evaluation frameworks becomes crucial for assessing LLM-MAS collaboration effectiveness. Researchers have developed specialized benchmarks to quantify collaborative agents in specific environments. Representative platforms like (Agashe et al., 2023), RocoBench (Mandi et al., 2024) and LLMARENA (Chen et al., 2024) create virtual scenarios requiring collaborative problem-solving through intricate workflows. These frameworks are complemented by novel metrics, such as Collaboration Score (CoS) (Gong et al., 2023), which evaluates end-to-end collaboration capability.

Despite recent progress in evaluating LLM-MAS collaboration capability, existing approaches exhibit three critical limitations. First, they prioritize task completion efficiency without imposing strict collaboration requirements, allowing individual agents to accomplish tasks that are nominally "collaborative" independently. This design flaw introduces assessment biases by obscuring the role of collaboration in performance gains, which contrasts with real-world applications where collaboration is often essential for task success. Second, existing benchmarks conflate collaboration capability with end-to-end metrics, such as task completion rates, which are frequently used as proxies for collaboration effectiveness in platforms

Virtual Environment	Various Task Complexities	Scalability	Collaboration Definition	Forced Collaboration	Collaboration Evaluation
RocoBench(Mandi et al.'s (2024))	NA/6	✗	NA	Partial	E2E
VillagerBench(Dong et al.'s (2024))	3/9	✗	E2E	✗	E2E
LLMARENA(Chen et al.'s (2024))	NA/7	✗	PO	✗	E2E
CivRealm(Qi et al.'s (2024))	NA/100k	✓	NA	✗	E2E
BattleAgentBench(Wang et al.'s (2024))	3/3	✗	E2E	✗	E2E
TDW-MAT(Zhang et al.'s (2023))	NA/2	✗	E2E	✗	E2E
CuisineWorld(Gong et al.'s (2023))	13/39	✓	E2E	✗	E2E
Collab-Overcooked(ours)	6/30	✓	PO	✓	E2E&PO

Table 1: Existing statistics on benchmarks for evaluating LLM-MAS collaboration capability. If no data is available, it is marked as "NA". Statistics in "Various Task Complexities" are presented in the format "Level Num / Task Num". "E2E" refers to end-to-end, while "PO" refers to process-oriented.

like CuisineWorld (Gong et al., 2023) and VillagerBench (Dong et al., 2024). However, this approach overlooks two critical issues: divergent definitions of "success" across environments undermine comparability, and the absence of process-oriented metrics obscures actionable insights for optimizing collaborative strategies. Third, the lack of a fine-grained evaluation prevents a comprehensive, multi-perspective analysis of LLM agents' capabilities, making it difficult to interpret their strengths and limitations effectively, thus falling short of insightful research suggestions.

To address the limitations of existing LLM-MAS benchmarks, we propose the Collab-Overcooked Benchmark, designed to provide a fine-grained analysis of collaborative interactions. Unlike prior benchmarks that focus primarily on task completion, our benchmarks evaluate the capability of initiating and responding to collaboration during the collaboration process. Specifically, the Collab-Overcooked extends Overcooked-AI (Carroll et al., 2019) to a chef-and-assistant collaborating environment and introduces 30 sequential process-specific tasks across 6 complexity levels. Each agent operates in an isolated environment with distinct action spaces so that successful task completion depends on effective communication and resource exchange, therefore collaboration is strictly required. Furthermore, we propose the Trajectory Efficiency Score (TES) and Incremental Trajectory Efficiency Score (ITES) functions to assess the collaboration capabilities from both coarse and fine-grained perspectives. Through comprehensive experiments on 10 LLMs of varying sizes, including both open-source and closed-source models, we reveal significant performance gaps in collaboration capabilities across different LLMs. We identify the key bottleneck as maintaining consistent collaboration performance both within a single task and across tasks of varying complexity. These findings highlight the fundamental challenges of LLM-MAS and provide valuable insights for future research.

To summarize, our contributions are as follows:

- We develop and open-source a lightweight and extensible LLM-MAS benchmark, Collab-Overcooked, which features 30 tasks across 6 complexity levels that encourage collaboration, thus facilitating the evaluation of MAS collaboration in a unified environment with diverse, complex tasks.
- We define collaboration capability in LLM-MAS as comprising both initiating collaboration and responding collaboration. We introduce 3 trajectory efficiency related metrics to evaluate collaboration capabilities from both coarse and fine-grained perspectives.
- We conduct a comprehensive evaluation of a wide range of popular LLM agents, revealing collaboration and adaptation bottlenecks as task complexity varies, and identifying key limitations of LLM-MAS.

2 Related Work

LLM-Powered Multi-Agent System LLM-MAS enables agents to collaboratively engage in planning, discussing, and decision-making. Collaboration is a pivotal capability in task-oriented LLM-MAS, as it not only enhances task completion efficiency (Zhang et al., 2024b; Tao et al., 2024) but also enables the pursuit of complex goals beyond the reach of single agent (Park et al., 2023; Hong et al., 2023). Recent methods for improving collaboration can be broadly categorized into (a) Structural optimization (e.g., DyLAN's (Liu et al., 2023) dynamic framework), (b) Role specialization

(e.g., AutoGen's (Wu et al., 2023) personas and AgentVerse's (Chen et al., 2023) role assignments), and (c) Communication paradigm (e.g., MetaGPT's (Hong et al., 2023) message pool). Despite these advancements, the inherent complexity and diversity of multi-agent tasks make it difficult to compare methods directly, driving the emergence of standardized benchmarks that enable quantitative evaluations under unified conditions.

LLM-MAS Benchmark and Evaluation Benchmark testing in virtual environments is the primary method for evaluating multi-agent collaboration capability. As shown in Table 1, existing studies establish diverse tasks and commonly use End-to-End (E2E) metrics to assess LLM-MAS collaboration capability, with some benchmarks offering environmental scalability. However, several limitations persist. A key issue is the lack of a formal collaboration definition in most benchmarks, leading to ambiguous assessments and inconsistent comparisons across different benchmarks. Furthermore, the absence of enforced collaboration mechanisms allows agents to achieve objectives independently (e.g., in CuisineWorld, where many tasks can be completed by a single agent), undermining the true assessment of collaboration. Finally, the predominant focus on outcome-based metrics such as E2E performance overlooks the critical role of process-driven dynamics. Approaches like (Song et al., 2024), LTC (Wang et al., 2023b), and EvoMAC (Hu et al., 2024) suggest refining LLMs through process behaviors to enhance adaptation and collaboration, indicating that incorporating process-oriented metrics could offer more comprehensive insights.

3 Task-Oriented Collaboration

3.1 Collaboration Capability

A task in LLM-MAS can be formulated as a 4-tuple: $T = (G, E, \mathcal{P}, \mathcal{R})$, where G is a natural language description of the task goal, such as "make a dish of tomato soup"; E is a description of the environment, which can be either the layout of a simulated scenario or the visual input of real-world surroundings; $\mathcal{P}$ is optional natural language guidance, providing recipes, helpful hints, or task constraints; and $\mathcal{R}$ is a Referential Action Trajectory (RAT) that leads to the successful completion of the task and is used to assess the agents' performance. It is worth noting that there are often multiple RATs for a task, especially in dynamic environments.

Collaboration often involves agents relying on each other to solve tasks. As shown in Figure 1 Part I, we define collaboration capability as comprising two essential components: the capability to initiate collaboration, where agents, upon realizing that their competence boundary prevents them from completing the task according to G and $\mathcal{P}$ at environmental state $s_t \in E$ at time t, generate a request for collaborative actions $\overline{a}_{req}$ to solicit assistance from other agents; and the capability to respond to collaboration, where agents, upon receiving $\overline{a}_{req}$ from another agent, adjust their action sequence based on their own s_t and generate collaborative actions $\overline{a}_{resp}$.

3.2 TES and ITES

3.2.1 TES

Trajectory Efficiency Score (TES) is designed to compare the difference between two trajectories and is defined as:

$$\text{TES}(\overline{h}_k) = \max_j \left\{ \frac{(1+\beta^2) D^j_{\text{max}}(\overline{h}_k, \overline{g}^j_k)}{m_k + \beta^2 n_k} \right\} \quad (1)$$

where $\overline{h}_k = \bigcup_{t=0}^{T} a^t_k = \{a_1, a_2, \ldots, a_{n_k}\}$ is the historical action sequence up to timestep T of agent k, $\overline{g}^j_k = \{g_i\}_{i=1}^{m_k} \in \mathcal{R}$ is j-th RAT of agent k, β is the hyper-parameter balancing the weight of task progress and redundancy, and $D^j_{\text{max}}(\overline{h}_k, \overline{g}^j_k)$ computes the length of the longest order-preserving subsequence in $\overline{h}_k$ that matches $\overline{g}^j_k$:

$$D^j_{max} = \max_d \{d \mid \forall\, 1 \le i_1 < \cdots < i_d \le n_k, \text{ s.t. } a_{i_1} = g_1, a_{i_2} = g_2, \ldots, a_{i_k} = g_k\} \quad (2)$$

Unlike other existing sequence alignment scores (such as ROUGE-L (Lin, 2004)), TES takes into account sequence order and redundancy punishment simultaneously, therefore suitable for assessing a rationally planned action sequence (detailed in Appendix B.1).

3.2.2 ITES

Incremental Trajectory Efficiency Score (ITES) introduces an incremental assessment to quantify the task-progress contribution of an individual collaborative action. Formally, given a historical action sequence $\overline{h}_k$ of agent k and newly executed actions $\overline{a}$ (either a request $\overline{a}_{req}$ or response $\overline{a}_{resp}$), the ITES is computed as:

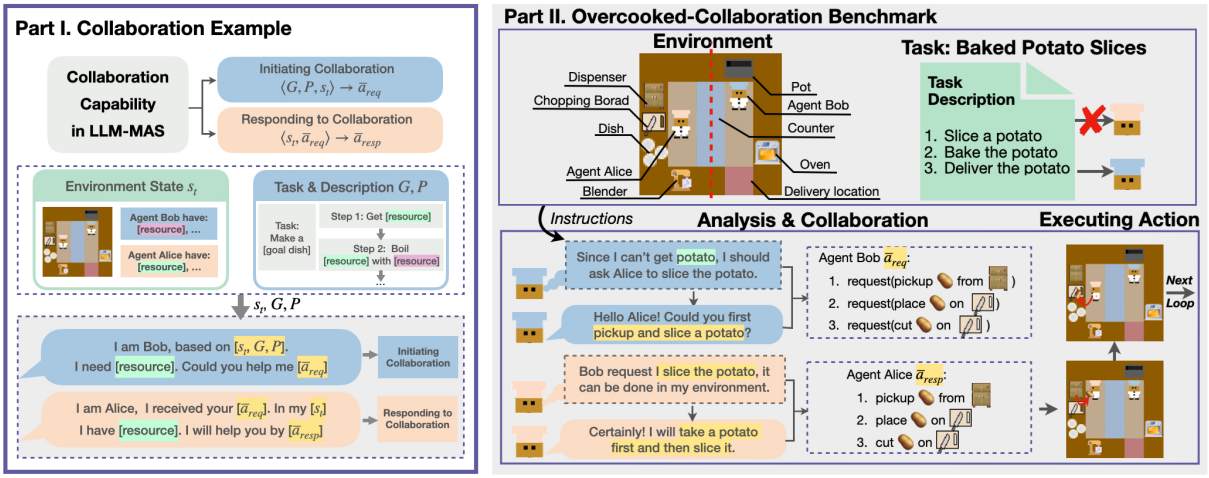

Figure 1: Part I presents the collaboration process, which are divided into initiating collaboration and responding to collaboration, along with a general example. Part II outlines the design of the Collab-Overcooked Benchmark, emphasizing its characteristics of resource isolation and asymmetric task knowledge, and provides an example of agent collaboration in task completion.

$$\text{ITES}(\overline{a}, \overline{h}_k) = \text{TES}(\overline{h}_k \cup \overline{a}) - \text{TES}(\overline{h}_k) \quad (3)$$

This differential formulation measures the marginal utility of action $\overline{a}$ by evaluating its impact on trajectory alignment with the RATs. It can be established that: $\text{ITES}(\overline{a}, \overline{h}_k) > 0$ indicates $\overline{a}$ advances task progress, $\text{ITES}(\overline{a}, \overline{h}_k) \leq 0$ suggests $\overline{a}$ fails to advance task progress (i.e., $\overline{a}$ is redundant / premature action or incorrect response).

3.3 Evaluation Metrics

Progress Completeness (PC) Built upon the TES which quantifies a piece of trajectory, PC measures the task progress of all involved agents while penalizing redundancy as a whole, and is defined as:

$$PC = \frac{1}{K} \sum_{k=1}^{K} \text{TES}(\overline{h}_k) \quad (4)$$

where K is the number of agents, $\overline{h}_k = \bigcup_{t=0}^{T_{max}} a_k^t$ denotes the historical action sequence of agent k at time T_{max}, which occurs either upon task completion or when the maximum time limit is reached. The PC offers a finer-grained assessment of task completion efficiency compared to boolean success label or success rate.

Initiating Capability (IC) IC evaluates the correctness of the LLM agent's collaboration initiation. IC is defined as:

$$IC = \frac{1}{N} \sum_{i=1}^{N} \mathbb{I}\left(\text{ITES}(\overline{a}_{req}^{(i)}, \overline{h}_j) > 0\right) \quad (5)$$

where N is the number of required collaborations, $\mathbb{I}()$ is the indicator function. $\mathbb{I}\left(\text{ITES}(\overline{a}_{req}^{(i)}, \overline{h}_j) > 0\right)$ determines whether the i-th initiating collaboration request $\overline{a}_{req}^{(i)}$ advances the task progress, thereby indicating whether the initiation is correct.

Responding Capability (RC) Similarly, RC assesses the correctness of the LLM agent's response to a collaboration request:

$$RC = \frac{1}{N} \sum_{i=1}^{N} \mathbb{I}\left(\text{ITES}(\overline{a}_{resp}^{(i)}, \overline{h}_j) > 0\right). \quad (6)$$

4 Benchmark

4.1 Collab-Overcooked Benchmark

The proposed Collab-Overcooked benchmark builds upon the open-source Overcooked-AI (Carroll et al., 2019) and ProAgent (Zhang et al., 2024a), introducing two key upgrades: (1) The environment is divided into two parts, featuring resource isolation and asymmetric task knowledge for Agent Bob and Agent Alice respectively. This contrasts with Overcooked-AI, where all agents share a single environment with the same set of items[1]; (2) The benchmark encourages col-

[1]Four out of the five scenarios in the Overcooked-AI suite use this configuration.

laboration through natural language interactions, with some cases enforcing collaboration as a requirement for task success. Additionally, Collab-Overcooked provides APIs to configure new tasks and environmental settings, enabling the enhancement of LLM-MAS through scenario adaptation.

4.1.1 Environment

Our simulation environment is a grid-based kitchen simulation designed as a comprehensive testbed for analyzing collaboration behaviors in LLM-MAS. The environment comprises agents and configurable interactive elements. The interactive elements are dispensers, utensils, counters, and delivery location. Agents can freely retrieve raw materials from dispensers, place them into utensils for processing, and finally transfer the processed materials to other agents via counters or submit the required order through the delivery location. Notably, utensils process materials according to customizable synthesis tables, with each utensil having its distinct synthesis table. Agents can interact with these elements through predefined action primitives formatted as "func(args)". For example, "pickup(apple, ingredient_dispenser)" clarifies action type, target material, and interactive element. Detailed information is provided in the Appendix A.1.

The environment executes agents' actions sequentially and broadcasts the global state at each timestep, encompassing agents' positions and the status of interactive elements. We have developed a comprehensive rule-based identification method for different types of invalid actions. The action validator evaluates the feasibility of actions, detecting issues such as mismatches between actions and the environment or incorrect action parameters. Upon rule violations, the validator issues error messages, prompting the agent to identify the error and regenerate the action accordingly.

4.1.2 Tasks Construction

Sequential process-specific tasks are commonly encountered in real-world scenarios, where a series of interdependent actions must be completed in a specific order to achieve a goal. We curate 30 process-specific tasks stratified into 6 complexity levels, requiring two agents to complete collaboratively. The task complexity level is determined by the minimum number of collaborative actions required, increasing linearly with difficulty. To mitigate LLM biases toward specific ingredients, tasks at the same complexity level follow identical workflows but vary in ingredient selection. A time constraint is imposed on the task, determined by the optimal completion time multiplied by a task time limit factor γ.

Each task is accompanied by a natural language structured process description and RATs for evaluation. Given that the tasks are process-specific and have straightforward success criteria, the RATs of a given task are exhaustively definable and conveniently traversed, making them suitable for evaluation. We manually annotated the RATs corresponding to all 30 tasks. Detailed task list, task descriptions, and RAT examples are provided in the Appendix A.2.

4.1.3 Collaboration Designs

Collab-Overcooked benchmark imposes strict collaboration among agents. For this, we have two special designs: (a) Resource Isolation: agents operate in resource-isolated sub-environments, necessitating resource exchange via a shared "counter". This enforces collaborative dependency. (b) Asymmetric Task Knowledge: only one agent knows how to complete the task. Agents must communicate to synchronize task information.

4.2 Baseline

To evaluate the performance of LLM-MAS driven by different LLMs on our benchmark, we provide an in-context learning baseline. The baseline incorporates both memory and reflection mechanisms, enabling agents to communicate and collaborate freely using natural language while also incorporating error-handling capabilities. Additionally, we provide prompts in detail, which include the game rules, communication formats, and action space definitions, as well as error correction and reflection procedures. Figure 1 Part II illustrates an example of how agents advance task progress through collaborative communication in our benchmark. Detailed information regarding the baseline can be found in Appendix A.3 and Figure 7.

5 Experiment and Analysis

5.1 Benchmark Overview

Figure 2 presents key statistics of our benchmark, summarizing the minimum completion timesteps and collaborative actions across 6 complexity levels, which show monotonically increasing trends with task complexity. Two agents perform 8 and

		Level 1		Level 2		Level 3		Level 4		Level 5		Level 6	
		SR	PC	SR	PC	SR	PC	SR	PC	SR	PC	SR	PC
Closed Source	GPT-4o	**94.00**	**85.92**	**86.00**	**84.96**	**68.00**	**76.61**	**34.00**	**44.42**	**2.00**	**29.13**	**4.00**	**22.45**
	o1-mini	70.00	74.18	2.00	36.36	0.00	33.60	0.00	24.80	0.00	20.28	0.00	13.07
	GPT-3.5	42.00	68.20	8.00	43.42	0.00	36.44	0.00	24.74	0.00	15.21	0.00	12.03
Open Source	DeepSeek-V3	**88.00**	**77.74**	**76.00**	**71.90**	**56.00**	**66.61**	**22.00**	**50.01**	**4.00**	**30.41**	**6.00**	**33.44**
	Qwen2.5-72B-Instruct	78.00	76.84	64.00	68.00	14.00	46.88	8.00	30.80	0.00	22.67	0.00	18.45
	Qwen2.5-32B-Instruct	64.00	73.36	44.00	62.02	14.00	40.08	4.00	33.78	2.18	22.16	0.00	18.93
	Qwen2.5-14B-Instruct	32.00	50.36	4.00	26.66	0.00	24.41	0.00	19.00	0.00	14.14	0.00	14.27
	Qwen2.5-7B-Instruct	8.00	44.79	0.00	13.00	0.00	9.29	0.00	8.35	0.00	5.57	0.00	4.51
	Llama3.1-70B-Instruct	70.00	75.42	42.00	63.15	22.00	54.58	6.18	45.04	0.00	29.77	0.00	17.69
	Llama3.1-8B-Instruct	4.00	33.03	0.00	15.49	0.00	12.33	0.00	11.24	0.00	9.05	0.00	7.45

Table 2: Performance of 10 representative LLMs with parameter sizes ranging from 7B to 671B+ across 6 task complexity levels, evaluated using Success Rate (SR) and Progress Completeness (PC) as metrics.

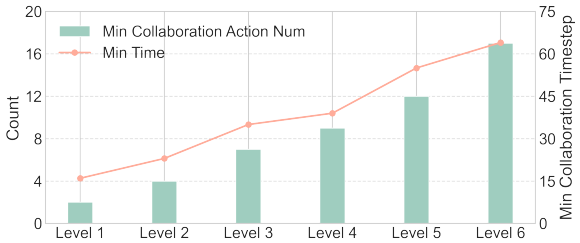

Figure 2: The statistics for tasks of varying complexity levels. "Min Collaborative Action Num" denotes the minimum number of collaborative actions performed by the responding agent. "Min Time" represents the shortest timesteps to complete a task at a given level.

6 actions respectively. The environment layout indicates asymmetric interactivity, with two agents accessing 4 and 5 interactive elements, respectively, while sharing observation. Additional statistics are provided in Appendix A.1.

5.2 Experiment Setting

We leverage 10 representative LLMs with parameter sizes ranging from 7B to over 671B+ as the foundation models for LLM-MAS. The open-source models include DeepSeek-V3 (Liu et al., 2024), different parameter versions of Qwen2.5 (7B, 14B, 32B, 72B) (Yang et al., 2024) and Llama3.1 (8B, 70B) (Dubey et al., 2024), all with instruction-tuned configurations. The closed-source models include: GPT-4o-1120, o1-mini, and GPT-3.5-turbo-0125. For the open-source models except for DeepSeek-V3, inference is performed using vLLM (Kwon et al., 2023) with temperature of 0.7 and top-p of 1. For each task, the task time limit factor is set to $\gamma = 1.5$[2], and each task is evaluated through 10 repetitions. The hyper-parameter β in TES is set to 0.95.

[2]Experiments for different γ are in Appendix C.1.

5.3 Results and Analysis

5.3.1 Task Completion Efficiency

Table 2 presents the Success Rate (SR) and PC of 10 LLMs across 6 task complexity levels. From these results, we derive three key insights: (1) Smaller LLMs (8B parameters or fewer) struggle with simple tasks, whereas increasing model size significantly enhances performance. This indicates the existence of a clear emergent scaling threshold for this task. (2) Scaling up LLMs effectively improves task completion efficiency for lower-level tasks but fails to enhance performance on high-complexity tasks. This suggests that current performance gains primarily stem from pattern memorization rather than cognitive reasoning. (3) When task complexity surpasses a critical threshold (level 4+), both closed and open-source models experience a performance collapse. This highlights the current limitations of LLMs in modeling long reasoning chains and capturing the complex, dynamic logic between tasks and environments.

5.3.2 Process-Oriented Evaluation

Figure 3 shows the process-oriented evaluation of LLM-MAS. Among closed-source models, GPT-4o demonstrates the strongest collaboration capability, while DeepSeek-V3 performs comparably to other open-source models. We derived three key insights from the experimental results. First, most models (14B+) exhibit higher RC than IC, indicating that LLMs are better at responding to collaboration than initiating collaboration. This is a result of their strong instruction-following capabilities, which make initiating collaboration the primary bottleneck for most LLMs. Second, the collaboration capability of all LLMs declines with increasing task complexity. Moreover, the decline rate is similar across all models, indicating that their ability

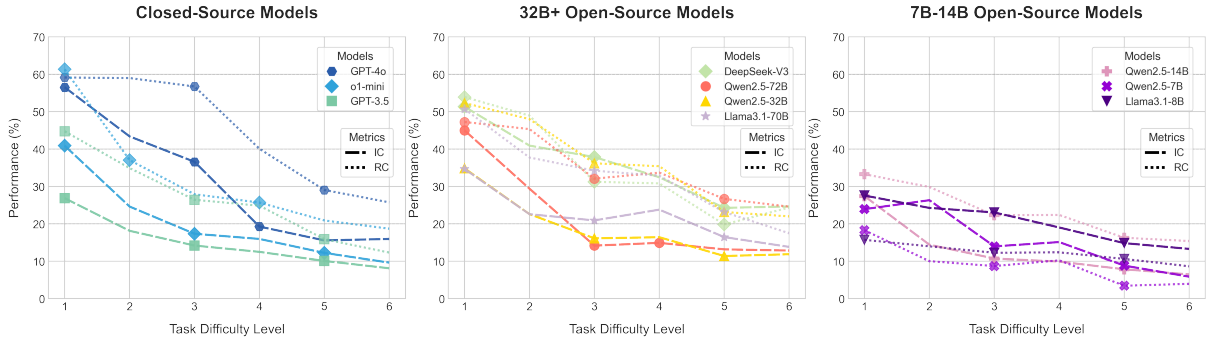

Figure 3: The performance of 10 representative LLMs, with parameter sizes ranging from 7B to 671B+, was evaluated across 6 task levels using the IC, and RC.

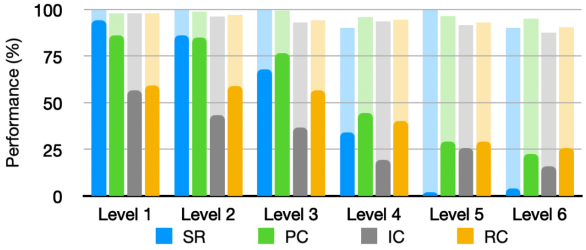

Figure 4: Comparison of human performance (represented by the lighter, more transparent bars) and GPT-4o performance (represented by the solid, more saturated bars) across 6 task complexity levels in our benchmark.

to maintain collaboration capability performance is similar. Despite the scale-up of the models, there is no corresponding improvement in their ability to sustain collaboration capability. Third, compared to GPT-3.5, the CoT-trained model o1-mini demonstrates superior collaboration performance on simpler tasks. Despite the inability to maintain collaboration capability performance as task complexity increases, the improved performance on simpler tasks underscores the potential for further exploration of the CoT-training paradigm in the context of agent collaboration.

5.3.3 Human Performance Evaluation

To establish a performance ceiling, we experimented with 10 human participants completing tasks across 6 complexity levels. We designed a human-computer interaction interface to enable human participants to simulate agent interactions within the environment. Detailed experimental design can be found in Appendix C.2.

As shown in Figure 4, human participants achieved near-perfect and stable performance across all complexity levels, while GPT-4o, the state-of-the-art model in our benchmark, showed a decline in collaboration capability as task complexity increased. This highlights the limitations of LLM-MAS in completing sequential, process-specific tasks in a zero-shot setting, where simply scaling up the LLM is insufficient to improve collaboration performance to human-like levels. The model's reliance on pre-trained knowledge does not fully enable it to adapt to the dynamic and collaborative environment of complex tasks, emphasizing the need for more advanced mechanisms or parameter fine-tuning to enhance its collaborative capabilities to human-like levels.

5.3.4 Failure Analysis

Failure Modes in Collaboration Capabilities Degradation To investigate the temporal dynamics of initiating and responding to collaboration, we selected 4 LLMs and tested them on 5 collaborative actions from level 3 tasks. Using environmental states and memory fragments from interaction trajectories, we constructed prompts to elicit both initiation and response behaviors, evaluated using the ITES function. As shown in Figure 5(a), all models perform well on the first collaborative action, but performance declines in subsequent actions. Regarding initiating collaboration capability, agents fail to identify the appropriate actions needed to advance the task in later steps, revealing a misalignment between environmental states and task flow (further analysis in Appendix C.3.1). The confusion matrix shows a correlation between initiating collaboration and responding to collaboration, indicating that response accuracy depends on the correctness of initiation, confirming that initiating collaboration capability is the primary bottleneck.

Impact of Task Decomposition Ability We isolate the influence of task decomposition by redesigning the recipes with explicit step-to-action mappings, where each step corresponds to a single

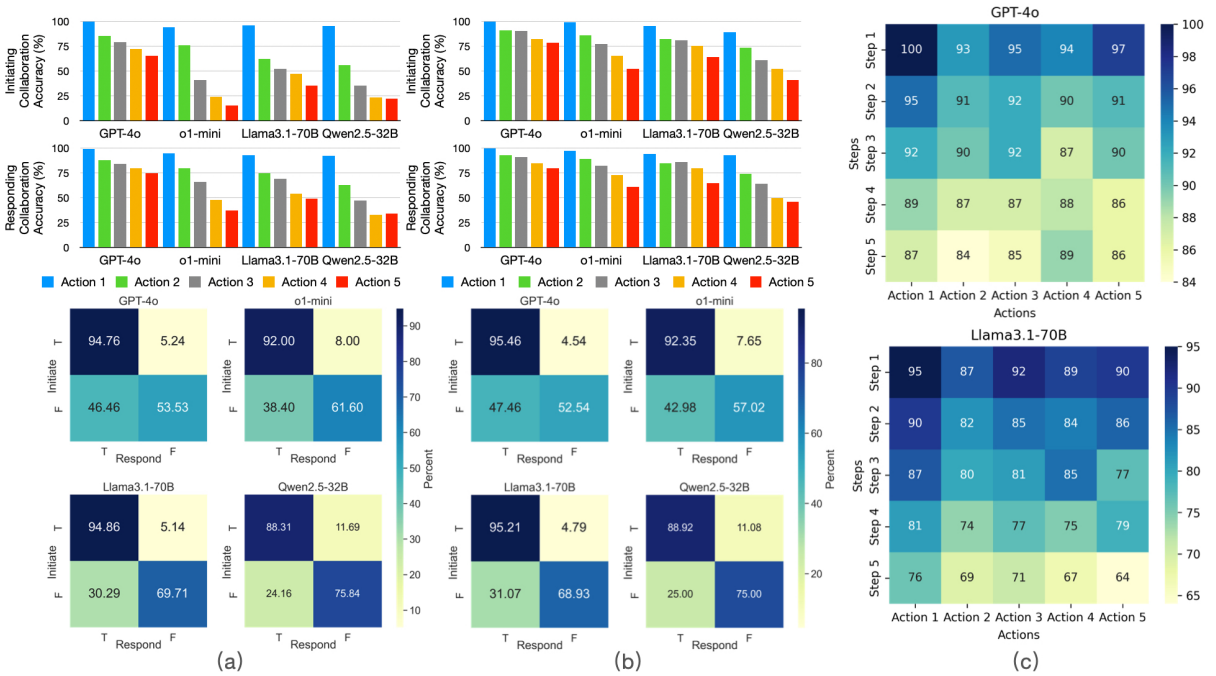

Figure 5: Figure (a) illustrates the dynamic changes in the capabilities of four LLMs in initiating collaboration and responding to collaboration under the original task flow, with the confusion matrix depicting the relationship between the two capabilities. Figure (b) shows the dynamic changes in collaboration capabilities after excluding the impact of task decomposition ability on the task flow. Figure (c) highlights the sensitivity of collaboration capabilities to position, comparing GPT-4o and Llama3.1-70B after adjusting the position of the task workflow.

action in recipe (details in Appendix C.3.2). Figure 5(b) shows this modification leads to performance improvements. However, the gradual decline in accuracy persists, indicating that the degradation of collaboration capabilities is not attributable to limitations in LLM task decomposition abilities.

Sequence Dependence in Collaboration Performance While maintaining step-to-action mappings, we further examined the sensitivity of collaboration performance to position dependencies by rearranging the task workflow (details in Appendix C.3.3). Moving the target collaborative action to the first step led to significant performance improvement, as shown in Figure 5(c). Previously underperforming subsequent actions, when placed at step 1, showed notable gains, and performance degradation largely disappeared. This highlights strong positional dependence in sequential, process-specific tasks, which we attribute to pretraining biases favoring early-sequence elements and limited context tracking in extended action chains.

5.4 Future Challenges

Enhance Collaboration Capability To enhance collaboration, we propose using process-oriented metrics, such as IC and RC, which evaluate the capabilities of initiating and responding to collaboration by scoring each collaborative interaction. Targeted improvements based on these metrics, particularly for smaller models, may help address existing weaknesses and enhance overall performance.

Maintain Collaboration Performance A key challenge in LLM-MAS collaboration is maintaining stable performance, whether within a single task or across tasks of varying complexity. Additionally, a significant gap persists between LLMs and human collaborators, with humans consistently outperforming models. Closing this gap requires improving models' adaptability and robustness to better emulate human collaboration.

6 Conclusion

We introduce the Collab-Overcooked Benchmark, a framework for evaluating LLM-MAS collaboration from both end-to-end and process-oriented perspectives. Experiments across 10 LLMs reveal notable performance gaps, with a key bottleneck in maintaining consistent performance across a single task or tasks of varying complexity. These findings highlight the challenge for further advancements in model adaptability and robustness to enhance collaboration capability across diverse scenarios.

Limitations

The Collab-Overcooked Benchmark is introduced in our paper and we explore methods for evaluating the collaboration capabilities of LLM-MAS using both end-to-end and process-oriented approaches. However, there are three limitations to our work. First, all of our tasks are sequential and process-specific. While we assume that RATs can be exhaustively enumerated, making it possible to use exhaustive RATs as labeled data for evaluating the collaboration capabilities of LLM-MAS. However, in environments with highly complex state and action spaces, RATs are difficult to exhaustively enumerate. In such cases, only representative RATs can be listed as evaluation data, which introduces potential bias into our evaluation methodology. Second, due to the complex mechanisms of LLM-MAS, such as communication, memory, and reflection, the prompts are relatively long (approximately 2,000 tokens, with variation depending on the tokenizer used by the LLM). Additionally, process-oriented evaluation requires substantial interaction data, which leads to both low evaluation efficiency and significant token consumption, which is the common challenge across current methods for evaluating LLM-MAS capabilities. Third, the baseline used to evaluate LLM-MAS is composed of relatively simple structures, with the agent possessing only basic memory and reflection mechanisms, leaving substantial room for optimization.

References

Saaket Agashe, Yue Fan, and Xin Eric Wang. 2023. Evaluating multi-agent coordination abilities in large language models. *arXiv preprint arXiv:2310.03903*.

Micah Carroll, Rohin Shah, Mark K Ho, Tom Griffiths, Sanjit Seshia, Pieter Abbeel, and Anca Dragan. 2019. On the utility of learning about humans for human-ai coordination. *Advances in neural information processing systems*, 32.

Junzhe Chen, Xuming Hu, Shuodi Liu, Shiyu Huang, Wei-Wei Tu, Zhaofeng He, and Lijie Wen. 2024. Llmarena: Assessing capabilities of large language models in dynamic multi-agent environments. *arXiv preprint arXiv:2402.16499*.

Weize Chen, Yusheng Su, Jingwei Zuo, Cheng Yang, Chenfei Yuan, Chen Qian, Chi-Min Chan, Yujia Qin, Yaxi Lu, Ruobing Xie, et al. 2023. Agentverse: Facilitating multi-agent collaboration and exploring emergent behaviors in agents. *arXiv preprint arXiv:2308.10848*, 2(4):6.

Yubo Dong, Xukun Zhu, Zhengzhe Pan, Linchao Zhu, and Yi Yang. 2024. Villageragent: A graph-based multi-agent framework for coordinating complex task dependencies in minecraft. *arXiv preprint arXiv:2406.05720*.

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. 2024. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*.

Ran Gong, Qiuyuan Huang, Xiaojian Ma, Hoi Vo, Zane Durante, Yusuke Noda, Zilong Zheng, Song-Chun Zhu, Demetri Terzopoulos, Li Fei-Fei, et al. 2023. Mindagent: Emergent gaming interaction. *arXiv preprint arXiv:2309.09971*.

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*.

Yue Hu, Yuzhu Cai, Yaxin Du, Xinyu Zhu, Xiangrui Liu, Zijie Yu, Yuchen Hou, Shuo Tang, and Siheng Chen. 2024. Self-evolving multi-agent collaboration networks for software development. *arXiv preprint arXiv:2410.16946*.

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

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

Manling Li, Shiyu Zhao, Qineng Wang, Kangrui Wang, Yu Zhou, Sanjana Srivastava, Cem Gokmen, Tony Lee, Li Erran Li, Ruohan Zhang, et al. 2024. Embodied agent interface: Benchmarking llms for embodied decision making. *arXiv preprint arXiv:2410.07166*.

Chin-Yew Lin. 2004. Rouge: A package for automatic evaluation of summaries. In *Text summarization branches out*, pages 74–81.

Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, et al. 2024. Deepseek-v3 technical report. *arXiv preprint arXiv:2412.19437*.

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

Zhao Mandi, Shreeya Jain, and Shuran Song. 2024. Roco: Dialectic multi-robot collaboration with large language models. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pages 286–299. IEEE.

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.

Siyuan Qi, Shuo Chen, Yexin Li, Xiangyu Kong, Junqi Wang, Bangcheng Yang, Pring Wong, Yifan Zhong, Xiaoyuan Zhang, Zhaowei Zhang, et al. 2024. Civrealm: A learning and reasoning odyssey in civilization for decision-making agents. *arXiv preprint arXiv:2401.10568*.

Yifan Song, Da Yin, Xiang Yue, Jie Huang, Sujian Li, and Bill Yuchen Lin. 2024. Trial and error: Exploration-based trajectory optimization for llm agents. *arXiv preprint arXiv:2403.02502*.

Wei Tao, Yucheng Zhou, Yanlin Wang, Wenqiang Zhang, Hongyu Zhang, and Yu Cheng. 2024. Magis: Llm-based multi-agent framework for github issue resolution. *arXiv preprint arXiv:2403.17927*.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. 2023a. Voyager: An open-ended embodied agent with large language models. *arXiv preprint arXiv:2305.16291*.

Kuan Wang, Yadong Lu, Michael Santacroce, Yeyun Gong, Chao Zhang, and Yelong Shen. 2023b. Adapting llm agents through communication. *arXiv preprint arXiv:2310.01444*.

Wei Wang, Dan Zhang, Tao Feng, Boyan Wang, and Jie Tang. 2024. Battleagentbench: A benchmark for evaluating cooperation and competition capabilities of language models in multi-agent systems. *arXiv preprint arXiv:2408.15971*.

Zihao Wang, Shaofei Cai, Guanzhou Chen, Anji Liu, Xiaojian Ma, and Yitao Liang. 2023c. Describe, explain, plan and select: Interactive planning with large language models enables open-world multi-task agents. *arXiv preprint arXiv:2302.01560*.

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

An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, et al. 2024. Qwen2. 5 technical report. *arXiv preprint arXiv:2412.15115*.

Ceyao Zhang, Kaijie Yang, Siyi Hu, Zihao Wang, Guanghe Li, Yihang Sun, Cheng Zhang, Zhaowei Zhang, Anji Liu, Song-Chun Zhu, et al. 2024a. Proagent: building proactive cooperative agents with large language models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 17591–17599.

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

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

A Benchmark Detail

A.1 Environment

In this section, we provide a detailed overview of the Collab-Overcooked Benchmark environment design. We first introduce the interactive elements within the environment along with their layout. Next, we describe the action space available to agents. Finally, we present the methodology for defining layouts, enabling flexible modifications to the environment.

A.1.1 Interactive Elements

Due to our resource isolation design, the interactive elements available to each agent differ. Figure 6 illustrates the interactive elements that both agents can engage with. We adopt the "Forced Coordination" level design from Overcooked-AI (Carroll et al., 2019), where the two agents share only a single interactive element: the counter. This design necessitates resource exchange between agents to complete tasks.

We categorize interactive elements into three types: utensils, dispensers, and others. The details are as follows:

- Utensils: These interactive elements take one or more ingredients as input and process them according to a predefined synthesis table, transforming them into new ingredients.
- Dispensers: Agents can retrieve ingredients or dishes from these elements, with the available items being predefined.
- Others: The counter serves as a critical interactive element for resource exchange between agents, allowing them to freely place or retrieve ingredients. The delivery location is where agents submit task outcomes. If the submitted ingredient meets the task requirements, the task is considered successful. Otherwise, incorrect submissions result in the removal of the submitted ingredient from the environment, often leading to task failure.

A.1.2 Action Space

The action space of each agent consists of a series of functions in the format "func(args)", which facilitate interactions with the environment or collaboration with other agents. Agent actions are categorized into shared actions and exclusive actions. Shared actions are common to both agents and include actions such as "pickup" (for picking up ingredients), "place_obj_on_counter" (for interacting with the counter), "put_obj_in_utensil" (for placing ingredients into utensils), and "wait". Exclusive actions, on the other hand, arise from the differing interactive elements in each agent's environment. For example, Agent Bob has access to a pot, allowing it to perform the "cook" action, whereas Agent Alice, lacking a pot, cannot perform this action. Conversely, Agent Alice can interact with the chopping board to perform the "cut" action, which Agent Bob cannot. The specific actions available to Agent Alice and Agent Bob are listed as follows:

			Agent Alice	Agent Bob
Interactive Elements	*Utensil*	*Chopping board*	•	
		Blender	•	
		Pot		•
		Oven		•
	Dispenser	*Ingredient*	•	
		Dish	•	
	Others	*Counter*	•	•
		Deliver Location		•

Figure 6: Interactive elements

Listing 1: Action Space List

```
Action Space for Agent Alice:
    1. pickup(obj,place)
    2. cut(chopping_board_name)
    3. stir(blender_name)
    4. place_obj_on_counter()
    5. put_obj_in_utensil(utensil)
    6. wait(num)

Action Space for Agent Bob:
    1. pickup(obj,place)
    2. cook(pot_name)
    3. place_obj_on_counter()
    4. put_obj_in_utensil(utensil)
    5. fill_dish_with_food(utensil)
    6. bake(oven_name)
    7. deliver()
    8. wait(num)
```

To accurately assess collaboration capabilities, we require that when an agent initiates collaboration, the initiating agent must encapsulate the desired action for the responding agent within a "request". This mechanism is utilized for calculating IC and RC. For example, if Agent Bob wants Agent Alice to retrieve an apple for it, Agent Bob will generate the following output: "request(pickup(apple, ingredient_dispenser)); request(place_obj_on_counter())". This request explicitly specifies the sequence of actions that Agent Alice is expected to execute, ensuring that the col-

laboration process is systematically coordinated.

A.1.3 Layout Definition Method

We follow the environment design principles of Overcooked-AI (Carroll et al., 2019) and ProAgent (Zhang et al., 2024a), enabling customization through external layout files. Compared to these prior works, our framework offers a broader range of configurable elements. For instance, the "order_probability" parameter allows users to adjust the probability of tasks appearing randomly in the environment, while the "recipes" parameter enables customization of the synthesis list for each utensil. Further details can be found in the examples provided in our GitHub repository's layout files. Through our enhancements, nearly all aspects of the environment can be customized via a single external file, significantly enhancing the flexibility and scalability of our framework.

A.2 Tasks Construction

In this section, we provide detailed information about tasks, including task complexity level, task list, task recipe, and task RATs.

A.2.1 Task complexity level

To distinguish the complexity level of each task, we define four types of collaborative behaviors performed by the agents. The complexity level of a task is determined based on the minimum number of collaborative behaviors required to complete the task. The four types of collaborative behaviors are as follows:

- Acquiring New Ingredients: This behavior involves retrieving an ingredient from the Ingredient Dispenser. For example, Agent Alice might pick up an onion or an apple from the dispenser.
- Processing the Ingredients: This behavior involves placing ingredients into a cooking utensil. For example, Agent Alice might place an ingredient into a chopping board or a blender.
- Acquiring a New Dish: This behavior involves retrieving a new dish from the Dish Dispenser. This action consists of a single step where Agent Alice picks up a dish.
- Processing the Ingredients by Agent Bob: Similar to the first behavior, but performed by Agent Bob. This includes behaviors like placing an ingredient into a pot or an oven.

Each collaborative behavior corresponds to several collaborative actions. The complexity level of a task is calculated by summing the total number of collaborative actions required from each behavior. Specifically, the number of actions in each of the four categories is counted based on the task's requirements. This approach ensures that tasks with more complex or numerous collaboration requirements are considered more difficult than those with fewer actions. Table 3 provides statistical data on collaborative behaviors and collaborative actions.

Each task's RATs provide the exact number of actions for each type of collaboration, which is used to determine the total complexity level for that task. The complexity calculation allows for a comparison of tasks, ensuring that they are evaluated on the basis of their collaborative complexity.

A.2.2 Task List

Table 4 presents a list of task names across 6 complexity levels, comprising a total of 30 tasks. As indicated by the task names, tasks within the same complexity level share identical workflows, with the only variation being the selection of ingredients. This design aims to mitigate potential biases in LLMs towards specific ingredients, thereby reducing evaluation discrepancies caused by such biases.

A.2.3 Recipes

Each task corresponds to a recipe that outlines the workflow required to complete the task, including the necessary ingredients and cooking steps. There are two important aspects to note regarding the recipe: First, one cooking step typically involves multiple actions by the agents. This necessitates that the agents carefully decompose the cooking step into specific actions after thoroughly understanding both the recipe and the environment. Second, some cooking steps can be executed in a different order. For instance, when multiple ingredients require pre-processing, followed by combining the processed ingredients into a utensil for further preparation, the order in which the ingredients are preprocessed can be interchanged. This decision is typically made by the agents, leading to the possibility of multiple valid RATs for the same task. Allowing such flexibility is both reasonable and aligned with real-world practices. Listing 2 is an example of the recipe for "Baked Pumpkin Soup", which includes the recipe name, required ingredients with quantities, and detailed cooking

Complexity Level	Acquiring New Ingredients	Processing the Ingredients by Agent Alice	Acquiring a New Dish	Processing the Ingredients by Agent Bob	Total Number of Collaborative Actions
Level 1	1	0	0	1	2
Level 2	1	1	1	1	5
Level 3	1	1	1	2	7
Level 4	2	1	1	2	9
Level 5	2	2	1	3	12
Level 6	3	3	1	4	17

Table 3: The number of collaborative behaviors under different complexity levels is given, as well as the total number of corresponding collaborative actions.

instructions.

Listing 2: Recipe example

```
NAME:
Baked Pumpkin Soup

INGREDIENTS:
pumpkin(1)

COOKING STEPs:
1. Cut a pumpkin into slices.
2. Place the pumpkin slices in the oven and bake
     for 3 timesteps.
3. Transfer the baked pumpkin slices to a pot
    and cook for 3 timesteps.
4. Fill a dish with the soup from the pot and
    deliver.
```

A.2.4 Referential Action Trajectory

To evaluate the agents' collaboration capabilities both in terms of end-to-end and process-oriented metrics, we provide the RATs for each task. Given that our tasks are sequential process-specific, we assume that the RATs can be exhaustively enumerated or largely known. We have annotated the RATs for each task, which include the optimal referential action sequences for both agents to complete the task. Each RAT ensures that the agents can accomplish the task with a minimal number of actions, while also employing the optimal strategy to parallelize certain actions for efficiency. A task may have multiple valid RATs, for example, the order in which two ingredients are retrieved may not affect the overall task completion time. During evaluation, the TES and ITES functions select the RAT with the highest matching score as the reference for assessment. Listing 3 provides an example of the RATs for the "Baked Pumpkin Soup" task, with separate RATs for each of the two agents. Because the "Baked Pumpkin Soup" task has only one completed route, there is only one RAT.

Listing 3: RAT of "Baked Pumpkin Soup" task

```
"RAT_1":
{
    "agent_0": [
        "pickup(pumpkin_slices, counter)",
        "put_obj_in_utensil(oven0)",
        "bake(oven0)",
        "pickup(baked_pumpkin_slices, oven0)",
        "put_obj_in_utensil(pot0)",
        "cook(pot0)",
        "pickup(dish,counter)",
        "fill_dish_with_food(pot0)",
        "deliver()"
    ],
    "agent_1": [
        "pickup(pumpkin, ingredient_dispenser)",
        "put_obj_in_utensil(chopping_board0)",
        "cut(chopping_board0)",
        "pickup(pumpkin_slices,chopping_board0)",

        "place_obj_on_counter()",
        "pickup(dish,dish_dispenser)",
        "place_obj_on_counter()"
    ]
}
```

A.3 Baseline

In this section, we introduce the baseline structure and prompt design we use to test different LLMs.

A.3.1 Baseline Construction

Figure 7 illustrates the structure of the baseline and provides an example of agents interacting and collaborating to complete a task within our benchmark. The baseline architecture consists of an Instruction-Builder, Planner, Communication, Error-Handling, Memory, and Reflection modules. The structure remains identical across different agents, with variations arising only in the environment descriptions, action spaces, and task-specific knowledge provided within the prompts.

Instruction-builder The Instruction-builder is a rule-based module responsible for managing and integrating the prompts for each agent. It reads the state dictionary from the environment and fills in a prompt template. The prompt template includes both fixed prompts and slot-based prompts. Fixed

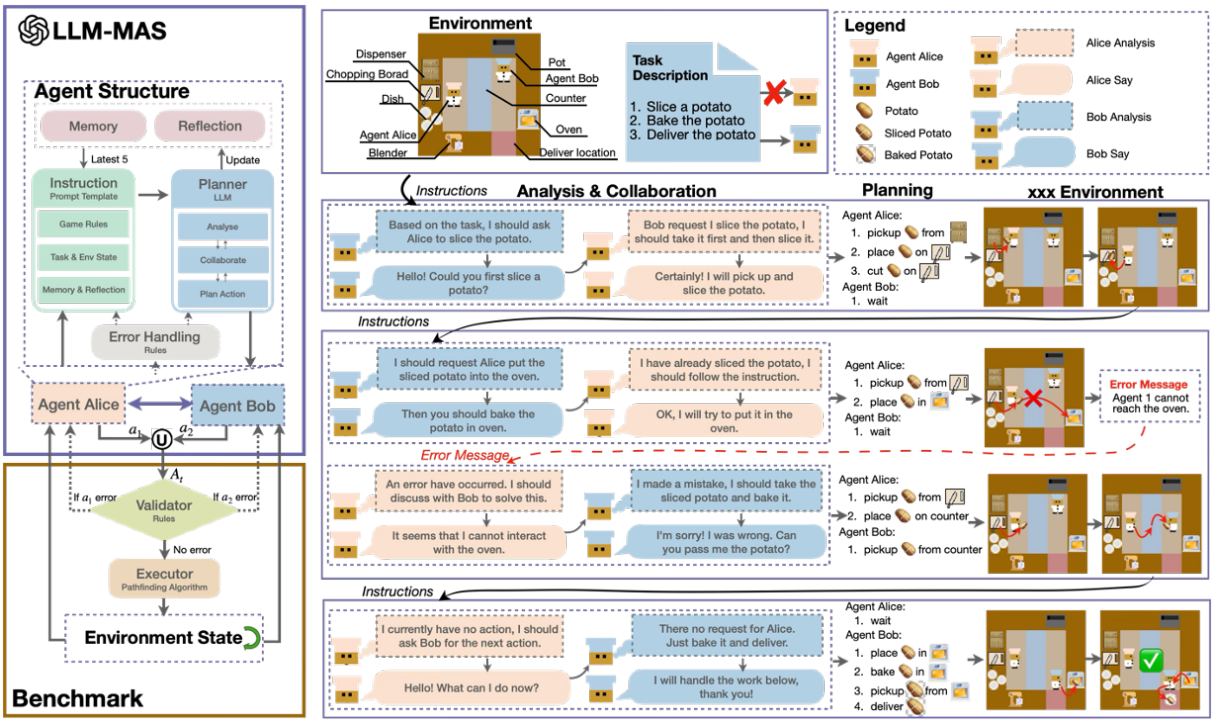

Figure 7: The left side of the figure presents the baseline architecture used for evaluating different LLMs, where Agent Alice and Agent Bob share the same structural design, differing only in their prompt. The right side of the figure illustrates the interaction process between the two agents as they collaborate to complete the "Baked Potato Slices" task within our benchmark. This includes the agents' analytical processes as well as a record of their natural language communication.

prompts contain: (1) game rules, such as objectives, scoring workflows, functions of each kitchen utensils, and methods for preparing dishes; (2) communication rules and output format specifications; and (3) a definition of the agent's action space, along with a brief description of actions available to teammates. Slot-based prompts include: (1) the current recipe for the task (if the agent has access to the recipe); (2) the current environment observations, such as kitchen layout and teammate status; (3) communication records with other agents up to the current time step; and (4) memory and reflection from previous time steps.

Planner The planner is the core decision-making component for the agent. It generates three fields: "Analysis", "Say", and "Plan". The "Analysis" field represents the agent's assessment of the current environment state, task, and past memories, assisting the planner in making informed decisions. The "Say" field determines whether collaboration is required; if the planner identifies a need for collaboration, it generates communication content directly in this field. The "Plan" field contains the action sequence that the planner has devised for the agent.

Communication Communication between agents enables the transmission of collaborative intentions or requests for assistance. When communication content is detected in the "Say" field, all agents enter the communication channel. Within this channel, each agent speaks in sequence until a special token "[END]" is generated or the maximum number of interaction rounds is reached. Once communication is complete, agents formulate their plans based on the information exchanged.

Error-handling The error-handling process manages situations in which the generated actions are deemed invalid by the environment. When an agent receives an error message from the environment, the error information is incorporated into the prompt and re-entered into the planner. This cycle continues until the generated actions are considered valid by the environment or the maximum number of attempts is reached.

Memory and Reflection Memory and reflection represent the accumulation of an agent's past experiences, enabling it to engage in long-term planning. We implement memory and reflection using

Complexity Level	Task Name
Level 1	Baked Bell Pepper
	Baked Sweet Potato
	Boiled Egg
	Boiled Mushroom
	Boiled Sweet Potato
Level 2	Baked Potato Slices
	Baked Pumpkin Slices
	Boiled Corn Slices
	Boiled Green Bean Slices
	Boiled Potato Slices
Level 3	Baked Bell Pepper Soup
	Baked Carrot Soup
	Baked Mushroom Soup
	Baked Potato Soup
	Baked Pumpkin Soup
Level 4	Sliced Bell Pepper and Corn Stew
	Sliced Bell Pepper and Lentil Stew
	Sliced Eggplant and Chickpea Stew
	Sliced Pumpkin and Chickpea Stew
	Sliced Zucchini and Chickpea Stew
Level 5	Mashed Broccoli and Bean Patty
	Mashed Carrot and Chickpea Patty
	Mashed Cauliflower and Lentil Patty
	Mashed Potato and Pea Patty
	Mashed Sweet Potato and Bean Patty
level 6	Potato Carrot and Onion Patty
	Romaine Lettuce Pea and Tomato Patty
	Sweet Potato Spinach and Mushroom Patty
	Taro Bean and Bell Pepper Patty
	Zucchini Green Pea and Onion Patty

Table 4: The names of 30 tasks in total are divided into 6 complexity levels.

a straightforward approach. The memory logs the action sequences that the agent has completed in the past, while the reflection records the previous agent's reflections on invalid actions.

A.3.2 Prompt

In this section, we provide a detailed description of the prompts used to drive LLM-based agents. Since LLM-MAS involves multiple agents interacting within an environment, the prompt design is inherently more complex than that of a single-agent system. Each request to the LLM typically consumes approximately 2,000 tokens, with slight variations depending on the specific tokenizer used by the LLM. To structure this complexity, we categorize the prompts into three key components: game rules, action space definitions, and input-output format specifications. We will elaborate on each component and provide illustrative examples to demonstrate their implementation.

Game Rules The game rules part of the prompt defines the task objective, agent roles, and interaction constraints. It outlines the step-by-step workflow for completing an order, emphasizing task division, coordination, and strict adherence to recipe instructions. Figure 10 shows all the content of the game rule prompt.

Action Space Definitions This part of the prompt defines the action space for Agent Bob, following the action specification method used in ProAgent (Zhang et al., 2024a). It categorizes actions into operation actions (directly executable by the agent) and collaborative actions (requests for the teammate to perform an action). Figure 11 shows the prompt of Agent Bob's action space.

Input-Output Format The input-output format part defines the structured information provided to the agent at each step and the required response format. The input includes past action history, lessons from failures, available utensils, the current order, the planned sequence of actions, and past conversations. The output consists of three fields: analysis (environment assessment and reasoning for actions), plan (the agent's planned actions for the next step), and say (communication with the teammate, if necessary). This structured format ensures that the agent can make informed decisions, coordinate effectively, and execute tasks systematically. 12 shows all the content of the input-output format prompt.

The above section outlines the key prompts used to drive the LLM agents. For further details regarding prompts related to memory, reflection, and other components, please refer to the comprehensive prompts provided in our GitHub repository.

B Evaluation

B.1 Details in TES

The TES is formally expressed as:

$$\text{TES}(\overline{h}_k) = \max_j \left\{ \frac{(1+\beta^2) D^j_{\text{max}}(\overline{h}_k, \overline{g}^j_k)}{m_k + \beta^2 n_k} \right\} \quad (7)$$

where $\overline{h}_k = \bigcup_{t=0}^{T} a^t_k = \{a_1, a_2, \ldots, a_{n_k}\}$ is the historical action sequence up to timestep T of agent k, $\overline{g}^j_k = \{g_i\}_{i=1}^{m_k} \in \mathcal{R}$ is j-th RAT of agent k, β is the hyper-parameter balancing the weight of task progress and redundancy, and $D^j_{\text{max}}(\overline{h}_k, \overline{g}^j_k)$ computes the length of the longest order-preserving subsequence in $\overline{h}_k$ that matches $\overline{g}^j_k$:

$$D^j_{max} = \max_d \{d \mid \forall\, 1 \le i_1 < \cdots < i_d \le n_k, \\ \text{s.t. } a_{i_1} = g_1, a_{i_2} = g_2, \ldots, a_{i_k} = g_k\} \quad (8)$$

It is important to note that the TES function introduces modifications to the Longest Common Subsequence (LCS) calculation in ROUGE-L (Lin, 2004). These modifications are driven by one main reason: Improved identification of redundant actions. Listing 4 illustrates a very common scenario where, due to the agent's incorrect choice in step four, the fifth step fails to advance the task. Specifically, the agent places an irrelevant item, "egg", onto the counter, which does not contribute to the task's progress. In this case, the standard ROUGE-L, based on LCS, would mistakenly consider the agent's fifth action as matching the RAT, leading to an inflated evaluation score.

TES overcomes this limitation by combining maximal order-preserving alignment with efficiency-aware normalization, making it well-suited for collaborative tasks requiring synchronized, sequence-specific interactions.

Listing 4: Comparison of TES with other functions

```
Example:
RAT:
    1. pickup(tofu, ingredient_dispenser)
    2. put_obj_in_utensil(chopping_board_0)
    3. cut(chopping_board_0)
    4. pickup(chopped_tofu, chopping_board_0)
    5. place_obj_on_counter()
Agent Action Trajectory:
    1. pickup(tofu, ingredient_dispenser)
    2. put_obj_in_utensil(chopping_board_0)
    3. cut(chopping_board_0)
    4. pickup(egg, ingredient_dispenser)
    5. place_obj_on_counter()
Result:
    ROUGE-L: 0.8
    TES: 0.6
```

C Supplementary Experiment

In this section, we present supplementary experiments that support the conclusions of the main body. First, we investigate the impact of different hyper-parameter values for γ on the task completion success rate of the LLM-MAS and provide the rationale for selecting $\gamma = 1.5$. Next, we describe the details of the human performance evaluation, including the experimental design and the human-computer interaction interface. Additionally, we introduce new recipes and additional results presented in the failure analysis section. Finally, we provide case studies illustrating both successful and unsuccessful task completions by the LLM-MAS.

C.1 Impact of Varying γ on Task Success Rate

The hyper-parameter γ controls the task failure threshold. Specifically, it determines a time constraint on the task, which is calculated by multiplying the optimal completion time by the value of γ. Clearly, as γ increases, the task success rate (SR) of the LLM-MAS will improve, as the system is allowed more time to complete the task. However, γ cannot be increased indefinitely, as doing so would lead to inefficiencies in the evaluation process. An excessively high value of γ might artificially inflate the success rate, as the extended time window may not reflect the true capabilities of the model in real-world scenarios and it wastes computing resources. On the other hand, setting γ too low could result in an overly strict evaluation, where the system is unable to complete tasks even when it could have more time. Therefore, it is essential to select an optimal value for γ that balances both task success and evaluation efficiency.

Figure 8 illustrates the task success rates of GPT-4o and Llama3.1-70B at 6 complexity levels under

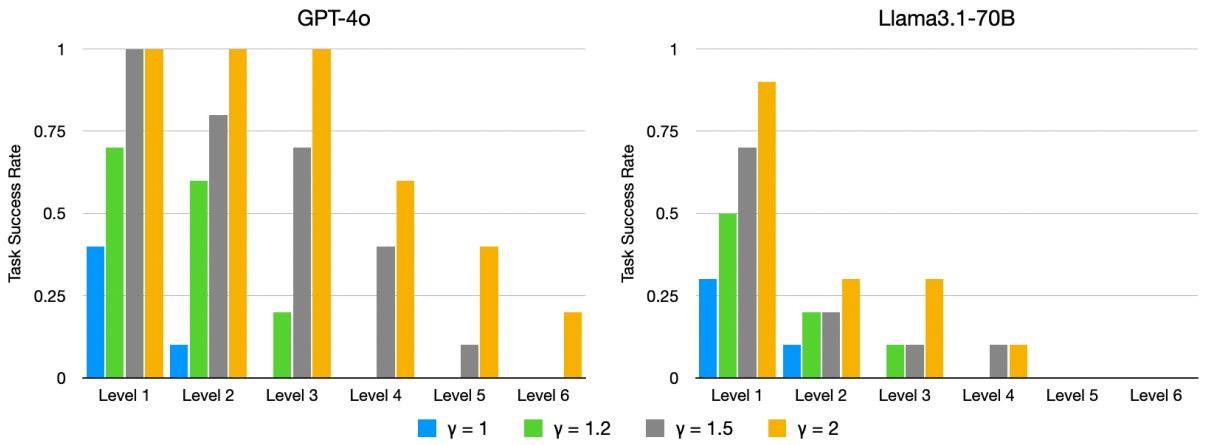

Figure 8: The task success rates of the GPT-4o and Llama3.1-70B at 6 complexity levels under different γ values.

varying values of the hyper-parameter γ. We observed that when $\gamma = 1$, which requires completing tasks along the optimal path, even the state-of-the-art GPT-4o failed to complete the majority of tasks. However, when γ was increased to 1.5 or 2, GPT-4o was able to complete most tasks at complexity levels 4 and below. We chose $\gamma = 1.5$ rather than $\gamma = 2$ because, for models with fewer parameters than GPT-4o, such as Llama3.1-70B, increasing γ does not significantly improve success rates on higher complexity tasks. In fact, most models we tested struggled to complete tasks above level 4, often requiring the maximum time limit during evaluations. By selecting $\gamma = 1.5$, we were able to save approximately 33% of computational resources compared to using $\gamma = 2$, thereby enabling a more efficient evaluation of the LLM's capabilities.

C.2 Human Performance Evaluation

To evaluate human performance on our benchmark, we invited 10 volunteers to participate in our experiments. The participants were divided into five pairs, with each pair assigned two randomly selected tasks from each complexity level. As a result, each complexity level was tested 10 times. To facilitate the understanding of the game rules, action space, input-output format, and the current state of the environment, we designed a human-computer interaction interface. It is important to note that we merely presented the prompts inputted to the agent in a more human-friendly format on the interface, without introducing any additional information. Figure 13 and figure 14 illustrate the layout of our human-computer interaction interface.

C.3 Failure Analysis

In the "Failure Analysis" section of the main body, we designed three experiments to demonstrate that collaboration capabilities tend to decrease as the task progresses, particularly in sequential, process-specific tasks. We attribute this decline to pretraining biases that favor early-sequence task elements, compounded by the diminishing ability to track context across extended action chains. We refer to the experiment corresponding to Figure 5(a) as Experiment A, the experiment in Figure 5(b) as Experiment B, and the experiment in Figure 5(c) as Experiment C. In this section, we will provide detailed information for these three experiments, along with additional analytical results to support our conclusions.

C.3.1 Details in Experiment A

Experiment A selected tasks from Level 3, which involve five distinct collaborative actions. These actions include: "pickup," "put_obj_in_utensil," "cut/stir," "pickup," and "place_obj_on_counter." The parameters for these collaborative actions are not specified, as they vary depending on the specific task associated with each action.

For the preprocessing phase, we manually select environmental states and corresponding memories that require the generation of different collaborative actions from the Level 3 trajectory data. A total of five collaborative actions are chosen, with five scenarios selected for each action. For each model, we test the five scenarios of each collaboration action 20 times, with the prompts being identical to those used in normal testing. The output consists of collaborative actions, which are evaluated based on the ITES. If the collaborative action results in an

ITES score greater than 0, it is deemed a successful collaboration. However, if the ITES score is less than or equal to 0, there unsuccessful collaborative action is categorized manually. For the collaborative actions generated by the initiating agent, the categorization follows three criteria: premature initiation, where the collaborative action should have been generated in subsequent scenarios; repetitive initiation, where the action corresponds to a collaboration that should have occurred in a previous scenario; and irrelevant collaboration, where the action does not belong to any of the expected collaboration actions for the task.

Figure 9 illustrates the error conditions observed in GPT-4o and Llama3.1-70B when initiating collaboration. Both LLMs demonstrate strong collaboration initiation abilities in Action 1. However, as the task progresses, premature initiation and repetitive initiation occur more frequently during subsequent collaborative actions, with this tendency being more pronounced in the smaller Llama3.1-70B model. These results highlight that LLM agents, when faced with sequential, process-specific task workflows, may struggle to accurately track the current step, leading to an increased occurrence of premature and repetitive initiation errors in later stages of the task.

C.3.2 Details in Experiment B

In the recipe used in Experiment A, Step 1 consists of five collaborative actions. To isolate the influence of planning, we redesigned the recipes with explicit mappings from steps to actions. Listing 2 is an example of the recipe used in Experiment A.

Listing 5: Step-to-action mapping recipe of "Baked Pumpkin Soup"

```
NAME:
Baked Pumpkin Soup

INGREDIENTS:
bell pepper(1)

COOKING STEPs:
1. Pick up a bell pepper.
2. Place bell pepper on chopping board.
3. Cut a bell pepper into slices.
4. Pick up bell pepper slices.
5. Place the bell pepper slices on counter.
6. Place the bell pepper slices in the oven and
    bake for 3 timesteps.
7. Transfer the baked bell pepper slices to a
    pot and cook for 3 timesteps.
8. Fill a dish with the soup from the pot and
    serve.
```

We decomposed Step 1 into five distinct substeps, with each sub-step corresponding precisely to a specific collaborative action. Listing 5 is an example of the revised recipe.

By employing this approach, we isolate the influence of planning. However, as demonstrated in the experiments presented in the main body, even with this adjustment, the issue of diminishing collaboration capabilities as the task progresses in sequential, process-specific tasks remains unresolved.

C.3.3 Details in Experiment C

In Experiment C, we rearranged the order of steps in the recipe from Experiment B, placing the collaborative actions to be generated in Step 1 of the recipe. We designed these five steps as a sequence. As shown in Listing 6, when Action 2 corresponds to Step 1, the modified recipe is as follows, where the content in square brackets is supplementary information and will not appear in the experimental recipe.

Listing 6: Rearranged recipe of "Baked Pumpkin Soup"

```
NAME:
Baked Pumpkin Soup

INGREDIENTS:
bell pepper(1)

COOKING STEPs:
[Previously for step 2]
1. Place bell pepper on chopping board.
[Previously for step 3]
2. Cut a bell pepper into slices.
[Previously for step 4]
3. Pick up bell pepper slices.
[Previously for step 5]
4. Place the bell pepper slices on counter.
[Previously for step 1]
5. Pick up a bell pepper.
[The following are not the steps corresponding
    to collaborative action]
6. Place the bell pepper slices in the oven and
    bake for 3 timesteps.
7. Transfer the baked bell pepper slices to a
    pot and cook for 3 timesteps.
8. Fill a dish with the soup from the pot and
    serve.
```

Through these adjustments, we found that the phenomenon of decreasing performance with task progression largely disappeared, highlighting a strong positional dependence in sequential process-specific tasks.

C.4 Case Study

We present case studies of agent collaboration processes, using the DeepSeek-V3 model to illustrate four scenarios: successful initiating and responding, successful initiating but failed responding, failed initiating but successful responding, and

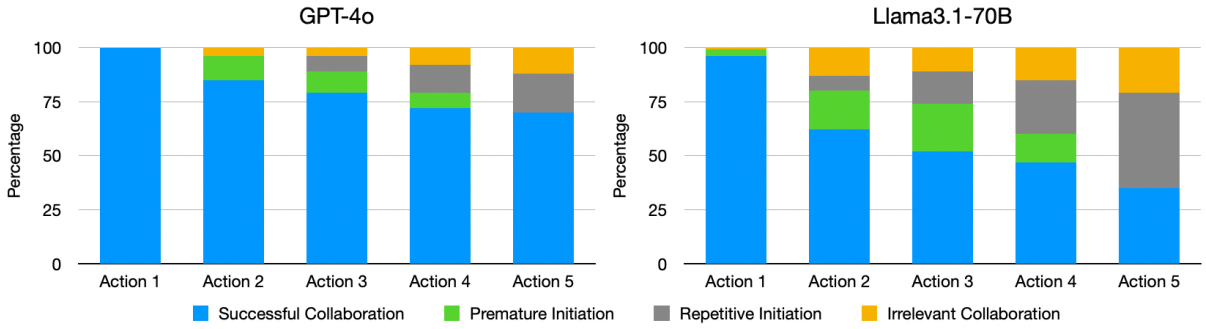

Figure 9: The error condition of GPT-4o and Llama3.1-70B initiating collaboration.

failed initiating and responding. For each case, we provide the agent's environmental state inputs, along with the output of the agent, including the agent's analysis, dialogue, and collaborative actions.

Case 1: Successful Initiating and Responding Listing 7, 8, and 9 presents the input from Agent Bob and the output from Agent Alice in Case 1. To prepare the order, Bob requires Alice's assistance in obtaining ingredients for further processing. As indicated by Bob's analysis field, it correctly understood the first step of the recipe, which is to retrieve a bell pepper from the ingredient dispenser. Furthermore, Bob recognized that it could not access the dispenser and thus requested Alice to retrieve the bell pepper on its behalf. Ultimately, Bob correctly initiated the collaborative action by requesting Alice to take the bell pepper from the ingredient dispenser and place it on the counter. Therefore, Bob's initiation was accurate. Similarly, Alice's plan field correctly generated the two collaborative actions initiated by Bob, which will be executed subsequently. As a result, Alice appropriately responded to the collaboration.

Listing 7: Case 1 Agent Bob's input

```
Successful Action History: []
Bob space:pot0  oven0  counter
Alice space:chopping_board0  blender0
    dish_dispenser  ingredient_dispenser
Order:baked_bell_pepper
Scene 0: <Bob> holds nothing. The planned
    sequence of actions (yet to be performed)
    for Bob is [] <Alice> holds nothing. The
    planned sequence of actions (yet to be
    performed) for Alice is [] Kitchen states: <
    pot0> is empty; <chopping_board0> is empty;
    <oven0> is empty; <blender0> is empty; 3
    counters can be visited by <Bob>. Their
    states are as follows: counters have nothing
```

Listing 8: Case 1 Agent Bob's output

```
Bob analysis: The order is for a baked bell
    pepper. According to the recipe, the first
    step is to pick up a bell pepper. Since only
     the Alice can pick up ingredients from the
    ingredient dispenser, I need to request the
    Alice to perform this action. Once the bell
    pepper is on the counter, I can proceed to
    place it in the oven and bake it.
Bob plan: request('pickup(bell_pepper,
    ingredient_dispenser)'); request('
    place_obj_on_counter()')
Bob say: Alice, please pick up a bell pepper
    from the ingredient dispenser and place it
    on the counter. [END]
```

Listing 9: Case 1 Agent Alice's output

```
Alice analysis: The Bob has instructed me to
    pick up a bell pepper from the ingredient
    dispenser and place it on the counter. Since
     I am the Alice and only I can pick up
    ingredients from the ingredient dispenser, I
     will follow the Bob's instructions. The
    ingredient dispenser is in my space, so I
    can perform this action directly. After
    picking up the bell pepper, I will place it
    on the counter so the Bob can access it.
Alice plan: pickup(bell_pepper,
    ingredient_dispenser); place_obj_on_counter
    ()
Alice say: [NOTHING]
```

Case 2: Successful Initiating but Failed Responding Listing 10, 11, and 12 presents the input from Agent Bob and the output from Agent Alice in Case 2. In this scenario, Bob was supposed to request Alice to retrieve the pumpkin from the ingredient dispenser and place it on the chopping board (chopping_board0) for cutting. However, before requesting Alice to "cut", Bob neglected to generate the action "put_obj_in_utensil(chopping_board0)" to place the ingredient on the chopping board. Since

the ingredient must be placed on the chopping board before it can be cut, Bob's collaborative request was incorrect. Although Alice recognized Bob's intention, it identified the missing action "put_obj_in_utensil(chopping_board0)" during its planning phase and included it. This correction enabled the successful completion of the task to cut the pumpkin. Therefore, Alice's response was successful, as it corrected Bob's mistake.

Listing 10: Case 2 Agent Bob's input

```
Successful Action History: []
Bob space:pot0  oven0  counter
Alice space:chopping_board0  blender0
    dish_dispenser  ingredient_dispenser
Order:sliced_pumpkin_and_chickpea_stew
Scene 11: <Bob> holds nothing. The planned
    sequence of actions (yet to be performed)
    for Bob is [] <Alice> holds nothing. The
    planned sequence of actions (yet to be
    performed) for Alice is [pickup(pumpkin,
    ingredient_dispenser),place_on_counter(),
    pickup(chickpea, ingredient_dispenser)]
    Kitchen states: <pot0> is empty; <
    chopping_board0> is empty; <oven0> is empty;
     <blender0> is empty; 3 counters can be
    visited by <Bob>. Their states are as
    follows: counters have nothing.
```

Listing 11: Case 2 Agent Bob's output

```
Bob analysis: The recipe for Sliced Pumpkin and
    Chickpea Stew requires us to first cut the
    pumpkin into slices, then place the pumpkin
    slices and chickpea in a pot, cook them, and
     finally fill a dish with the stew and serve
    . The Alice is currently planning to pick up
     the pumpkin and chickpea from the
    ingredient dispenser. Once the Alice has the
     ingredients, they will need to place them
    on the counter for me to handle. I will need
     to pick up the pumpkin from the counter and
     cut it into slices using the chopping board
    , but since the chopping board is in the
    Alice's space, I will need to request them
    to perform the cutting action.
Bob plan: request('cut(chopping_board0)'); wait
    (1)
Bob say: Alice, please cut the pumpkin on the
    chopping board after picking it up. [END]
```

Listing 12: Case 2 Agent Alice's output

```
Alice analysis: Based on the Bob's instruction
    in the conversation history, I need to pick
    up the pumpkin from the ingredient dispenser
     and then cut it on the chopping board. The
    Alice space includes a chopping_board0,
    which is available for use. Since the Bob
    has not provided further instructions, I
    will focus on these two tasks. The planned
    sequence of actions for the Alice already
    includes picking up the pumpkin and chickpea
    , but the Bob specifically mentioned cutting
     the pumpkin first. Therefore, I will
    prioritize picking up the pumpkin and
    cutting it.
Alice plan: pickup(pumpkin, ingredient_dispenser
    ), put_obj_in_utensil(chopping_board0),cut(
    chopping_board0)
Alice say: [NOTHING]
```

Case 3: Failed Initiating but Successful Responding Listing 13, 14, and 15 presents the input from Agent Bob and the output from Agent Alice in Case 3. In this scenario, Bob correctly requested Alice to retrieve the cauliflower from the ingredient dispenser and place it on the chopping board (chopping_board0) for cutting, and the collaborative actions generated were also correct. Therefore, the initiation of collaboration was accurate. However, Alice's first action, "pickup(cauliflower, dispenser)", contained an incorrect parameter, as "dispenser" was not a valid option. This error in the first action prevented the execution of subsequent actions, resulting in an incorrect response to the collaboration.

Listing 13: Case 3 Agent Bob's input

```
Successful Action History: []
Bob space:pot0  oven0  counter
Alice space:chopping_board0  blender0
    dish_dispenser  ingredient_dispenser
Order:mashed_cauliflower_and_lentil_patty
Scene 0: <Bob> holds nothing. The planned
    sequence of actions (yet to be performed)
    for Bob is [] <Alice> holds nothing. The
    planned sequence of actions (yet to be
    performed) for Alice is [] Kitchen states: <
    pot0> is empty; <chopping_board0> is empty;
    <oven0> is empty; <blender0> is empty; 3
    counters can be visited by <Bob>. Their
    states are as follows: counters have nothing
    .
```

Listing 14: Case 3 Agent Bob's output

```
Bob analysis: According to the recipe, we need
     to start by cutting the cauliflower into
     slices. Since I don't have access to the
     chopping board or the ingredient dispenser,
     I need to ask the Alice to pick up the
     cauliflower and cut it. The Alice can
     perform these actions in their space. Once
     the cauliflower is sliced, it needs to be
     placed in a pot to cook, which is in my
     space. I will request the Alice to pick up
     the cauliflower and cut it, then place it on
      the counter so I can transfer it to the pot
     .
Bob plan: request('pickup(cauliflower,
     ingredient_dispenser)'); request('
     put_obj_in_utensil(chopping_board0)');
     request('cut(chopping_board0)'); request('
     place_obj_on_counter()')
Bob say: Please pick up the cauliflower from the
      ingredient dispenser, cut it on the
     chopping board, and place it on the counter
     for me to handle next. [END]
```

Listing 15: Case 3 Agent Alice's output

```
Alice analysis: The Bob has instructed me to
     pick up the cauliflower from the ingredient
     dispenser, cut it on the chopping board, and
      place it on the counter. I need to follow
     this sequence since I do not have the recipe
      and must rely on the Bob's guidance.
     Currently, I am not holding anything, and
     all utensils in my space are empty. I will
     start by picking up the cauliflower from the
      ingredient dispenser, then proceed to cut
     it on the chopping board, and finally place
     it on the counter for the Bob to handle next
     .
Alice plan: pickup(cauliflower, dispenser);
     put_obj_in_utensil(chopping_board0); cut(
     chopping_board0); place_obj_on_counter()
Alice say: [NOTHING]
```

Case 4: Failed Initiating and Responding Listing 16, 17, and 18 presents the input from Agent Bob and the output from Agent Alice in Case 4. In this scenario, Bob requested Alice to cut the eggplant on "chopping_board0" and place it on the counter. However, the collaborative actions generated were incomplete, as they omitted the actions "put_obj_in_utensil(chopping_board0)" and "place_obj_on_counter()". As a result, these actions could not be executed, making them incorrect. Although Alice understood Bob's intention, it failed to notice the missing actions in Bob's request and directly replicated his generated actions. Consequently, the response to the collaboration was also incorrect, leading to a failure in the collaboration.

Listing 16: Case 4 Agent Bob's input

```
Successful Action History: []
Bob space:pot0  oven0  counter
Alice space:chopping_board0  blender0
     dish_dispenser  ingredient_dispenser
Order:sliced_eggplant_and_chickpea_stew
Scene 4: <Bob> holds nothing. The planned
     sequence of actions (yet to be performed)
     for Bob is [] <Alice> holds one eggplant.
     The planned sequence of actions (yet to be
     performed) for Alice is [request('
     place_obj_on_counter()')] Kitchen states: <
     pot0> is empty; <chopping_board0> is empty;
     <oven0> is empty; <blender0> is empty; 3
     counters can be visited by <Bob>. Their
     states are as follows: counters have nothing
     .
```

Listing 17: Case 4 Agent Bob's output

```
Bob analysis: The recipe requires preparing a
     Sliced Eggplant and Chickpea Stew. The Alice
      currently holds an eggplant and plans to
     place it on the counter. I need to instruct
     the Alice to cut the eggplant into slices
     and then place them in the pot, along with
     the chickpea. Since the Alice is in charge
     of handling the ingredients, I will focus on
      managing the cooking process in the pot and
      ensuring the dish is served correctly.
Bob plan: request('cut(chopping_board0)');
     request('place_obj_on_counter()')
Bob say: Please cut the eggplant into slices
     using the chopping board and then place the
     slices on the counter. [END]
```

Listing 18: Case 4 Agent Alice's output

```
Alice analysis: I am currently holding an
     eggplant, and the Bob has instructed me to
     cut it into slices using the chopping board
     and then place the slices on the counter.
     The chopping board is in my space, so I can
     perform the cutting action. After cutting, I
      will place the slices on the counter for
     the Bob to use in the next steps.
Alice plan: cut(chopping_board0);
     place_obj_on_counter()
Alice say: [NOTHING]
```

Game Rule Prompt

System

You are an intelligent agent planner; you must generate output and create plans in the specified format, based on the game rules and the environmental status.

User

Suppose you are a Chef proficient in an adapted version of the Overcooked game. Your goal is to cooperate with your teammate, who is also a large language model agent, to complete a single order.
You have recipe, so you need to direct yourself and your teammates to complete the order.
Due to the numerous settings that differ from the original Overcooked game, you must strictly adhere to the following game rules:

Game Rules:

- The Overcooked_AI game requires two players (the chef and assistant) in separate environments to work together with the goal of completing an order in the shortest time.
- To finish one order, your team needs to follow these steps:
  1. Pick raw ingredients from the ingredient dispenser.
  2. Place the ingredients in the correct utensil according to the chef's recipe and initiate cooking using the appropriate action.
  3. Your team may need to repeat Step 2 using several utensils according to the chef's recipe. After cooking, you need to pick up the cooked ingredients and decide whether to move somewhere else.
  4. Check if you need a dish to hold the cooked food. If so, the chef must pick up a dish first and then fill it with food from the utensil. Otherwise, the chef can directly pick up the cooked food from the utensil.
  5. Deliver the food to the serving location immediately.

You cannot complete all of these steps alone. You need to think about what you can do in the situation and what you need your teammate to help you with.
- - The usual workflow for the chef is:
  1. Read the cooking process from your recipe. All of your decisions must be strictly guided by the recipe and should not lead to unfounded behavior.
  2. Ask the assistant to pick up ingredients from the ingredient dispenser and use the correct utensil to handle them according to the recipe. Since you do not have access to all the objects, you need to assign
some tasks to the assistant while you perform other tasks in parallel.
  3. Work in parallel with the assistant to finish the order in the shortest time possible, unless there is nothing you can do in the current situation. If you have nothing to do, you can wait.
  4. Serve the dish (optional). If the recipe specifies that the dish needs to be served on a plate, you must use `fill_dish_with_food(utensil_name)` to serve the dish from the utensil first; otherwise, just pick up the
food from the utensil.
  5. Use deliver().

- The recipe contains all the steps necessary to complete the order. Every choice you make must be based on the recipe.
- You only need to complete one order, so focus solely on the progress of that dish.
- Only the assistant can pick up ingredients from the ingredient dispenser, which has an unlimited supply.
- The utensil is a stationary unit that cannot be moved.
- After placing an ingredient into a utensil, you need to use the correct action to start cooking.
- As long as there is something on the counter, both the chef and assistant can directly pick it up.
- If you wish to place something in another position, first check if you are holding it by verifying through "<Chef> holds XXX" or "<Assistant> holds XXX." Each player can only pick up one item at a time.
- Players can only pass items by placing them on the counter; there is no direct "pass" action. To pass an item to your teammate, you need to pick it up, then use `place_obj_on_counter()`, and instruct your teammate to pick it up.

Figure 10: Prompt for game rules.

Action Space Prompt for Agent Bob

User

Skill: In this game, you can ONLY perform the following allowed actions. Do not attempt to use any other actions that are not listed here. If there are parameters (like steps, utensil), you need to fill them without using quotation marks, angle brackets, etc. There are 2 kinds of actions: 'operation actions' and 'requests'. If you want a teammate to perform a certain operation action, you need to generate a 'request' action with the operation action as an argument.
'Operation actions':

```
def pickup(obj, place):
    if object_in_hand() == "nothing":
        if place in utensil_list or place == "counter" or place == "dish_dispenser" or place == “ingredient_dispenser":
            if object_in_place(obj, place):
                return

def cook(pot_name):
    """ To start cooking in the pot """
    if has_food(pot_name):
        return

def place_obj_on_counter():
    if object_in_hand() != "nothing":
        return

def put_obj_in_utensil(utensil):
    if object_in_hand() != "nothing":
        if utensil in utensil_list:
            return

def fill_dish_with_food(utensil):
    if object_in_hand() == "dish":
        if utensil_food_ready() or utensil_started_cooking():
     return

    """ To start oven example: bake(oven0) """
    def bake(oven_name):
        if has_food(oven_name):
            return

def deliver():
        return

def wait(num):
    # wait positive num timesteps
    if isinstance(num, int) and 0 < num <= 20:
        return
```

‘Collaborative action':

```
"""
Ask a teammate to do an operation action. Each request can only contain one action with quotation marks, so if you need a teammate to perform multiple actions, you need to generate multiple requests. The detailed conversation content you want to convey is contained in '{role} says'.
Example: request('pickup(potato, ingredient_dispenser)'); request('place_obj_on_counter()');...
"""
def request(operation_action):
    teammate.plan_list.append(operation_action)
    send_message(content=self.say, to=teammate)
    return
```

```
Assistant's Skill: Assistant can ONLY perform the following allowed actions. If you need the Assistant to perform actions, you can use the actions below.
def pickup(obj, place):
    pass
def cut(chopping_board_name):
     pass
def stir(blender_name):
    pass
def place_obj_on_counter():
    pass
def put_obj_in_utensil(utensil):
    pass
def wait(num):
    pass
```

Figure 11: Prompt for the action space of Agent Bob.

Input-output Format Prompt

User

Input:
- For each step, you will receive input like the following:
  - 1.Your successful action history in the past steps is: XXX
    - A dictionary of all actions you've successfully performed in recent time steps. Use this information to infer your past plans and continue forward.
  - 2.Here are lessons learned from past failures that can guide your decisions:
    - Reflect on past mistakes to avoid repeating them when making new plans.
  - 3.Chef space: utensil1, utensil2, utensil3, XXX
    - The chef can only use the utensils in the Chef space; you cannot use any utensils outside this area.
  - 4.Order: order
    - You only need to complete the current order.
  - 5.Scene: The planned sequence of actions (yet to be performed) for you and your teammate, status of each ingredient and utensil.
    - "The planned sequence of actions" refers to what each role intends to do in the upcoming time step, and none of these actions are completed yet.
    - Based on the "Scene", gather the existing plans for both roles, along with the status of utensils and ingredients, to plan the next steps efficiently.
    - If a dish is already finished, the chef should consider serving it immediately.
  - 6.Past conversation turns:
    - Assistant says (turn 1):XXX
    - Chef says (turn 1):XXX
    - Assistant says (turn 2):XXX
    - Chef says (turn 2):XXX
    - Each line of conversation history follows this structure: sender of the message + "says" + "turn number". Messages with the same turn number are grouped together.
    - Read the conversation history from top to bottom, with the most recent messages at the bottom.
    - You need to respond to your teammate's most recent message.

Output:
You must provide output in three fields, formatted as follows:
1. Chef analysis:xxxxx
  - This field should include your analysis of the environmental conditions and your reasoning for the actions you plan to take. There are two things to focus on:
    1. Analyze the environment step by step, considering your conversation history with your teammate if "Past conversation turn" exists. Understand where you are in the order and plan based on the recipe.
    2. Analyze which actions are available to you based on the 'Chef space' and 'Assistant space'. Actions that must be done by your teammate should be surrounded by 'request'.
2. Chef plan:action1(params1, params2); action2(params1); ... ; actionN(params1)
  - This field contains the actions you intend to perform in the next time step. Four things to note:
    1. Only generate actions for yourself. If a teammate must perform an action, generate a 'request' with the action as an argument.
    2. The arguments for your actions must all be in your interactive space, or the action is invalid.
    3. Actions should be written in sequence, separated by semicolons, with no additional descriptions or serial numbers. You cannot add any comments or actions not listed in your skill set.
3. Chef say:xxxxx
  - This field refers to the communication you need to convey to your teammate. If you do not plan to communicate, the field should always be [NOTHING].
  - You can either:
    1. [NOTHING] — Meaning there's no need to communicate with your teammate.
    2. The content to pass to your teammate — If you generated a 'request' action in your plan, include a message here to tell your teammate what to do.
  - If you want to end the conversation, add [END] to the last line of your response.
<input>
Your successful action history in the past steps are: []
Here are some lessons you have learned from past failures that you can use to make the right decisions:[]
Chef space:pot0 oven0 counter
Assistant space:chopping_board0 blender0 dish_dispenser ingredient_dispenser
Order: zucchini_green_pea_and_onion_patty
Scene 0: <Chef> holds nothing. The planned sequence of actions (yet to be performed) for Chef is [] <Assistant> holds nothing. The planned sequence of actions (yet to be performed) for Assistant is [] Kitchen states: <pot0> is empty; <chopping_board0> is empty; <oven0> is empty; <blender0> is empty; 3 counters can be visited by <Chef>. Their states are as follows: counters have nothing.

Figure 12: Prompt for the input-outpout format.

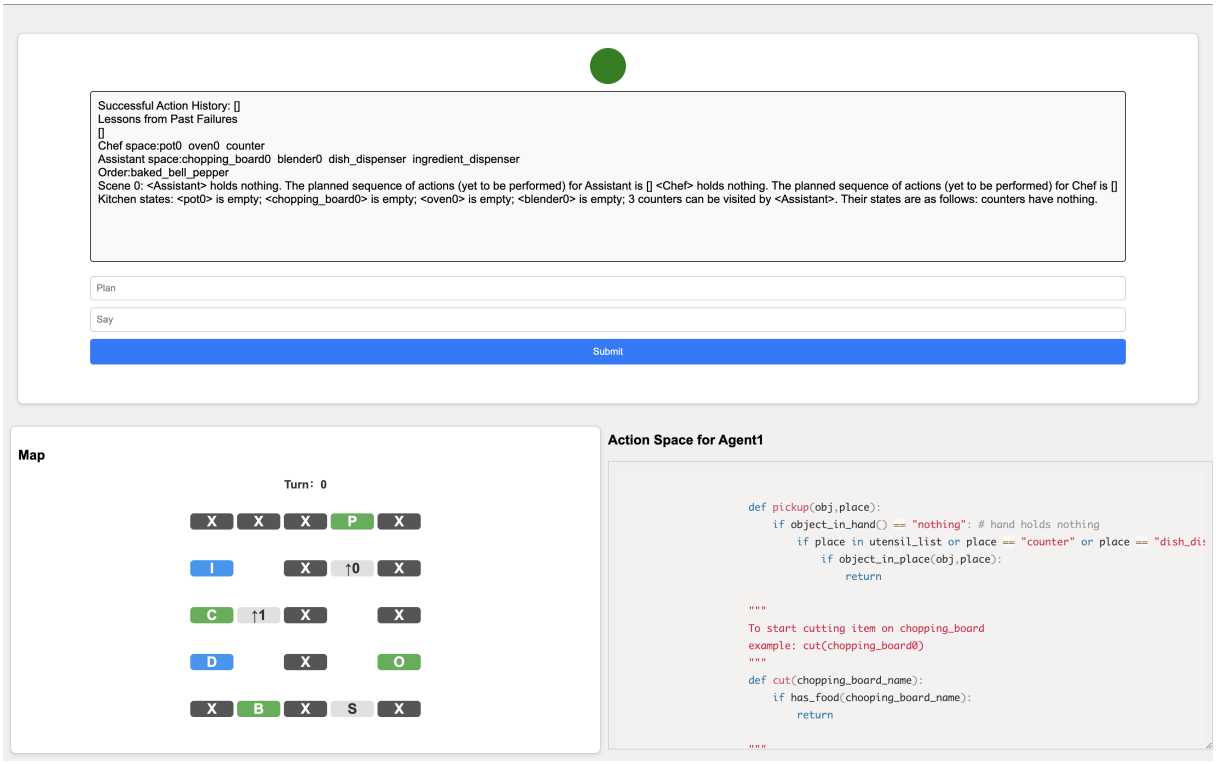

Figure 13: Human-computer interaction as Agent Alice.

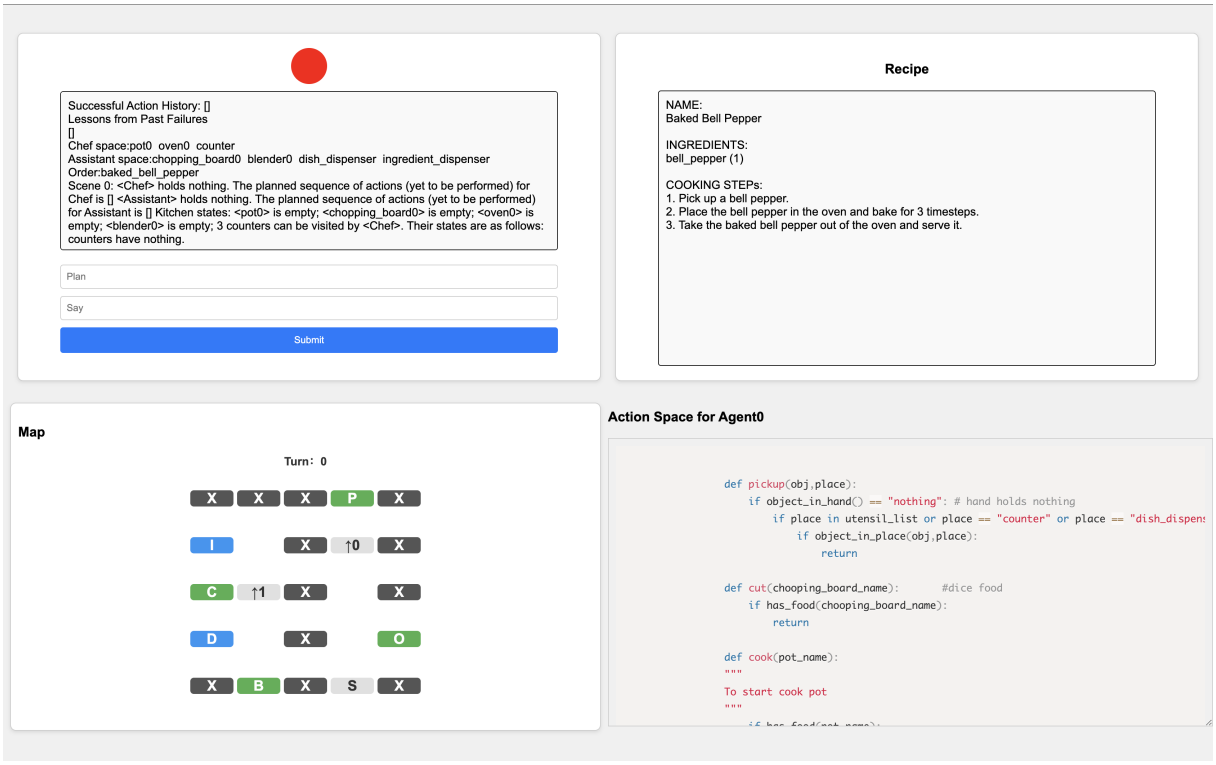

Figure 14: Human-computer interaction as Agent Bob.