DeMAC: Enhancing Multi-Agent Coordination with Dynamic DAG and Manager-Player Feedback

Yuhan Liu[1,2], Cong Xu[2]*, Lu Liu[2], Yihua Wang[1,2], Feiyu Chen[1,2], Qi Jia[2], Yaqian Zhao[2], Zhichun Wang[1], Xiang Li[3],

[1]Beijing Normal University,

[2]Inspur (Beijing) Electronic Information Industry Company,

[3]Key Laboratory of Computing Power Network and Information Security, Ministry of Education,

Qilu University of Technology (Shandong Academy of Sciences)

✉: zxyuhan@mail.bnu.edu.cn, xucong@ieisystem.com

Abstract

Multi-agent systems (MAS) powered by large language models (LLMs) have shown potential in tackling multifaceted problems through advanced understanding and reasoning. However, they struggle to adapt to evolving task dependencies and to handle uncertainties, such as shifting priorities or unpredictable disruptions. These constraints undermine their ability to dynamically adjust long-term strategies and inter-agent collaboration. To address these challenges, we propose **DeMAC**, a **D**ynamic **E**nvironment-Aware **M**anager-Player **A**gents **C**oordination framework that enhances multi-agent coordination through long-term strategic planning. DeMAC uses a dynamically updated directed acyclic graph (DAG) and a Manager-Player Dual-Feedback mechanism to align strategic and operational decisions. Moreover, DeMAC enables agents to maintain collaboration and dynamically adapt to changing environmental conditions, outperforming traditional reinforcement learning and human-agent collaboration in the Overcooked simulation. Experimental results highlight DeMAC's ability to tackle complex coordination tasks, demonstrating its potential to advance LLM-based MAS in dynamic, complex task dependency environments.

1 Introduction

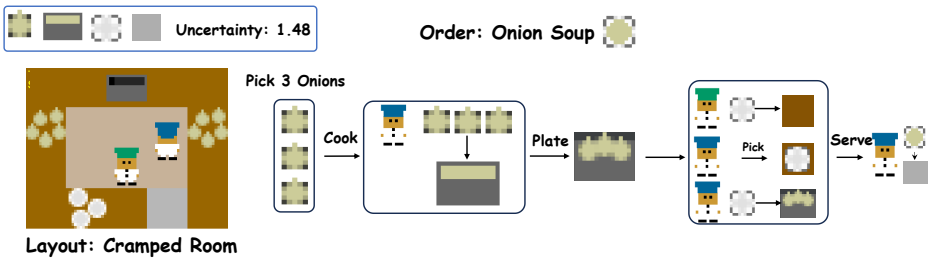
Multi-agent systems (MAS) based on large language models (LLMs) (Achiam et al., 2023; Wang et al., 2024a) have shown significant potential in tackling multifaceted challenges by utilizing LLMs' advanced language understanding, reasoning, and generalization capabilities. LLM-based multi-agents are applied to abstract language tasks including debates, fact verification, and third-party arbitration (Wang et al., 2024c; Hong et al., 2024; Chan et al., 2024; Wang et al., 2024b). Beyond these abstract tasks, recent investigations have begun evaluating their efficacy in physical simulation environments (Park et al., 2023; Wang et al., 2023; Chen et al., 2024b). In these settings, MAS often require strategic reasoning capabilities that not only enable agents to navigate dynamic environments but also anticipate or plan actions from other agents (van der Hoek et al., 2005; Zhang et al., 2024c; Duan et al., 2024).

The Overcooked environment is widely leveraged in MAS research (Wu et al., 2021; Carroll et al., 2019) for its realistic simulation of complex tasks and dynamic interactions, ideal for multi-agent coordination (Tan et al., 2024). The variety of items and actions influences environmental uncertainty and task dependencies. Figure 1a illustrates a scenario where the variety of items and actions are limited, the task flow is relatively fixed, resulting in low-level uncertainty. In low-level uncertainty scenario, agents only require basic coordination because task dependency relationships are clear. In contrast, Figure 1b depicts a dynamically changing environment with diverse task flows, leading to high-level uncertainty driven by factors like ingredient locations, pot states, and resource competition, which complicate task dependencies. Action sequences (e.g., chopping lettuce or onions first) reshape dependencies, influencing task prioritization and coordination. In high-level uncertainty, agents must adapt their coordination strategies to address changing conditions and evolving task dependencies.

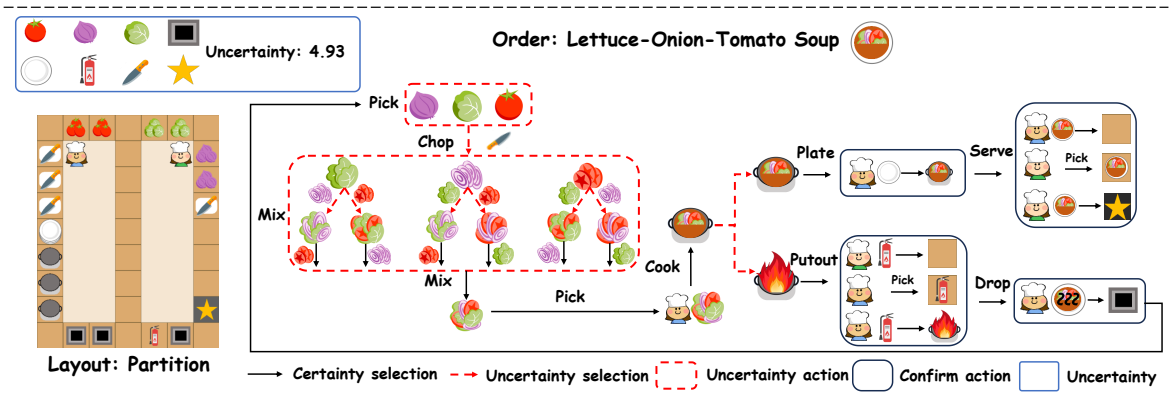
In Overcooked environments, most LLM-based MAS (Zhang et al., 2024a; Agashe et al., 2024; Zhang et al., 2024d; Liu et al., 2024; Yan et al., 2024) rely on a synthesis of reinforcement learning and human-agent interaction. However, in complex task dependency and uncertain environments (Figure 1b), these approaches demonstrate some constraints. Reinforcement learning optimizes individual agent policies but limits global perspective

*✉: Corresponding author

Findings of the Association for Computational Linguistics: EMNLP 2025, pages 14072–14098
November 4-9, 2025



(a) Low-level uncertainty scenario: basic coordination in Cramped Room layout



(b) High-level uncertainty scenario: complex multi-ingredient coordination in Partition layout

Figure 1: Comparison of task uncertainty and dependency in multi-agent cooking processes. (a) In a low-level uncertainty scenario (uncertainty value: 1.48, for specific calculation methods, refer to D.1), agents coordinate in a simpler task with minimal dependencies, preparing onion soup through basic actions. (b) In a high-level uncertainty scenario (uncertainty value: 4.93), agents handle multiple ingredients and complex task dependencies.

and long-term collaboration (Sarkar et al., 2022; Fontaine et al., 2021; Knott et al., 2021; Sutton, 2018). This results in misaligned objectives and inefficiencies in multi-agent collaboration (Nottingham et al., 2023), ultimately undermining cohesive team performance. Human-agent interaction enables decentralized collaboration (Bradshaw et al., 2017; Liu et al., 2024), but it struggles to cope with the dynamic and unpredictable nature of human decisions, such as suddenly extinguishing a fire or deviating from planned tasks. This process frequently leads to re-planning of tasks, ultimately hindering effective coordination. These approaches are insufficient for delivering cohesive long-term strategies and dynamic inter-agent collaboration required for complex task dependency, uncertain environments.

To address these limitations, we propose **D**ynamic **E**nvironment-Aware **M**anager-Player **A**gent **C**oordination Framework (DeMAC), which targets the challenges of task dependencies and uncertainties in MAS. DeMAC utilizes a directed acyclic graph (DAG) (similar to the task flow in Figure 1b) to represent task dependencies and execution order. This ensures that agents follow a consistent, conflict-free plan, addressing the limitations of previous methods, where agent coordination often leads to conflicting goals and inconsistent task execution. DeMAC consists of three key modules: (i) Dynamic DAG Information Update: This module ensures the DAG remains updated to reflect task progress and environmental changes, addressing task dependencies and uncertainty. Dynamically adjusts the task flow, optimizing task order, and minimizing conflicts. (ii) Manager-Player Dual-Feedback: Integrates strategy formulation and high-level planning to synchronize both strategic and operational decisions and employs a reflection loop to optimize agent coordination and ensure alignment with long-term goals. (iii) Environment Perception: Converts environmental data into actionable insights by abstracting grid-level observations into high-level representations, allowing agents to adjust their strategies and reduce uncertainty in decision-making.

In summary, the key contributions of this paper are as follows:

- DeMAC enhances MAS robustness and efficiency with long-term strategic planning and harmonized actions, enabling coherence amid complex dependencies and uncertainties.
- We introduce a dynamic DAG that updates task statuses and dependencies, ensuring precise coordination among agents and improving responsiveness in complex, evolving environments.

- Experimental results show that DeMAC outperforms reinforcement learning and human-agent methods, demonstrating its potential for complex collaborative tasks in MAS.

2 Related Work

2.1 Multi-agent collaboration

MAS are widely used in fields such as distributed control, intelligent transportation, and robotics (Van der Hoek and Wooldridge, 2008). Reinforcement learning enhances MAS collaboration via individual optimization and global rewards (Rashid et al., 2020; Hu and Foerster, 2020; Yu et al., 2022; Knott et al., 2021; Zhang et al., 2023; Qian et al., 2024). Previous methods are limited by fixed task settings and struggle with novel scenarios (Zhang et al., 2024a). MultiAgentBench (Zhu et al., 2025) spans collaboration and competition for broader evaluation. LLMs enhance MAS adaptability and strategy (Chen et al., 2024a; Tran et al., 2025; Li et al., 2023a; Wu et al., 2024). For instance, Li et al. (2023b) evaluated LLM-based agents in Theory of Mind tasks within cooperative games. AutoForm (Chen et al., 2024c) demonstrated efficiency gains from structured communication, while MetaGPT (Hong et al., 2024) streamlined collaboration through role assignment. Both traditional and LLM-based MAS face critical challenges: the former often lack flexibility in dynamic settings, while the latter struggle to maintain coherent alignment with long-term objectives.

2.2 Reasoning in Physical Simulation Environments via LLMs

Studies have explored LLMs for task-solving in physical simulation environments (Huang et al., 2024; Li et al., 2024; Zhang et al., 2024d; Kim et al., 2024; Ma et al., 2025), demonstrating their potential in understanding and planning complex scenarios. DEPS (Wang et al., 2023) applies LLMs for reasoning in physical simulations but focuses on single-agent planning. The LLM Compiler (Kim et al., 2024) utilizes DAG structures to enhance parallel reasoning and coordinate complex tasks efficiently. And coELA (Zhang et al., 2024b) evaluates modular cooperative skills across diverse scenes. VillagerAgent (Dong et al., 2024) applies DAG-based multi-agent planning, but its DAG follows an incremental update mechanism without pruning or structural refinement, which limits its capacity for long-term adaptation and failure recovery. Proagent (Zhang et al., 2024a) uses LLMs to observe and infer the intentions of reinforcement learning agents, facilitating cooperation. Some studies use LLM-human interactions to improve decision-making in dynamic environments by addressing the lack of a global perspective (Liu et al., 2024; Yan et al., 2024; Tan et al., 2024). However, current LLM-based MAS frameworks remain limited in sustaining coherent, long-horizon strategies across complex, partially observable environments like Overcooked, where task interdependence and dynamic replanning are critical.

3 Method

3.1 DeMAC framework overview

In this section, we present the methodology for designing and implementing DeMAC, as shown in Figure 2. DeMAC consists of three components: Environment Perception (§3.2), Manager-Player Dual-Feedback (§3.3), and Dynamic DAG Update Information (§3.4). Environment Perception synthesizes inputs into actionable intelligence. Manager-Player Dual-Feedback facilitates strategy formulation by dynamically assessing tasks and roles while enabling adaptive DAG decomposition for robust planning. Lastly, Dynamic DAG Update Information continuously refines task dependencies, enabling adaptive coordination in response to changing conditions.

The Overcooked environment can be modeled as a simplified Partially Observational Markov Decision Process (POMDP) : $\langle \mathcal{S}, \mathcal{A}, \mathcal{O}, \mathcal{E} \rangle$. Here, $\mathcal{S}$ represents the set of environment states, $\mathcal{A}$ denotes the agents' actions, $\mathcal{O}$ represents observations, $\mathcal{E}$ encapsulates both autonomous environment evolution and state transition dynamics, allowing us to omit an explicit $\mathcal{P}$ without loss of generality.

3.2 Environment perception

The Environment Perception module, illustrated in Figure 3, enhances DeMAC by providing dynamic environmental awareness to support agent decision-making. The Environmental Analysis (EA) component leverages LLMs to process environmental inputs (task orders, player states, and map data) into structured, actionable outputs. By analyzing these inputs, EA helps agents adapt to both external environmental changes and their own actions. The EA defined by the prompt z_e[1].

[1]For details on Agent profiles, please refer to App. A.1

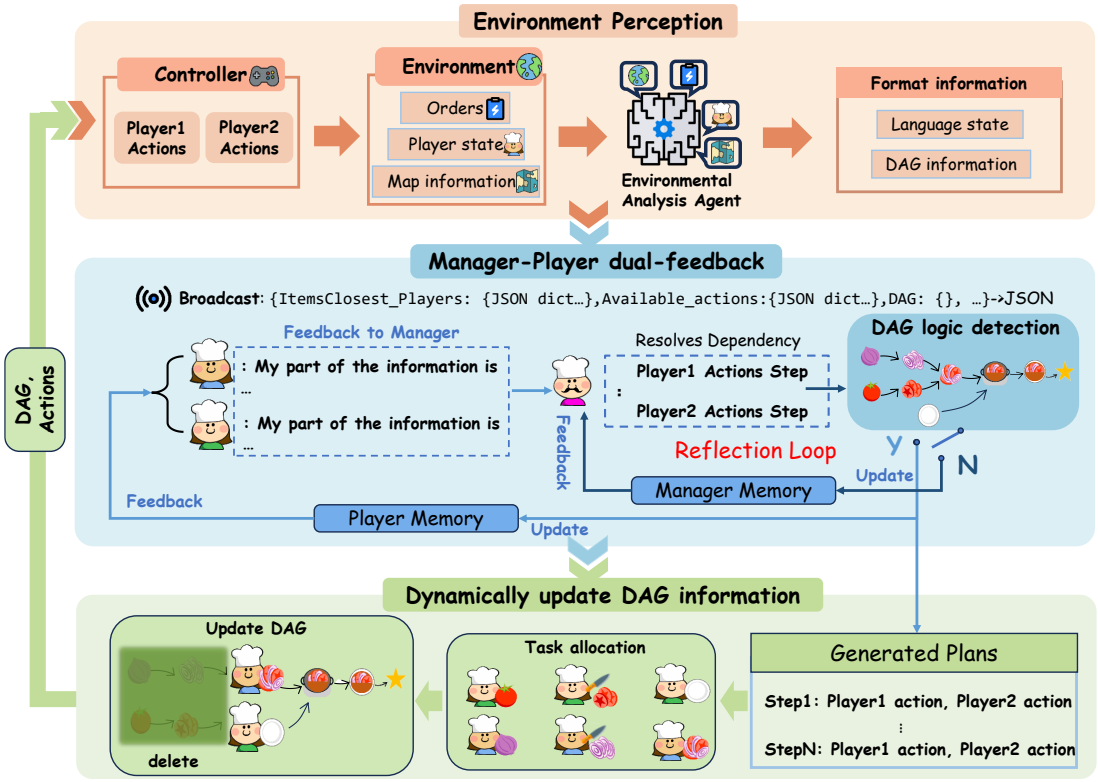


Figure 2: Overview of the DeMAC framework, illustrating Environment Perception, Manager-Player Dual-Feedback, and Dynamically update DAG information. These modules enable adaptive coordination, efficient task allocation, and consistent logical validation for multi-agent operations.

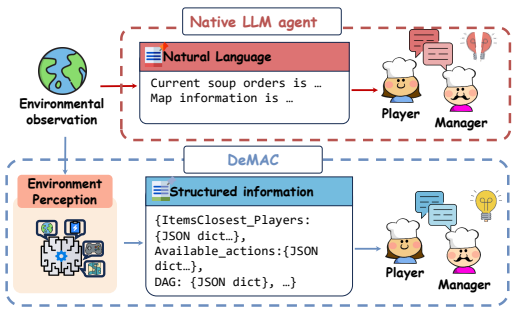


Figure 3: Comparison of native LLM agent communication versus Environment Perception in DeMAC in Overcooked environment.

At each time step t, the observation state $O_t \in \mathcal{O}$ encapsulates orders, player states, and map details, representing the environment. Observations $\mathcal{O}$ are derived based on $\mathcal{S} \times \mathcal{A} \times \mathcal{E} \rightarrow \mathcal{O}$, where the inclusion of $\mathcal{E}$ highlights the independent evolution of the environment, requiring agents to respond to dynamic conditions. Feature extraction from observations follows the transformation:

$$L_t, D_t = LLM_{EA}(p_t, O_t || z_e) \quad (1)$$

where L_t and D_t denote the structured language state and DAG-based task dependencies at time t. Specifically, the structured language state $L_t = (Q_t, R_t, V_t)$ encodes: (i) the set of observed entities Q_t (e.g., object categories, attributes, contextual tags), (ii) the relative spatial configuration R_t (distances, reachable zones, and spatial relations), and (iii) the valid actions V_t available to the agent given current affordances (e.g., pick, cook, chop). The prompt p_t incorporates partial information from (L_t, D_t), enabling it to adapt dynamically to the evolving environment and current task context.

In this DAG structure[2], nodes represent item states (e.g., Chopped Lettuce, Cooked Soup), while edges represent state transitions caused by actions (e.g., Chop, Cook). DAG ensures actions follow the correct sequence, preventing cycles or deadlocks, and effectively managing task execution for timely completion.

3.3 Manager-Player dual-feedback

The Manager-Player Dual-Feedback module is essential to the DeMAC framework and facilitates cohesive multi-agent coordination through hierarchical feedback. Environmental data from the Environment Perception (L_t, D_t) is broadcasted to

[2]For more specific DAG design, please refer to App. C.1

all agents, ensuring synchronized understanding. Operating at two levels, this dual-feedback mechanism continuously refines task assignments and supports long-term strategic planning based on dynamic environmental outputs.

Player-Level Feedback. At this level, Player-Agents generate descriptive information about their local environment, including nearby objects, feasible actions, as well as other relevant information, instead of output actions directly:

$$m_t^i = LLM_{player}^i(L_t, D_t || z_p^i) \tag{2}$$

where m_t^i represents the information generated by Player-Agent i, which provides contextual insights like nearby objects and possible actions. z_p^i is the system prompt that defines the behavior and role of Player-Agent i.

Manager-Level Strategic Planning. The function of the Manager-Agent can be formalized as $LLM_{manager} : (\mathcal{M} \times \mathcal{D} \times \mathcal{L}) \rightarrow \mathcal{A}$, where $\mathcal{M}$ represents the collective memory or refined information (m_t^i) provided by each Player-Agent and $\mathcal{A}$ represents the generated actions. The Manager-Agent utilizes a DAG D_t, to analyze task dependencies and plan task to Player based on conditions and feedback:

$$A_{t+T} = LLM_{manager}(D_t, L_t, m_t^1, \ldots, m_t^n || z_m) \tag{3}$$

where A_{t+T} denotes the sequence of actions for Player from time t to $t+T$. z_m is the system prompt that defines the Manager-Agent's overall strategy and objectives, guiding it in processing feedback and dynamically adjusting task assignments. Refined feedback from Player-Agents (m_t^i) is integrated to dynamically adjust task assignments, ensuring a coherent and adaptive task sequence that meets overall system goals.

During each planning step $(t, t+1, \ldots, t+T)$, the Manager-Agent deconstructs the DAG to derive specific tasks and assigns them to Player-Agents[3], maintaining consistent progress towards system objectives.

Manager-level Feedback.[4] DeMAC follows a *centralized planner with decentralized perception*: the manager maintains a task-DAG while players act locally and return structured feedback. This provides long-horizon consistency under strong dependencies but introduces risks of a single point of failure and stale global state. We therefore add a **manager-level feedback** loop that monitors planner health and plan freshness. After Manager planning, the Manager-Agent uses the DAG Logic Detection to validate the logical consistency of the plan:

$$d_t = DAG_{Logic}(D_t, A_{t+T}) \tag{4}$$

where d_t represents the outcome of the logical check. If inconsistencies are detected, the Manager-Agent replans and adjusts the tasks accordingly:

$$A_{(t+T)_{new}} = LLM_{\text{Manager}}(D_t, L_t, d_t, m_t^1, \ldots, m_t^n || z_m) \tag{5}$$

where $A_{(t+T)_{new}}$ is the updated action sequence ensuring compliance with DAG constraints. Part of output result is shown as $\{\text{Step } t : \text{Player1: } X_t, \text{Player2: } Y_t\}_{t=1}^T$.

The reflection loop improves task accuracy and efficiency by incorporating continuous feedback and memory updates, which ensures alignment in task progress across agents. Based on the result d_t, either the Manager agent or Player memory is updated. If the logical check fails, the Manager agent corrects inconsistencies. Otherwise, the Player memory stores current allocations and state.

3.4 Dynamically update DAG information

The Dynamically Update DAG Information Module maintains task consistency by adapting to evolving dependencies and updating task allocations in the Overcooked environment. It consists of three key components: Plan Generation, Task Allocation, and DAG Update. This module ensures task dependency consistency, supporting DeMAC's coordination in dynamic environments.

Plan Generation and Task Allocation. The Manager-Agent evaluates the current DAG (D_t), state (L_t), and feedback from Player-Agents (m_t^i) to construct a multi-step action plan, where each step defines the actions assigned to each Player-Agent. This plan ensures synchronized execution and minimizes conflicts by aligning with both the current environment state and the long-term strategic objectives of the task.

DAG Update. After task allocation, the DAG is updated to reflect changes in task progress and environmental conditions:

$$D_{t+T} = UpdateDAG(D_t, A_{t+T}) \tag{6}$$

[3] For a comprehensive description of the configuration files for Manager and Player agents, refer to App.B.1

[4] See App. B.2 for details of the feedback-reflection loop.

where D_{t+T} represents the updated DAG, keeping plans and allocations aligned with the current state. Dynamic DAG Update ensures that task progress and dependencies are aligned with conditions. $UpdateDAG$ involve adding new nodes for incoming tasks, removing completed nodes based on task status, and adjusting dependencies dynamically to maintain task feasibility.

4 Experiments

In this section, we present the experimental setup and results to evaluate the DeMAC framework's performance. We conducted experiments in two Overcooked environments[5] (Wu et al., 2021; Carroll et al., 2019) to assess DeMAC under varying task complexities. The experiments focus on evaluating the effectiveness of DeMAC in terms of task completion, efficiency, and adaptability across different layouts and environments.

4.1 Experimental setup

Classic Environment. The classic environment (Carroll et al., 2019), illustrated in Figure 1a, tasks agents with collaboratively preparing as many soups as possible under time constraints. Five distinct layouts are employed, spanning challenges from fine-grained motion coordination to strategic task planning, enabling a comprehensive evaluation of foundational agent capabilities.

Extended Environment. The extended environment (Wu et al., 2021; Liu et al., 2024) introduces dynamic task flows to evaluate DeMAC's adaptability. Agents manage varying soup orders with strict time limits—delays incur penalties, and unattended soup may burn and cause fires requiring intervention. This setting imposes planning pressure, challenging both coordination and decision-making under evolving conditions.

Metrics.[6] In accordance with the standard evaluation protocol of the Overcooked benchmark, our experiments are conducted with two Player agents by default. We adopt established metrics to evaluate agent performance across environments: (i) **Game Score**: Agents earn points by completing food orders. In the classic environment, only onion soup is required, whereas the extended environment includes a dynamic mix of dishes. (ii) **Steps and Success Rate**: We measure the number of steps required for completing simpler tasks (e.g., vegetable salad) and compute the corresponding success rate. (iii) **Token Consumption**: To assess computational efficiency, we track the number of LLM tokens consumed during task execution. (iv) **NQ (Number of Queries)**: The number of LLM queries made reflects the planning efficiency and system responsiveness (Zhang et al., 2024d).

4.2 Results and Analysis

Classic Environment Performance. Table 1 compares the average performance of SP (Carroll et al., 2019), PBT (Jaderberg et al., 2017), FCP (Strouse et al., 2021), MEP (Zhao et al., 2023), COLE (Li et al., 2023c), CAC (Agashe et al., 2024), VillagerAgent (Dong et al., 2024), ProAgent (Zhang et al., 2024a) [7], and our DeMAC. The results show that some methods struggle with complex task interdependencies due to their inability to deconstruct DAG dependencies, leading to local optimizations that overlook broader task relationships. In contrast, DeMAC dynamically disassembles the DAG, adjusting task allocations based on environmental information, ensuring that tasks are executed in the optimal order, even in complex scenarios. The key observations are as follows: (i) Superior Performance Across Layouts: DeMAC consistently achieves superior scores across all the Overcooked layouts compared to the reinforcement learning baselines (PBT, MEP, SP, COLE, FCP). (ii) Adaptability and Robustness: DeMAC demonstrates competitive robustness under high-uncertainty layouts such as Forced Coord. and Count. Circ., where its performance surpasses all LLM-based and RL baselines. On Cramped Rm., Asym. Adv., and Coord. Ring., VillagerAgent achieves higher scores, whereas DeMAC remains competitive and consistently outperforms CAC and ProAgent under the same backbones. (iii) Enhanced LLM-based Performance: The gains of DeMAC hold for both GPT-3.5-Turbo and GPT-4-Turbo/4o, indicating that the improvements stem from our DAG-aware coordination and retrieval rather than merely from a stronger language model.

Extended Environment Performance. We adopt a step-by-step approach to testing, starting with simpler tasks. Table 2 shows the average performance of various methods across these simpler cooking tasks, including comparisons with reinforcement learning-based approaches such as FB

[5] For details of tasks, refer to the App. D.3 and D.4

[6] For details of metrics, please refer to App. D.

[7] ProAgent with GPT-4 Turbo results are based on our re-evaluation under the same experimental settings to ensure a fair comparison.

Table 1: Performance comparison across different Overcooked layouts in terms of average score (Classic Environment). The number in brackets in the first column represents the uncertainty value of the layout.

AI Agent	Cramped Rm. (1.48)	Asym. Adv. (1.61)	Coord. Ring (1.62)	Forced Coord. (1.93)	Count. Circ. (2.06)
PBT	179.8 ± 26.8	182.2 ± 27.9	141.3 ± 28	61.7 ± 46	64.7 ± 45.9
MEP	185 ± 15	184.7 ± 41.8	167.2 ± 22.4	39.3 ± 16.9	81.5 ± 27.5
SP	168.5 ± 15.2	183.3 ± 27.5	133.3 ± 23.7	30.2 ± 21.9	64.7 ± 45.8
COLE	169.2 ± 16.8	201.3 ± 34.5	168.8 ± 26.1	57.3 ± 36.4	100.8 ± 31.1
FCP	196 ± 11.9	185.7 ± 22.7	148.8 ± 19.4	44.7 ± 36.4	60 ± 38.3
$\text{CAC}_{\text{GPT-3.5-Turbo}}$	33.3 ± 10.9	46.6 ± 10.9	40.0 ± 0.0	66.6 ± 14.4	53.3 ± 5.4
$\text{ProAgent}_{\text{GPT-3.5-Turbo}}$	197.3 ± 6.1	229.8 ± 21.9	183 ± 31.7	49.7 ± 33.1	128.5 ± 28.1
$\text{DeMAC}_{\text{GPT-3.5-Turbo}}$	202.4 ± 13.1	246.7 ± 23.8	183.6 ± 10.2	158.7 ± 8.9	161.3 ± 4.9
$\text{CAC}_{\text{GPT4-Turbo}}$	173.3 ± 6.7	260.0 ± 11.6	140.0 ± 0.0	180.0 ± 11.6	160.0 ± 0.0
$\text{ProAgent}_{\text{GPT4-Turbo}}$	202.4 ± 24.6	255.3 ± 22.5	169.6 ± 35.4	68.67 ± 71.1	132.8 ± 24.6
$\text{VillagerAgent}_{\text{GPT4-Turbo}}$	213.3 ± 9.4	304 ± 8.76	226.7 ± 18.9	120 ± 16.97	148 ± 4.38
$\text{DeMAC}_{\text{GPT4-Turbo}}$	205.7 ± 9.4	289.3 ± 14.4	192.0 ± 9.8	193.3 ± 20.2	168.3 ± 12.2
$\text{DeMAC}_{\text{GPT4o}}$	208.0 ± 9.8	281.4 ± 23.8	188.0 ± 9.8	184.1 ± 13.1	165.3 ± 8.8

Table 2: Performance comparison of various methods for different cooking tasks in terms of steps and average success rate (Extended Environment).

Method	Tomato (Step ↓)	Tomato-Lettuce (Step ↓)	Tomato-Lettuce-Salad (Step ↓)	Average Success (↑)
BD	23.71	42.39	44.32	0.98
UP	25.34	57.32	48.64	0.94
FB	25.63	45.44	47.33	0.95
D&C	31.42 (Except-Full)	58.65 (Except-Full)	83.78 (Except-Full)	0.61
Greedy	32.91 (Except-Full)	63.21 (Except-Full)	74.13 (Except-Full)	0.57
HLA*	**42.15**	**65.93**	**58.31**	**0.87**
DeMAC	**26.04**	**41.87**	**34.25**	**0.96**

"Except-Full" means that the method cannot complete the task on the "Full-Divider" map, and the average value is calculated after excluding this map. "*" represents human-machine collaboration.

Table 3: Performance comparison across different Overcooked layouts in terms of average score (Extended Environment). The numbers in brackets in the first column indicate the uncertainty values of the layouts, while those in the remaining columns represent the standard deviations.

Method	Ring (4.35)	Partition (4.66)	Bottleneck (4.93)	Quick (4.20)
$\text{SMOA*}_{\text{GPT-3.5-Turbo}}$	$80.9_{(29.1)}$	$33.0_{(27.1)}$	$102.4_{(35.6)}$	$60.8_{(48.5)}$
$\text{FMOA*}_{\text{GPT-3.5-Turbo}}$	$92.5_{(21.7)}$	$57.7_{(37.9)}$	$103.8_{(30.6)}$	$71.2_{(50.2)}$
$\text{HLA*}_{\text{GPT-3.5-Turbo}}$	$114.4_{(19.4)}$	$100.3_{(36.4)}$	$130.3_{(19.7)}$	$117.2_{(45.3)}$
$\text{DeMAC}_{\text{GPT-3.5-Turbo}}$	$\mathbf{120.3}_{(12.1)}$	$\mathbf{112.7}_{(8.0)}$	$\mathbf{136.1}_{(10.8)}$	$\mathbf{126.4}_{(15.1)}$
$\text{DeMAC}_{\text{Qwen2.5-Max}}$	$\mathbf{115.3}_{(8.3)}$	$\mathbf{110.3}_{(10.8)}$	$\mathbf{126.5}_{(17.4)}$	$\mathbf{141.5}_{(13.1)}$
$\text{DeMAC}_{\text{Llama-3.3}}$	$\mathbf{118.3}_{(16.8)}$	$\mathbf{108.8}_{(14.9)}$	$\mathbf{127.8}_{(19.6)}$	$\mathbf{136.9}_{(21.9)}$

"*" represents human-machine collaboration.

(Tesauro et al., 1995), UP (Sutton, 2018), D&C (Ephrati and Rosenschein, 1994), Greedy (Mnih et al., 2015), and BD (Wu et al., 2021).

Key insights from Table 2: DeMAC outperforms in the Tomato-Lettuce-Salad task due to its advanced global planning and multi-step coordination. In contrast, reinforcement learning methods, optimized for short-term rewards, often assume "local optimum equals global optimum" and tend to excel in tasks with smaller state spaces. D&C and Greedy fail in Full-Divider because they cannot explicitly coordinate on the same subtask to pass objects across the counters. DeMAC demonstrates substantially fewer steps to complete tasks compared to HLA*, the LLM human-agent interaction baseline, and has a success rate that is 9% higher.

Table 3 shows game scores across various Overcooked layouts, with SMOA* and FMOA* being variants of HLA* (Liu et al., 2024), an LLM-based human-agent collaborative framework. To assess the generality of the DeMAC framework, we tested it using several models, including GPT-3.5-Turbo (the same underlying model as HLA), Qwen-Max (Yang et al., 2024), and the open-source Llama (Dubey et al., 2024). The results show consistent performance across all models, demonstrating DeMAC's robustness in dynamic task planning and coordination, independent of the LLM used.

Under the same LLM, DeMAC outperforms other methods in all layouts, especially in high-uncertainty environments like "Partition" and "Quick", with improvements of 12.36% and 7.85%, respectively. The higher scores and lower variance highlight its superior strategic planning and coordination. As uncertainty increases (e.g., in Partition (4.66) and Bottleneck (4.93)), other methods struggle, leading to fewer completed orders and lower scores. In contrast, DeMAC dynamically

Table 4: Number of Queries (NQ) to LLMs for task completion across different layouts (Classic Environment). *Overall* represents the average score across 400 timesteps. *First Order* indicates the performance on completing the first order in the environment. *Second Order* measures performance on the subsequent order, reflecting the system's ability to maintain consistency and efficiency across sequential tasks.

Method	Cramped Rm.			Forced Coord.		
	Overall	**First Order**	**Second Order**	**Overall**	**First Order**	**Second Order**
Proagent	96.8 ± 25.0	9.1 ± 3.96	22.7 ± 8.24	580.6 ± 65.62	111.3 ± 6.20	192.8 ± 72.55
CAC	518.8 ± 19.76	58.2 ± 14.46	116.1 ± 7.34	563.11 ± 64.99	86.7 ± 29.96	177.3 ± 31.63
READ-J	–	23.9 ± 1.49	–	–	27.2 ± 0.53	–
DeMAC	$\mathbf{27.5 \pm 3.56}$	$\mathbf{2.3 \pm 0.64}$	$\mathbf{4.9 \pm 1.58}$	$\mathbf{38.5 \pm 3.75}$	$\mathbf{6.1 \pm 1.14}$	$\mathbf{9.8 \pm 2.99}$

Table 5: Ablation study on the DeMAC framework (Extended Environment). *First Order Step*: Measures the efficiency in completing primary tasks. *Second Order Step*: Evaluates the ability of agents to handle dependencies and long-term planning. **Manager is the core of planning, it cannot be removed.**

DeMAC Variants	Env Perception	DAG Info	Player-level Feedback & Reflection Loop	First Order		Second Order Step ↓	Score ↑
				Succ. (%) ↓	Step ↓		
Variant 1	✓	✓	✓	85.9	80.9	132.9	123.6
Variant 2	✓	✗	✓	81.5	88.3	156.2	106.3
Variant 3	✓	✓	✗	80.0	87.4	165.4	98.7
Variant 4	✗	✓	✓	82.1	86.3	145.1	110.9
Variant 5	✗	✗	✓	78.3	90.4	177.8	86.7
Variant 6	✗	✗	✗	24.7	198.7	324.1	42.1

adjusts its task dependencies via its updated DAG structure, maintaining high performance even in complex scenarios.

Token Consumption and LLM Queries. Token consumption, defined as the total tokens used per action, reflects computational efficiency and resource usage. Figure 4 shows the token consumption required for generating actions by various methods. The consumption of HLA is 4.12 times that of DeMAC, and the consumption of ProAgent is 3.61 times that of DeMAC. The results suggest that DeMAC's strategic planning and efficient decision-making lead to lower token consumption. DeMAC consumes more tokens in the extended environment, but still less than the ProAgent tested in the classic environment.

Table 4 presents the evaluation of the number of queries made to the LLMs during the task execution. All methods were tested under the same underlying language model. The key observations are as follows: (i) In scenarios involving sequential order completion, DeMAC demonstrates query efficiency in sequential tasks, suggesting its ability to learn efficient strategies for subsequent tasks. (ii) DeMAC requires substantially fewer LLM queries compared to Proagent and CAC, which highlights its efficiency in communication and decision-making. (iii) In layouts with higher uncertainty, the number of queries made to the LLMs by all compared methods increases, but is still the lowest by DeMAC.

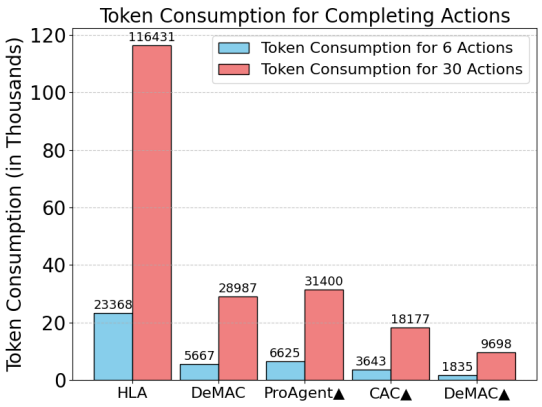


Figure 4: Comparison of token consumption (in thousands) across different methods for 6 and 30 actions. ▲ indicates tests conducted in the classic environment; others tested in the extended environment.

4.3 Ablation Studies

The ablation study in the Extended Environment [8] (Ring Layout, 30 trials) evaluated DeMAC's key components, as shown in Table 5. Removing DAG Info (Variant 2) increases steps and lowers the score, showing its role in structured task dependencies. Without Environment Perception (Variant 4), the score drops further, highlighting its importance in adaptability. Variant 3 (No Player-Level

[8] For ablation experiments on different layouts, please refer to App. E.1

Table 6: Uncertainty-oriented comparison on two layouts (mean ± std over 10 episodes; lower is better).

Layout / Method	Avg calls/replan	Replan times	Plan queries	Conflict ratio
Cramped Room				
DeMAC	$\mathbf{1.486 \pm 0.241}$	$\mathbf{7.300 \pm 1.567}$	$\mathbf{28.100 \pm 2.767}$	$\mathbf{20.62}$
ProAgent	3.477 ± 0.290	48.500 ± 7.276	104.700 ± 10.531	31.66
Counter Circuit				
DeMAC	$\mathbf{1.312 \pm 0.198}$	$\mathbf{7.400 \pm 1.625}$	$\mathbf{26.700 \pm 5.851}$	$\mathbf{21.70}$
ProAgent	3.194 ± 0.161	35.800 ± 4.450	108.300 ± 10.264	24.84

Feedback) results in a notable decline in success and efficiency, but the most severe impact is seen in Variant 6, where success rate plummets to 24.7%, steps surge (198.7 First Order, 324.1 Second Order), and the score drops to 42.14. Among all components, Player-Level Feedback & Reflection Loop is the most crucial, as its absence causes the sharpest decline in both success rate and efficiency, proving essential for coordination and overall task performance. To further examine DeMAC's scalability, we extend our test to a 3-agent setup (see App. E.2).

4.4 Uncertainty-Oriented Evaluation

To provide quantitative evidence of how DeMAC handles execution uncertainty, we report a set of *proxy indicators* that reflect conflict frequency and repair effort during online planning. We follow the evaluation protocol of ProAgent: each layout is run for 10 episodes with 400 time steps per episode under matched inference settings.

Metrics. We measure four interpretable indicators (lower is better):

- **Avg. calls per replan** : average #LLM calls made when a detected conflict triggers re-planning; smaller values indicate lower repair cost per inconsistency.
- **Replan times** : total #re-planning events; fewer events suggest stronger robustness of the initial plan.
- **Plan queries** : #LLM queries during normal planning (no conflicts); reflects baseline planning effort.
- **Conflict ratio** (%) : $\frac{\#\text{Replans}}{\#\text{Replans}+\#\text{PlanQueries}} \times 100\%$. A smaller ratio means the agent spends more time in stable execution than in recovery.

Across both layouts, DeMAC consistently lowers the *repair effort* and *instability*: (1) Avg. calls per replan drop by ∼57% (1.486 vs. 3.477) on *Cramped Room* and ∼59% (1.312 vs. 3.194) on *Counter Circuit*; (2) replan times reduce by ∼85% (7.3 vs. 48.5) and ∼79% (7.4 vs. 35.8), respectively; (3) plan queries decrease by ∼73% and ∼75%; (4) conflict ratio decreases from 31.66% to 20.62% and from 24.84% to 21.70%. These improvements indicate that DeMAC maintains more stable execution under identical sources of uncertainty (layout geometry, object randomness, and agent initialization), requiring fewer conflict-driven repairs and less LLM effort.

We view these indicators as practical surrogates for uncertainty reduction: the DAG-aware planner reduces conflict incidence by enforcing long-horizon dependencies; the manager–player feedback lowers ambiguity in local decisions; and selective retrieval curbs unnecessary planning queries.

5 Conclusion

This paper proposes the DeMAC Framework that integrates long-term planning into LLM-based multi-agent systems for collaborative tasks. By leveraging dynamic DAG structures for task decomposition and harmonizing actions, DeMAC enables agents to maintain coherence amidst intricate dependencies and environmental uncertainties. Experiments conducted in the Overcooked simulation environment show significant improvements in traditional reinforcement learning methods and human-agent collaboration in terms of game score, success rate and task completion efficiency. Future work includes extending DeMAC to more diverse and realistic environments and exploring its applicability in real-world scenarios that require sophisticated agent coordination and long-term planning.

Limitations

Multimodal Information. In physical simulation tasks, successful task execution often depends on the perception of visual, spatial layouts and temporal sequences, rather than text information alone.

However, the DeMAC framework primarily relies on language models for reasoning, lacking the ability to process multimodal information such as visual and spatial data. This limitation becomes particularly evident in tasks involving complex path planning, object localization, and visual feedback. Future work could integrate visual-language multimodal models into the framework to enhance agents' comprehensive perception of environmental details, thereby improving their adaptability to multimodal tasks.

Quantification of Uncertainty Reduction. While the framework effectively reduces uncertainty in task planning and execution by leveraging Dynamic DAG and environment perception, we have not been able to develop a direct, precise method to measure the reduction of uncertainty in a quantifiable manner. Instead, we rely on indirect indicators, such as evaluating game scores or using ablation experiments to assess the contribution of each component of the DeMAC framework to overall performance improvement. While these evaluations demonstrate the effectiveness of the framework and its components, they do not provide a clear, numerical measure of the exact uncertainty reduction achieved in the environment. Future work could explore more direct methods to quantify the reduction of uncertainty, potentially providing deeper insights into the framework's performance and its impact on decision-making under uncertainty.

Generalization to Diverse Multi-Agent Environments. While DeMAC demonstrates strong performance in Overcooked-style cooperative tasks, its applicability to broader multi-agent environments remains unexplored. Scenarios involving heterogeneous agents, competitive dynamics, or decentralized coordination may introduce challenges that differ fundamentally from structured cooperative cooking tasks. Evaluating DeMAC in such settings—e.g., mixed-motive tasks, resource contention, or open-world environments—would provide stronger evidence of its generality and robustness across diverse multi-agent applications.

Acknowledgments

We thank the anonymous reviewers and the area chair for their constructive feedback. This work was jointly conducted by fixed members of the laboratory and their collaborators. The authors were supported in part by the Key Laboratory of Computing Power Network and Information Security, Ministry of Education, Qilu University of Technology (Shandong Academy of Sciences) through the Laboratory Open Project under Grant No. 2023PY001.

References

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

Saaket Agashe, Yue Fan, Anthony Reyna, and Xin Eric Wang. 2024. Llm-coordination: Evaluating and analyzing multi-agent coordination abilities in large language models. *Preprint*, arXiv:2310.03903.

Jeffrey M Bradshaw, Paul J Feltovich, and Matthew Johnson. 2017. Human–agent interaction. In *The handbook of human-machine interaction*, pages 283–300. CRC Press.

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. In *Advances in Neural Information Processing Systems*.

Chi-Min Chan, Weize Chen, Yusheng Su, Jianxuan Yu, Wei Xue, Shanghang Zhang, Jie Fu, and Zhiyuan Liu. 2024. Chateval: Towards better LLM-based evaluators through multi-agent debate. In *The Twelfth International Conference on Learning Representations*.

Guangyao Chen, Siwei Dong, Yu Shu, Ge Zhang, Jaward Sesay, Börje Karlsson, Jie Fu, and Yemin Shi. 2024a. Autoagents: A framework for automatic agent generation. In *Proceedings of the Thirty-Third International Joint Conference on Artificial Intelligence, IJCAI-24*, pages 22–30. International Joint Conferences on Artificial Intelligence Organization. Main Track.

Weize Chen, Yusheng Su, Jingwei Zuo, Cheng Yang, Chenfei Yuan, Chi-Min Chan, Heyang Yu, Yaxi Lu, Yi-Hsin Hung, Chen Qian, Yujia Qin, Xin Cong, Ruobing Xie, Zhiyuan Liu, Maosong Sun, and Jie Zhou. 2024b. Agentverse: Facilitating multi-agent collaboration and exploring emergent behaviors. In *The Twelfth International Conference on Learning Representations*.

Weize Chen, Chenfei Yuan, Jiarui Yuan, Yusheng Su, Chen Qian, Cheng Yang, Ruobing Xie, Zhiyuan Liu, and Maosong Sun. 2024c. Beyond natural language: Llms leveraging alternative formats for enhanced reasoning and communication. *arXiv preprint arXiv:2402.18439*.

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*.

Jinhao Duan, Renming Zhang, James Diffenderfer, Bhavya Kailkhura, Lichao Sun, Elias Stengel-Eskin, Mohit Bansal, Tianlong Chen, and Kaidi Xu. 2024. Gtbench: Uncovering the strategic reasoning limitations of llms via game-theoretic evaluations. *Preprint*, arXiv:2402.12348.

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*.

Eithan Ephrati and Jeffrey S Rosenschein. 1994. Divide and conquer in multi-agent planning. In *AAAI*, volume 1, page 80.

Matthew Fontaine, Ya-Chuan Hsu, Yulun Zhang, Bryon Tjanaka, and Stefanos Nikolaidis. 2021. On the Importance of Environments in Human-Robot Coordination. In *Proceedings of Robotics: Science and Systems*, Virtual.

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

Hengyuan Hu and Jakob N Foerster. 2020. Simplified action decoder for deep multi-agent reinforcement learning. In *International Conference on Learning Representations*.

Wenlong Huang, Fei Xia, Dhruv Shah, Danny Driess, Andy Zeng, Yao Lu, Pete Florence, Igor Mordatch, Sergey Levine, Karol Hausman, et al. 2024. Grounded decoding: Guiding text generation with grounded models for embodied agents. *Advances in Neural Information Processing Systems*, 36.

Max Jaderberg, Valentin Dalibard, Simon Osindero, Wojciech M Czarnecki, Jeff Donahue, Ali Razavi, Oriol Vinyals, Tim Green, Iain Dunning, Karen Simonyan, et al. 2017. Population based training of neural networks. *arXiv preprint arXiv:1711.09846*.

Sehoon Kim, Suhong Moon, Ryan Tabrizi, Nicholas Lee, Michael W. Mahoney, Kurt Keutzer, and Amir Gholami. 2024. An llm compiler for parallel function calling. In *Proceedings of the International Conference on Machine Learning (ICML)*.

Paul Knott, Micah Carroll, Sam Devlin, Kamil Ciosek, Katja Hofmann, Anca Dragan, and Rohin Shah. 2021. Evaluating the robustness of collaborative agents. In *Proceedings of the 20th International Conference on Autonomous Agents and MultiAgent Systems*, AAMAS '21, page 1560–1562, Richland, SC. International Foundation for Autonomous Agents and Multiagent Systems.

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

Huao Li, Yu Chong, Simon Stepputtis, Joseph Campbell, Dana Hughes, Charles Lewis, and Katia Sycara. 2023b. Theory of mind for multi-agent collaboration via large language models. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 180–192, Singapore. Association for Computational Linguistics.

Xiaoqi Li, Mingxu Zhang, Yiran Geng, Haoran Geng, Yuxing Long, Yan Shen, Renrui Zhang, Jiaming Liu, and Hao Dong. 2024. Manipllm: Embodied multimodal large language model for object-centric robotic manipulation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 18061–18070.

Yang Li, Shao Zhang, Jichen Sun, Yali Du, Ying Wen, Xinbing Wang, and Wei Pan. 2023c. Cooperative open-ended learning framework for zero-shot coordination. In *International Conference on Machine Learning*, pages 20470–20484. PMLR.

Jijia Liu, Chao Yu, Jiaxuan Gao, Yuqing Xie, Qingmin Liao, Yi Wu, and Yu Wang. 2024. Llm-powered hierarchical language agent for real-time human-ai coordination. In *Proceedings of the 23rd International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2024)*, pages 1219–1228.

Xuetao Ma, Wenbin Jiang, and Hua Huang. 2025. Problem-solving logic guided curriculum in-context learning for llms complex reasoning. *arXiv preprint arXiv:2502.15401*.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, et al. 2015. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533.

Kolby Nottingham, Prithviraj Ammanabrolu, Alane Suhr, Yejin Choi, Hannaneh Hajishirzi, Sameer Singh, and Roy Fox. 2023. Do embodied agents dream of pixelated sheep: Embodied decision making using language guided world modelling. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 26311–26325.

Joon Sung Park, Joseph C O'Brien, Carrie J 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 (UIST)*, pages 2–22.

Chen Qian, Wei Liu, Hongzhang Liu, Nuo Chen, Yufan Dang, Jiahao Li, Cheng Yang, Weize Chen, Yusheng

Su, Xin Cong, et al. 2024. Chatdev: Communicative agents for software development. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 15174–15186.

Tabish Rashid, Mikayel Samvelyan, Christian Schroeder De Witt, Gregory Farquhar, Jakob Foerster, and Shimon Whiteson. 2020. Monotonic value function factorisation for deep multi-agent reinforcement learning. *Journal of Machine Learning Research*, 21(178):1–51.

Bidipta Sarkar, Aditi Talati, Andy Shih, and Dorsa Sadigh. 2022. Pantheonrl: A marl library for dynamic training interactions. In *Proceedings of the 36th AAAI Conference on Artificial Intelligence, Demo Track (AAAI Demo Track)*.

DJ Strouse, Kevin McKee, Matt Botvinick, Edward Hughes, and Richard Everett. 2021. Collaborating with humans without human data. *Advances in Neural Information Processing Systems*, 34:14502–14515.

Richard S Sutton. 2018. Reinforcement learning: An introduction. *A Bradford Book*.

Weihao Tan, Wentao Zhang, Shanqi Liu, Longtao Zheng, Xinrun Wang, and Bo An. 2024. True knowledge comes from practice: Aligning large language models with embodied environments via reinforcement learning. In *Proceedings of the International Conference on Learning Representations (ICLR)*.

Gerald Tesauro et al. 1995. Temporal difference learning and td-gammon. *Communications of the ACM*, 38(3):58–68.

Khanh-Tung Tran, Dung Dao, Minh-Duong Nguyen, Quoc-Viet Pham, Barry O'Sullivan, and Hoang D. Nguyen. 2025. Multi-agent collaboration mechanisms: A survey of llms. *ArXiv*, abs/2501.06322.

Wiebe van der Hoek, Wojciech Jamroga, and Michael Wooldridge. 2005. A logic for strategic reasoning. In *Proceedings of the Fourth International Joint Conference on Autonomous Agents and Multiagent Systems*, AAMAS '05, page 157–164, New York, NY, USA. Association for Computing Machinery.

Wiebe Van der Hoek and Michael Wooldridge. 2008. Multi-agent systems. *Foundations of Artificial Intelligence*, 3:887–928.

Lei Wang, Chen Ma, Xueyang Feng, Zeyu Zhang, Hao Yang, Jingsen Zhang, Zhiyuan Chen, Jiakai Tang, Xu Chen, Yankai Lin, et al. 2024a. A survey on large language model based autonomous agents. *Frontiers of Computer Science*, 18(6):186345.

Qineng Wang, Zihao Wang, Ying Su, Hanghang Tong, and Yangqiu Song. 2024b. Rethinking the bounds of LLM reasoning: Are multi-agent discussions the key? In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 6106–6131, Bangkok, Thailand. Association for Computational Linguistics.

Zhenhailong Wang, Shaoguang Mao, Wenshan Wu, Tao Ge, Furu Wei, and Heng Ji. 2024c. Unleashing the emergent cognitive synergy in large language models: A task-solving agent through multi-persona self-collaboration. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 257–279, Mexico City, Mexico. Association for Computational Linguistics.

Zihao Wang, Shaofei Cai, Guanzhou Chen, Anji Liu, Xiaojian Ma, and Yitao Liang. 2023. Describe, explain, plan and select: Interactive planning with LLMs enables open-world multi-task agents. In *Thirty-seventh Conference on Neural Information Processing Systems*.

Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, Ahmed Hassan Awadallah, Ryen W White, Doug Burger, and Chi Wang. 2024. Autogen: Enabling next-gen LLM applications via multi-agent conversation.

Sarah A. Wu, Rose E. Wang, James A. Evans, Joshua B. Tenenbaum, David C. Parkes, and Max Kleiman-Weiner. 2021. Too many cooks: Bayesian inference for coordinating multi-agent collaboration. *Topics in Cognitive Science*, 13(2):414–432.

Xue Yan, Yan Song, Xinyu Cui, Filippos Christianos, Haifeng Zhang, David Henry Mguni, and Jun Wang. 2024. Ask more, know better: Reinforce-learned prompt questions for decision making with large language models. *Preprint*, arXiv:2310.18127.

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*.

Chao Yu, Akash Velu, Eugene Vinitsky, Jiaxuan Gao, Yu Wang, Alexandre Bayen, and Yi Wu. 2022. The surprising effectiveness of ppo in cooperative multi-agent games. *Advances in Neural Information Processing Systems*, 35:24611–24624.

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.

Hancheng Zhang, Guozheng Li, Chi Harold Liu, Guoren Wang, and Jian Tang. 2023. Himacmic: Hierarchical multi-agent deep reinforcement learning with dynamic asynchronous macro strategy. In *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 3239–3248.

Hongxin Zhang, Weihua Du, Jiaming Shan, Qinhong Zhou, Yilun Du, Joshua B. Tenenbaum, Tianmin Shu, and Chuang Gan. 2024b. Building cooperative embodied agents modularly with large language models. *Preprint*, arXiv:2307.02485.

Yadong Zhang, Shaoguang Mao, Tao Ge, Xun Wang, Adrian de Wynter, Yan Xia, Wenshan Wu, Ting Song, Man Lan, and Furu Wei. 2024c. Llm as a mastermind: A survey of strategic reasoning with large language models. *arXiv preprint arXiv:2404.01230*.

Yang Zhang, Shixin Yang, et al. 2024d. Towards efficient llm grounding for embodied multi-agent collaboration. *arXiv preprint arXiv:2405.14314*.

Rui Zhao, Jinming Song, Yufeng Yuan, Haifeng Hu, Yang Gao, Yi Wu, Zhongqian Sun, and Wei Yang. 2023. Maximum entropy population-based training for zero-shot human-ai coordination. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, pages 6145–6153.

Kunlun Zhu, Hongyi Du, Zhaochen Hong, Xiaocheng Yang, Shuyi Guo, Zhe Wang, Zhenhailong Wang, Cheng Qian, Xiangru Tang, Heng Ji, and Jiaxuan You. 2025. Multiagentbench: Evaluating the collaboration and competition of llm agents. *Preprint*, arXiv:2503.01935.

Appendix

A Environment perception

Environment perception converts the obtained information into structured information output, which can then be used by decision-making and planning. We use Langchain's [9] output parser as the extractor.

A.1 Agent Profiles

The detailed prompting z_e of the Environmental Analysis Agent is shown in Figure 7. We also designed the corresponding output JSON schema, an example of which is shown in Figure 8.

To facilitate the understanding of environmental information by the LLM, we adopted a few-shot learning approach, providing the agent with several examples, as illustrated in the Figure 16 and Figure 17.

B Manager-Player Dual-Feedback

The Manager-Player Dual-Feedback module is a core part of the DeMAC framework, designed to ensure effective coordination.

B.1 Agent Profiles and Roles

Manager-Agent. It is responsible for aggregating the information provided by each Player-Agent, analyzing task dependencies, and making strategic decisions regarding task allocations. The Manager-Agent system prompt z_m is shown in Figure 9. The Manager-Agent prompt for inputting information is shown in Figure 10. Given the limitations of LLMs in interpreting grid-based spatial information, we choose to provide LLMs with simplified position and state information of each player. The input information received comes from Player-Agent and broadcast information. Mutable prompt components are marked in < >. We also provide an example of Manager-Agent output Figure 11.

Player-Agent. The Player-Agent focuses on providing detailed contextual information about its local environment rather than directly selecting actions. Its main role is to perceive its surroundings, identify nearby objects, assess feasible actions, and generate descriptive insights that can be used by the Manager-Agent. The Player-Agent system prompt z_p is shown in Figure 12. The Player-Agent prompt for inputting information is shown in Figure 13. Mutable prompt components are marked in < >.

[9] https://python.langchain.com

B.2 Reflection Loop

DAG Logic Detection. The **DAG Logic Detection** module is a crucial component in ensuring that task assignments adhere to logical execution sequences, respecting the dependencies and constraints of a dynamic environment. Within the **DeMAC** framework, each order and sub-task is represented as a node in a **DAG**, with directed edges defining dependencies between tasks. This structure guarantees that each action occurs in the correct sequence, preventing violations of logical task flow.

To facilitate effective collaboration and coordination, the DAG Logic Detection operates through the following steps:

B.2.1 Dependency Analysis

The **Manager-Agent** leverages the DAG to analyze which tasks are eligible for execution based on the current state of dependencies. Nodes corresponding to sub-tasks are only activated when all their prerequisite conditions are met. For example, a *Chop Action* must be completed before an *Assemble Action* or *Cook Action* can be initiated.

B.2.2 Consistency Check

After generating a plan for each **Player-Agent**, the **Manager-Agent** applies the DAG Logic Detection mechanism to validate that all dependencies are correctly maintained. For example, before an agent is assigned a *Cook Action*, the DAG ensures that all required ingredients have been chopped and are available. If any inconsistency is detected, it is flagged for correction.

B.2.3 Logic Detection and Error Handling

Once a plan is generated, it is checked against the DAG to identify logical inconsistencies. Consider the following example scenario:

Example: Error Detection in Task Sequencing Suppose two agents, **Player 1** and **Player 2**, must cooperate to prepare a *Lettuce Tomato Soup*. The DAG dependency sequence for this dish is structured as follows:

$$
\begin{array}{ccc}
\text{Fresh Lettuce} & \xrightarrow{\text{Chop}} & \text{Chopped Lettuce} \\
\text{Fresh Tomato} & \xrightarrow{\text{Chop}} & \text{Chopped Tomato} \\
& \downarrow \quad \downarrow & \\
\text{Lettuce Tomato} & \xrightarrow{\text{Assemble}} & \text{Lettuce Tomato} \\
& \xrightarrow{\text{Cook}} & \text{Cooked Lettuce Tomato} \\
& \xrightarrow{\text{Serve}} & \text{Complete Lettuce Tomato Soup}
\end{array}
\tag{7}
$$

Given the current environment state, the **Manager** produces the following long-term plan:

- **Step 1**: Player 1: Chop Lettuce, Player 2: Chop Tomato,

  Step 2: Player 1: Cook Lettuce Tomato, Player 2: Serve Lettuce Tomato Soup

Applying **Logic Detection** to this plan, the system identifies an inconsistency: The Cook Action should occur only after the Assemble Action. The Serve Action should occur only after the Cook Action.

If the environment already contains an *assembled Lettuce Tomato*, the error detection process refines its output: *"There is an error in Step 2. Player 2: The Serve action should occur only after the Cook action. Please re-plan."*

B.2.4 Adaptive Task Re-plan

If an inconsistency is detected, such as assigning a *Cook Action* before all required ingredients are ready, the Manager-Agent automatically revises the task assignments. The correction prompt provided to the Manager is illustrated in Figure 15, and an example of the logic correction process is shown in Figure 14. The Manager then generates an updated plan that adheres to the DAG constraints and redistributes tasks to the **Player-Agents**, ensuring a coherent workflow without redundant steps or unnecessary backtracking. The specific algorithm flow is shown in Algorithm 1.

This iterative validation and correction process ensures that the Manager's long-term strategy aligns with the dynamically evolving environment and DAG constraints, optimizing coordination and task execution efficiency.

C Dynamically DAG information

This component is responsible for capturing and representing the relationships between different tasks or sub-tasks within a dynamically changing environment.

Algorithm 1 Task Plan Refinement with Logical Consistency

Input: Initial plan with N steps, $max\_iter = 5$
Output: Refined and executable task plan
Step 1: Initialization
$iter \leftarrow 0$, $steps \leftarrow N$
Step 2: Iterative Refinement
while $iter \leq max\_iter$ **do**
  Generate candidate $task\_seq$ with $steps$ steps
  if $task\_seq$ satisfies DAG constraints **then**
    return $task\_seq$
  else
    Revise $task\_seq$ via Manager Agent
  end if
  $iter \leftarrow iter + 1$
end while
Step 3: Step Reduction
if no valid plan found **then**
  $steps \leftarrow steps - 1$
  if $steps > 1$ **then** retry planning
  elsegoto Step 4
  end if
end if
Step 4: Fallback Strategy
if still invalid **then**
  Execute independent tasks (e.g., *Chop Actions*)
  Resolve conflicts dynamically during execution
end if
Step 5: Execution
if plan valid **then** begin execution
end if

C.1 DAG information

The DAG is used to represent the interdependencies among various tasks assigned to agents. Each node in the DAG represents a specific task or subtask, while directed edges represent the dependencies between these tasks. The absence of cycles ensures that there is always a clear progression from one task to the next, making it possible to efficiently allocate resources and manage dependencies without creating conflicts or deadlocks. Figures 16 and 17 display portions of the DAGs.

C.2 Depth and Breadth of the DAG

Depth of the DAG. The depth of the DAG is defined as the longest path length from the starting node to the final node. This metric corresponds

to the critical path of the task execution process, representing the minimum number of sequential steps required to complete the workflow.

Example: Calculating DAG Depth. Consider a workflow where the task starts with Fresh Lettuce and concludes with *Lettuce Tomato Soup*. The state transitions are as follows:

$$\begin{aligned} \text{Fresh Lettuce} &\xrightarrow{\text{Chop Lettuce}} \text{Chopped Lettuce} \\ &\xrightarrow{\text{Assemble}} \text{Lettuce Tomato} \\ &\xrightarrow{\text{Cook}} \text{Lettuce Tomato Soup} \end{aligned} \quad (8)$$

The longest sequence of dependencies consists of three sequential steps:

1. Chop Lettuce
2. Assemble Lettuce and Tomato
3. Cook Lettuce Tomato Soup

Thus, the depth of the DAG is 3, indicating that at least three sequential actions are required to complete this task.

Breadth of the DAG. The breadth of the DAG refers to the number of tasks that can be executed in parallel at the same hierarchical level. A wider breadth indicates a higher degree of parallelism, meaning that multiple tasks can be processed concurrently, thereby increasing system efficiency.

Example: Calculating DAG Breadth. At a certain level of the DAG, consider the following nodes: Node3 (Chopped Lettuce), Node4 (Chopped Tomato).

Since both chopping actions can occur independently, they belong to the same level in the DAG. The breadth at this level is 2, as two operations can be executed simultaneously. A higher breadth enhances efficiency by allowing parallel execution, reducing total processing time.

D Experimental Settings

In a multi-agent system, the size of state space directly affects the complexity and computational requirements of the system.

We test DeMAC and other methods on two different Overcooked environments. Overcooked environment is a grid game where each agent's goal is to collaborate, create soup, and deliver it to the designated delivery location.

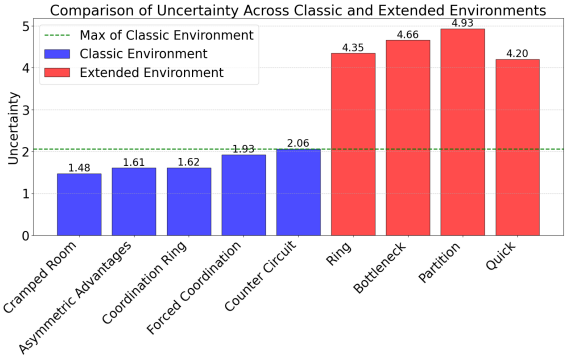


Figure 5: Comparison of uncertainty between the Classic and Extended environments.

D.1 Uncertainty in Multi-Agent Systems

The uncertainty in multi-agent systems arises from a variety of spatial and task-related factors, including bottlenecks in the map, resource distribution dispersion, task dependency complexity, and agent path conflicts. To quantify the overall uncertainty, we propose a metric U, which integrates these factors into a unified representation.

The total uncertainty U is computed as a weighted sum of individual components:

$$\begin{aligned} U = &w_1 \cdot B_{\text{rate}} + w_2 \cdot C_{\text{rate}} + w_3 \cdot \frac{1}{1 + R_{\text{std}}} \\ &+ w_4 \cdot P_{\text{avg}} + w_5 \cdot T_{\text{dep}} + w_6 \cdot D_{\text{graph}} \end{aligned} \quad (9)$$

where each term corresponds to a specific aspect of uncertainty in the environment. B_{rate} represents the bottleneck rate, defined as the proportion of nodes with only one connection:

$$B_{\text{rate}} = \frac{|\mathcal{B}|}{|\mathcal{V}|} \quad (10)$$

where $|\mathcal{B}|$ is the number of bottleneck nodes, and $|\mathcal{V}|$ is the total number of nodes. C_{rate} quantifies path conflicts, defined as the fraction of edges traversed by multiple agents:

$$C_{\text{rate}} = \frac{|\mathcal{E}_{\text{conflict}}|}{|\mathcal{E}|} \quad (11)$$

where $|\mathcal{E}_{\text{conflict}}|$ is the number of conflicting edges, and $|\mathcal{E}|$ is the total number of edges in the graph.

The dispersion of resources, denoted as R_{std}, is introduced to quantify how spatially scattered the resources are within the environment. Intuitively, this measure reflects the degree to which individual resource positions deviate from their collective center. A larger value of R_{std} indicates that resources

are widely spread out, implying higher spatial heterogeneity and potentially greater difficulty for agents to coordinate. Conversely, a smaller value means that resources are concentrated in a compact region, which usually facilitates easier coordination but may also increase competition. Formally, R_{std} is computed as the standard deviation of the pairwise distances among resources:

$$R_{\text{std}} = \sqrt{\frac{1}{|\mathcal{R}|} \sum_{i=1}^{|\mathcal{R}|} \|r_i - \bar{r}\|^2} \tag{12}$$

where $\mathcal{R}$ is the set of resources, r_i is the position of resource i, and $\bar{r}$ denotes their centroid.

The average path length, P_{avg}, reflects the mean shortest path between nodes:

$$P_{\text{avg}} = \frac{1}{|\mathcal{V}|(|\mathcal{V}|-1)} \sum_{u \neq v} d(u, v) \tag{13}$$

where $d(u, v)$ is the shortest path distance between nodes u and v.

Task dependency complexity, T_{dep}, accounts for the sequential relationships among tasks, calculated as:

$$T_{\text{dep}} = \sum_{t \in \mathcal{T}} |\text{Pre}(t)| \tag{14}$$

where $\mathcal{T}$ is the set of tasks, and $\text{Pre}(t)$ represents the predecessor tasks of t.

Finally, D_{graph} measures the maximum shortest path in the graph:

$$D_{\text{graph}} = \max_{u,v \in \mathcal{V}} d(u, v) \tag{15}$$

The weights $w_1, w_2, w_3, w_4, w_5, w_6$ are non-negative and normalized such that:

$$\sum_{i=1}^{6} w_i = 1 \tag{16}$$

These weights can be adjusted based on the specific scenario to reflect the relative importance of each component.

To analyze the impact of environmental complexity on uncertainty, we applied this metric to two distinct environments: Classic Environment and Extended Environment. The results, shown in Figure 5, reveal that Extended Environment exhibits significantly higher uncertainty due to increased bottleneck rate and task dependency complexity. This demonstrates the utility of the metric in capturing both spatial and task-related complexities. The decision to use the same uncertainty calculation method across both environments is driven by the fundamentally similar uncertainty-related challenges they present. Despite differences, both environments share key challenges such as layout similarity, where tasks are represented as interconnected DAGs with specific dependencies. Additionally, task dependencies in both environments cause uncertainties, as one task cannot proceed until another is completed. Agent blocking is another common challenge, where one agent's inability to complete a task can delay or restrict the progress of other agents. Given these shared uncertainties, a unified uncertainty calculation method ensures consistent handling across both environments.

D.2 LLMs

We employ the GPT-3.5-Turbo, GPT-4-Turbo, LLaMA, Qwen and GPT-4o, as the underlying language model to build our Agent framework. We uniformly set the temperature to 0.1 for all experiments.

D.3 Classic overcooked environment

The Classic overcooked environment has 5 different layouts: Cramped Room, Asymmetric Advantages, Coordination Ring, Forced Coordination and Counter Circuit. Each layout targets different problems[10].

Task Description. Players place 3 onions in a pot, leave them to cook for 20 timesteps, put the resulting soup in a dish, and serve it, giving all players a reward of 20 score. Agents need to plan and collaborate reasonably to achieve higher scores within the specified time steps.

Game Score. To evaluate the agents' performance, we used a scoring system where the agents' objective was to serve as many soups as possible within a time limit of 400 timesteps. To ensure robustness and avoid biases due to randomness, we performed 20 trials (in the study by (Carroll et al., 2019), where each algorithm was tested 5 times.) on each layout and reported the average score as the final evaluation metric for comparison with other methods.

[10]For the specific functions and settings of each layer in the Classic overcooked environment, please refer to (Carroll et al., 2019)

D.4 Extended overcooked environment

The Overcooked benchmark (Liu et al., 2024) incorporates the Overcooked game mechanics designed by (Wu et al., 2021), introducing some extensions that make the environment more challenging. Table2 is based on the experimental environment established by (Wu et al., 2021), featuring three distinct tasks (Tomato, Lettuce Salad and Lettuce-Tomato Salad), as illustrated in Figure 16. Additionally, three maps with unique strategic significance were designed to support these tasks.

In this extended version, agents are required to prepare a variety of soups, each with different ingredient requirements, while managing dynamic changes and additional constraints.

Dynamic Orders and Additional Challenges. Unlike the Classic environment, the extended environment presents agents with dynamic soup orders, which change continuously throughout the game. Each type of soup requires a different set of ingredients, and each order comes with a specific time limit[11]. Agents are rewarded for successfully completing an order within the given time, while missing the deadline results in penalties. The increased diversity of tasks, as well as the timing constraints, demand more sophisticated planning and quick adaptation from agents.

Task Description. Agents in the extended environment must place the correct ingredients in the pot, allow sufficient cooking time, and deliver the completed dish to the serving station. However, agents must also adapt to changing orders and prioritize tasks based on the time-sensitive nature of each order. Effective prioritization is crucial, as poorly timed actions may lead to unfinished or burnt soups, significantly impacting overall performance. This environment includes the risk of soup burning if left unattended in the pot for too long. Burnt soup not only results in a penalty but also requires agents to extinguish the fire and discard the burnt contents before resuming their tasks. At the same time, the success rate and Step tests of simple tasks (Same as the experiment set up in (Wu et al., 2021)) were also conducted. In the experiment comparing the success rates and steps for simple tasks, the methods in the original algorithm's implementation are limited to these fixed tasks and cannot complete the entire game process, which is why no game score comparison was made.

Game Score. Similar to the (Liu et al., 2024), the performance of agents in the extended environment is measured using a scoring system based on the number of successful soups delivered. Successful delivery of a soup yields a certain number of points, depending on the complexity of the recipe. Each trial consisted of 100 seconds, and we conducted 20 trials per layout to ensure statistical robustness (in the study by (Liu et al., 2024), where each algorithm was tested 15 times.). The average score from these trials was used to compare DeMAC with other methods.

D.5 Token Consumption

Token consumption refers to the total number of tokens utilized by an LLM to generate a sequence of actions for an agent.

Measurement Setup. To assess token consumption, we conducted experiments in both classic and extended Overcooked environments, comparing DeMAC to baselines such as ProAgent, CAC. To ensure fairness, the layout and LLMs parameters selected in each experiment are consistent. Each experiment ran for 15 times trials, and the average token consumption across all trials was calculated to ensure statistical robustness. Figure 4 illustrates the token consumption (in thousands) for 6 and 30 actions generated during various Overcooked tasks.

D.6 Number of Queries (NQ) to LLMs for Task Completion

The number of queries (NQ) metric measures the number of times an agent has to query the LLM to obtain a feasible plan for the task at hand.

Measurement Setup. To calculate the NQ metric, we monitored the number of requests made to the LLMs during task execution across various layouts. The Overcooked layouts used included Cramped Room, Forced Coordination, and Coordination Ring. Similar to token consumption, each layout was evaluated over 15 times trials, and the mean value was reported. Table 4 presents the average number of queries for task completion across the different Overcooked layouts.

D.7 Latency

Table 7 reports the macro action latency across representative planning frameworks. Compared to HLA, a recent hierarchical agent framework, DeMAC further improves latency while enabling

[11]The orders and score set in the experiments in the main text are consistent with those in the original paper (Wu et al., 2021; Liu et al., 2024).

Table 7: Comparison of Macro Action Latency across methods.

	SMOA	FMOA	HLA	**DeMAC**
Mac. Act. Latency (s)	4.16 (1.01)	2.30 (1.81)	1.07 (0.22)	**1.03** (0.43)

fine-grained task assignment via dependency-aware planning. Notably, SMOA and FMOA are ablated versions of HLA, where essential reasoning and coordination mechanisms are removed.

Despite issuing more model queries than HLA, DeMAC benefits from its ability to generate multiple macro actions for multiple agents within a single planning iteration. This increases planning throughput and significantly reduces the amortized latency per action. The results validate that the DeMAC structured, yet parallel planning strategy offers a more scalable and responsive solution for multi-agent coordination.

E Ablation study

E.1 Different layouts

As shown in Table 8, the ablation study reveals the distinct contributions of each core component in DeMAC. When DAG Info is removed, particularly in more complex layouts such as Partition and Bottleneck, we observe substantial performance drops—25.98% and 29.30% respectively. These declines suggest that DAG Info is essential for coordinating tasks and ensuring the correct execution order. Player-level Feedback & Reflection Loop also shows a considerable impact, especially in complex environments. The removal of this component leads to a 39.02% drop in Bottleneck and 30.80% in Partition, emphasizing the importance of continuous feedback for agent coordination. Env Perception has a secondary but still noticeable impact on performance. Its absence leads to moderate drops in environments like Bottleneck (23.37%) and Partition (17.41%), where environmental awareness is crucial for adapting to changing conditions and managing spatial tasks. This module also plays a key role in shielding DeMAC from low-level grid noise by converting spatial layouts into abstract, high-level cues, thus enabling more stable and focused decision-making.

Overall, the ablation results demonstrate that DAG Info and Player-level Feedback are the most critical components for task coordination and ensuring effective execution, especially in layouts that require handling complex task interdependencies and dynamic agent interactions. When all components are removed, as in Variant 6, the performance drops drastically by up to 79.18% in Partition and 78.04% in Bottleneck, reinforcing the conclusion that each of these components plays an integral role in maintaining high performance across different layouts. The combined absence of all these components leads to a severe decline in efficiency, highlighting their complementary functions in ensuring effective task execution in dynamic environments.

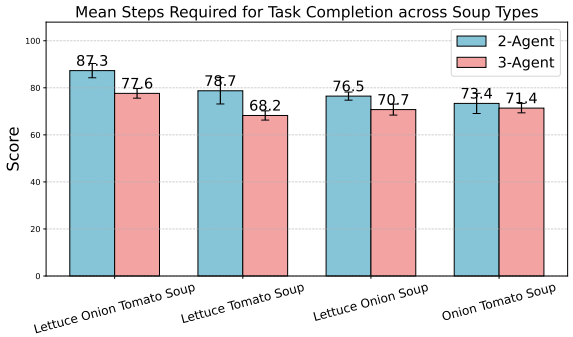


Figure 6: Comparison of average step and standard deviations across four soup types under 2-agent and 3-agent configurations.

Complexity-Conditioned Ablation. We quantify how module importance scales with task complexity by ordering layouts (Ring→Quick→Partition→Bottleneck) using our uncertainty/critical-path statistics and reporting relative score drops when ablating each module. The degradation monotonically increases with complexity; the Feedback Loop shows the largest sensitivity in constrained layouts, reflecting the need for runtime re-planning under uncertainty.

The steeper drops on Partition/Bottleneck indicate the symbolic dependency representation (DAG Info) and online correction (Feedback Loop) are increasingly critical as interleaving and contention grow. This connects Table 9 with Table 5's per-module results.

E.2 Different agents

We further investigate the impact of the number of agents on task efficiency by comparing the average number of steps required to complete various soup-making tasks under 2-agent and 3-agent configurations. As illustrated in Figure 6, increasing the number of agents consistently reduces the required steps across all task types, indicating improved collaboration efficiency.

Table 8: Ablation study on different Overcooked layouts.

DeMAC Variants	Env Perception	DAG Info	Player-level Feedback & Reflection Loop	Performance Scores			
				Ring *(4.35)*	**Partition** *(4.66)*	**Bottleneck** *(4.93)*	**Quick** *(4.20)*
Variant 1	✓	✓	✓	123.6	114.3	134.8	127.6
Variant 2	✓	✗	✓	106.3 ↓ 13.99%	84.6 ↓ 25.98%	95.3 ↓ 29.30%	104.7 ↓ 17.95%
Variant 3	✓	✓	✗	98.7 ↓ 20.15%	79.1 ↓ 30.80%	82.2 ↓ 39.02%	96.8 ↓ 24.14%
Variant 4	✗	✓	✓	110.9 ↓ 10.28%	94.4 ↓ 17.41%	103.3 ↓ 23.37%	116.4 ↓ 8.78%
Variant 5	✗	✗	✓	86.7 ↓ 29.85%	76.4 ↓ 33.16%	79.4 ↓ 41.10%	90.1 ↓ 29.39%
Variant 6	✗	✗	✗	42.1 ↓ 65.94%	23.8 ↓ 79.18%	29.6 ↓ 78.04%	62.4 ↓ 51.10%

Layout (uncertainty)	Env. Perception ↓	DAG Info ↓	Feedback Loop ↓
Ring (4.35)	35.36%	36.60%	43.04%
Quick (4.20)	29.75%	32.81%	37.62%
Partition (4.66)	43.25%	46.11%	54.99%
Bottleneck (4.93)	**47.50%**	**49.48%**	**58.53%**

Table 9: Average score drops when ablating each module across layouts with increasing complexity.

Specifically, in complex tasks such as *Lettuce Onion Tomato Soup*, the average step count drops from 87.3 (2-agent) to 77.6 (3-agent), demonstrating that a third agent can significantly ease coordination and parallelization under intricate dependencies. Similar trends are observed in other recipes, such as *Lettuce Tomato Soup* (78.7 $\rightarrow$ 68.2) and *Lettuce Onion Soup* (76.5 $\rightarrow$ 70.7), where the addition of an extra agent enhances the system's ability to divide labor and reduce redundancy.

Notably, the performance gain is more prominent in multi-ingredient recipes, where multiple chopping and assembly actions must be handled concurrently.

These results highlight the importance of adaptive agent collaboration in dynamic environments. While increasing the agent count generally leads to better task efficiency, the marginal gains depend heavily on task structure and action dependencies.

F Future Outlook

Looking ahead, there are several exciting opportunities to expand and apply the DeMAC framework. Two key directions for its future development are outlined below:

Extension to MAS with More Agents: While DeMAC has shown strong results with two agents, it is well-positioned to be extended to MAS involving more agents. With its dynamic task planning and coordination capabilities, DeMAC can be adapted to handle more complex environments where multiple agents must collaborate. The goal is to develop strategies for coordinating a larger number of agents, resolving conflicts, and ensuring effective communication among them. By scaling the framework, we aim to explore how it can efficiently manage task dependencies and optimize decision-making in a multi-agent context, paving the way for broader applications in collaborative AI environments.

Integration with Real-World Applications: Another promising direction for DeMAC is its integration into real-world applications. The framework's ability to adapt to changes in task dependencies and environmental conditions makes it a strong candidate for use in industries such as logistics, smart cities, and robotics. In the future, we plan to explore how DeMAC can be adapted to decision-making tasks in these fields. This includes examining how the framework can handle real-world constraints, process data, and improve coordination in dynamic, complex settings. Potential applications could range from automated scheduling to intelligent resource allocation and coordination across distributed systems.

Environmental Analysis Agent

```
You are a highly professional information extractor tasked with simplifying and
compressing key information from the Overcooked game environment for subsequent
planning agents.

----------------------------------------
Task:
1.Extract the following:
    1. Timestep: Current game timestep.
    2. Current Order: Details of the current order.
    3. Closest Coordinates: List the closest coordinates of each item to Player 1 and
       Player 2 from "Map_info".
    4. Action DAG Paths: Different action paths generated through directed independent
       graphs.
2.If the current order has ≤ 2 actions, include all actions for the next order.

Instructions:
#If an attribute's value is unknown, set it to null.
#Follow the exact format of the example provided.
#Keep the output concise and clear.
Important: Always refer to the examples to guide your response.
```

Figure 7: Environmental Analysis Agent system prompt.

JSON schema

```
available_actions: str = Field(
    description = "List the possible actions Player 1 and Player 2 can take separately
at the current timestep.")

timestep: Optional[int] = Field(
    default=None,
    description = "Current timestep number in the game.")

current_order: str = Field(
    description = "Details about the current active order.")

dag_orders_path: Optional[list] = Field(
    description = ‘‘‘
    1. Extract the first action list from "Current order Action paths" and place it in
the "First" dictionary.
    2. Extract remaining actions from "Current order Action paths" and place them as a
list in the second part of the output.
    3. Extract the first action list from "Next order Action paths" and place it in the
"Next order First" dictionary.
    4. Ensure the output follows the example format.
    5. If “Current order Action paths” has ≤ 2 actions, output all actions for the
“Next order Action paths” instead of just the “Next order First.”
    ’’’)

map_info: Optional[str] = Field(
    default=None,
    description = '''
    1. List distances from Player 1 and Player 2 to the items needed for the current
order.
    2. Extract relevant order and action information from "Map_info.“
    3. Provide the locations of Player 1, Player 2, and current items in natural
language, including details of current and next orders.
    4. Include any contents currently cooking in the pot.
    '''
)
```

Figure 8: JSON schema example.

Manager-Agent system prompt

You are the **Manager** in a simplified **Overcooked game** involving two players, **Player 1** and **Player 2**. Your role is to effectively direct their actions to maximize cooperation, player engagement, and scoring efficiency.

GAME SCENARIO:
You supervise Player 1 and Player 2 as they collaborate in a kitchen to prepare and serve different types of soup under time constraints. Your objective is to manage the players in a way that maximizes the total score through timely and accurate completion of orders.

GAME GUIDELINES:
1.Dependencies
- **Assemble Ingredients**:
  - Lettuce Onion Soup: ["Chop Lettuce", "Chop Onion"]
  - Lettuce Tomato Soup: ["Chop Lettuce", "Chop Tomato"]
  - Onion Tomato Soup: ["Chop Onion", "Chop Tomato"]
  - Lettuce Onion Tomato Soup: ["Chop Lettuce", "Chop Tomato", "Chop Onion"]
- **Cook Soup**:
  - Lettuce Onion Soup: ["Assemble Lettuce Onion Soup Ingredients"]
  - Lettuce Tomato Soup: ["Assemble Lettuce Tomato Soup Ingredients"]
  - Onion Tomato Soup: ["Assemble Onion Tomato Soup Ingredients"]
  - Lettuce Onion Tomato Soup: ["Assemble Lettuce Onion Tomato Soup Ingredients"]
- **Serve Soup**:
  - Lettuce Onion Soup: ["Cook Lettuce Onion Soup"]
  - Lettuce Tomato Soup: ["Cook Lettuce Tomato Soup"]

2.Error Handling:
- If Player X encounters an error during an action in Step n, select an action from the **"Next Order First"** list in the DAG action information.
- Only output the action itself in this format: PlayerX: Action. For example, instead of outputting "Player2: Next Order First: Chop Tomato", simply state "Player2: Chop Tomato".

3.Primary Actions:
- Prioritize actions listed in the **primary DAG** action information, particularly those under **"First"**.

4.Ingredient Management:
- After performing an **"Assemble"** action, the ingredients used must be **chopped again** before being used for any subsequent "Assemble" action.

5.Action Uniqueness:
- Avoid repeating non-chopping actions whenever possible.

6.Parallel Actions:
- Actions by **Player 1** and **Player 2** are executed in **parallel**.

7.Distinct Actions:
- Player 1 and Player 2 **cannot perform the same action simultaneously** (e.g., both players cannot "Chop Onion" in the same step).

8.Handling Uncertainty:
- If uncertain about which action to take next, proceed with the **"Next Order First"** action from the DAG.

9.Serve Action Priority:
- If there are **cooked items in the pot**, immediately perform the **Serve** action.

Figure 9: Manager-Agent system prompt.

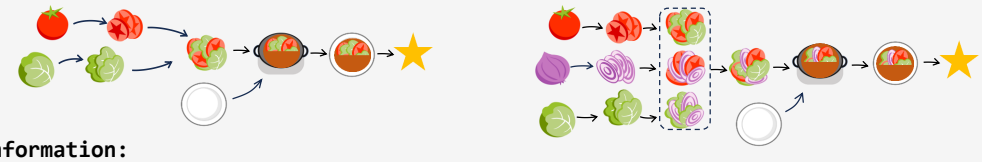
Manager-Agent input prompt

```
Evaluate the players' positions and current tasks to create a concise, coordinated
action plan for effective collaboration and timely completion of orders.
-------------------------------------------
INPUTS:
•Player's Info: <player_info>
•DAG Action Info: <dag_info>
•Available Actions: <available_actions>

GUIDELINES:
1.Prioritize Execution:
    •If the 'First' action in DAG Action Info is a "Serve", prioritize it.
    •Next Order First actions involving "Cook" should be executed first.
    •Prioritize actions in this order: Cook > Serve > Assemble > Chop.
    •Actions must be from DAG Action Info or Available Actions, prioritizing DAG Action
    Info.
    •Avoid both players performing the same action simultaneously.
    •Focus on Cook, Serve, and Assemble actions over Chop whenever possible.
2.Steps & Parallel Actions:
    •Limit actions to 3-4 steps.
    •Each step must include actions for Player 1 and Player 2.
    •Actions in each step are parallel, with no sequential order between players.
3.Response Format:
    •Thought: Provide reasoning for action choices and strategic considerations.
    •Steps: List actions for each step using the specified structure.

RESPONSE FORMAT:
use this format for clarity and easy parsing:
{
    "Thought": "Explain the rationale behind your chosen actions and any strategic
considerations.",
    "Steps": {
        "Step1": "Player1: action, Player2: action",
        "Step2": "Player1: action, Player2: action",
        "...": "..."
    }
}
```

Figure 10: Manager-Agent input prompt.

Orders: 

Input information:

•DAG Action Info: <[{'First': ['Chop Lettuce', 'Chop Tomato']}, ['Assemble Lettuce Tomato Soup Ingredients', 'Cook Lettuce Tomato Soup', 'Serve Lettuce Tomato Soup'], {'Next order First': ['Chop Onion', 'Chop Tomato', 'Chop Lettuce']}, ['Assemble Lettuce Onion Tomato Soup Ingredients', 'Cook Lettuce Onion Tomato Soup', 'Serve Lettuce Onion Tomato Soup']]>

Player's Info: <{"Items&Players":{
'Player1' : "Hold nothing, The one closest to Player1 is Fresh Tomato, followed by Fresh Onion. Firstly, I can go chop tomatoes, Player 2 may go chop Lettuce",
'Player2' : "Hold nothing, The one closest to Player2 is Fresh Lettuce, followed by Fresh Tomato. I can go chop Lettuce",
'Counter_Items': "None",
},
"Current order: Lettuce Tomato Soup, Next order: Lettuce Onion Tomato Soup, Third order: Lettuce Tomato Soup"}>

•Available Actions: <['Chop Tomato', 'Chop Lettuce', 'Chop Onion']>

Output:

Manager: 'Steps': {'Step1': 'Player1: Chop Lettuce, Player2: Chop Tomato', 'Step2': 'Player1: Assemble Lettuce Tomato Soup Ingredients, Player2: Chop Onion', 'Step3': 'Player1: Cook Lettuce Tomato Soup, Player2: Chop Lettuce', 'Step4': 'Player1: Serve Lettuce Tomato Soup, Player2: Chop Tomato'}

Figure 11: Manager-Agent Decision-Making Based on Environment and DAG Action Information (Example).

Player-Agent system prompt

You are **Player** in a simplified **Overcooked game** where you and other agent collaborate to complete various soup orders. Your role is to **observe your surroundings** and **provide essential information** to assist in effective planning and task execution by others.

GAME SCENARIO:

In this game, your primary role is not to execute actions directly, but to communicate your immediate environment details to help the team make better decisions.

SPECIFIC REPORTING GUIDELINES:

1.Surrounding Objects: List objects near you, including their **type** and **coordinates** (e.g., ingredients, vegetable, pot).
2.Important Changes: Inform if there are significant events in your area (e.g., a **burning pot** or **completed soup** ready for serving).
3.Team Coordination: Provide **updates** on your observations so that **task allocation** and **strategic decisions** can be optimized by the Manager-Agent.

Figure 12: Player-Agent system prompt.

Player-Agent input prompt

```
Current Scenario
•Environment Info: Includes kitchen stations, ingredients, orders, and task DAG: <
environment>.
•Player Info: <communication>.

GUIDELINES:
•Discuss next possible tasks considering:
    •Proximity to ingredients or items.
    •Order progress.
    •Interaction with other players' actions.
•Length: Be concise, max 20 words.

RESPONSE FORMAT:
Use the following for easy JSON parsing:
{
    "Nearby_items": "Crucial items near you for next steps.",
    "Communication": "Brief strategy or thoughts to other agents.",
    "Possible_tasks": "Tasks you can perform next, from 'available_actions' or
'dag_orders_path', considering position and order needs."
}
```

Figure 13: Player-Agent input prompt.

```
Input information:

Manager: 'Steps': {'Step1': 'Player1: Chop Lettuce, Player2: Chop Tomato', 'Step2':
'Player1: Assemble Lettuce Tomato Soup Ingredients, Player2: Chop Onion', 'Step3':
'Player1: Cook Lettuce Tomato Soup, Player2: Chop Tomato', 'Step4': 'Player1: Serve
Lettuce Tomato Soup, Player2: Assemble Lettuce Onion Tomato Soup Ingredients'}
--------------------------------
Logic detection result:
Step 4, Player2's action: The required ingredient 'Chop Lettuce' already used in Step 2
by Player1 needs to be chopped again before it can be used in step 4 by Player2's action
'Assemble Lettuce Onion Tomato Soup Ingredients".

Output:
Manager: 'Steps': {'Step1': 'Player1: Chop Lettuce, Player2: Chop Tomato', 'Step2':
'Player1: Assemble Lettuce Tomato Soup Ingredients, Player2: Chop Onion', 'Step3':
'Player1: Cook Lettuce Tomato Soup, Player2: Chop Lettuce', 'Step4': 'Player1: Serve
Lettuce Tomato Soup, Player2: Chop Tomato'}
```

At this step, there are not enough ingredients available.

Figure 14: Logic Detection and Correction Example.

Manager-Agent correct prompt

The previous sequence of steps contained logical errors. To ensure correctness and efficiency, please revise the steps to maintain logical consistency and promote efficient collaboration.

INPUTS:

- **Error Reason:** <reason>
- **Player's Info:** <player_info>
- **DAG Action Info:** <dag_info>
- **Available Actions:** <available_actions>

Re-evaluation Guidelines:

1.Violation Handling:

- If a violation occurs for Player X in Step n, choose an action from "Next Order First" in DAG action information (e.g., output 'Player2: Chop Tomato').
- If unable to resolve a violation, reduce the total number of steps in the sequence.

2.Prioritization:

- Prioritize actions in DAG action information (especially actions labeled 'First').
- **Minimize "Chop" Actions:** Unless "Chop" is specifically required by the DAG, avoid using it frequently.

New Steps Proposal:

Re-evaluate the provided information and generate a new, logically consistent action sequence using the format below to facilitate parsing:

```
{
    "Correction": "Describe improvements and the logical reasoning for the changes.",
    "Steps": {
        "Step1": "Player1: action, Player2: action",
        "Step2": "Player1: action, Player2: action",
        "...": "..."
    }
}
```

Figure 15: Manager-Agent Corrective Prompt.

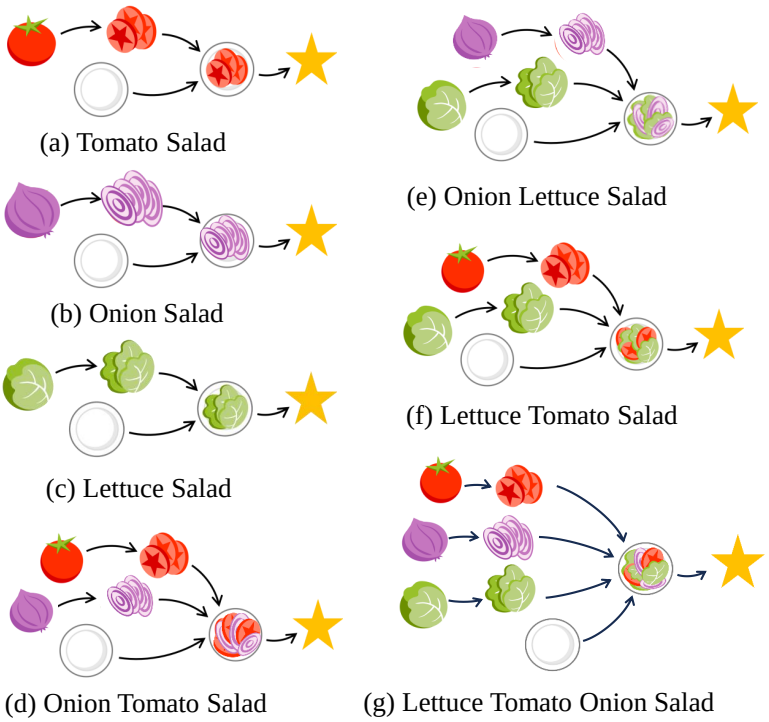


Figure 16: Partial dAG representations for salad recipes in Overcooked.

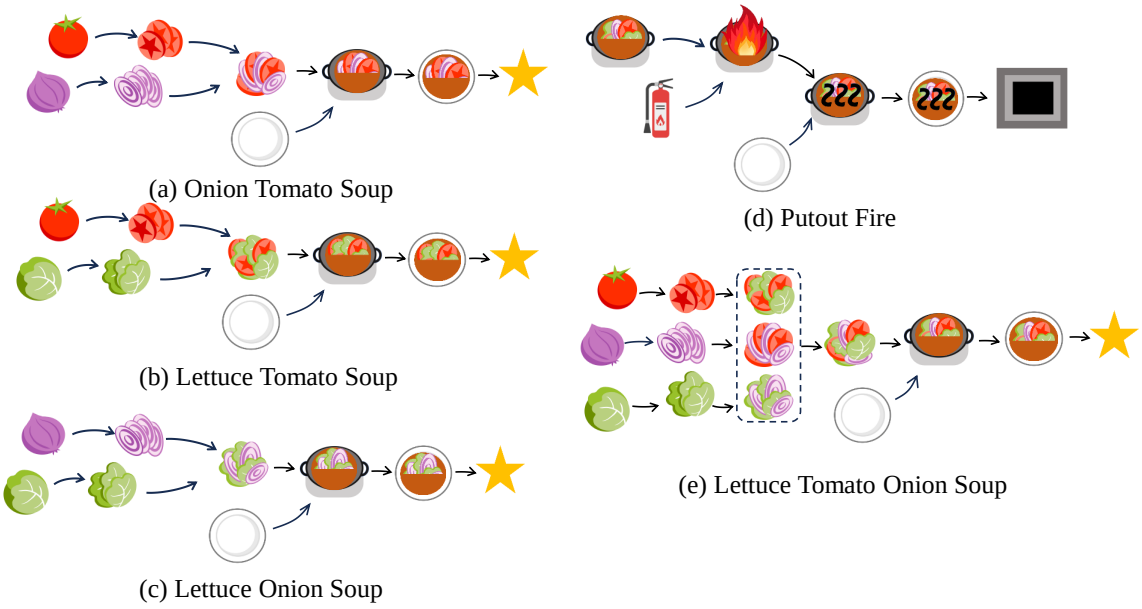


Figure 17: Partial DAG representations for soup recipes and other tasks in Overcooked.