ProAgent: Building Proactive Cooperative AI with Large Language Models

Ceyao Zhang[1,2,*†] **Kaijie Yang**[3,*] **Siyi Hu**[4,*]
Zihao Wang[2,5] **Guanghe Li**[2] **Yihang Sun**[2] **Cheng Zhang**[2] **Zhaowei Zhang**[2,5] **Anji Liu**[2,5]
Song-Chun Zhu[2,5] **Xiaojun Chang**[4] **Junge Zhang**[6]
Feng Yin[1] **Yitao Liang**[2] **Yaodong Yang**[2,‡]

[1] SSE&FNii, The Chinese University of Hong Kong, Shenzhen,
[2] Institute for Artificial Intelligence, Peking University,
[3] Monash University,
[4] ReLER, AAII, University of Technology Sydney,
[5] Beijing Institute for General Artificial Intelligence (BIGAI),
[6] Institute of Automation, Chinese Academy of Sciences

Abstract

Building AIs with adaptive behaviors in human-AI cooperation stands as a pivotal focus in AGI research. Current methods for developing cooperative agents predominantly rely on learning-based methods, where policy generalization heavily hinges on past interactions with specific teammates. These approaches constrain the agent's capacity to recalibrate its strategy when confronted with novel teammates. We propose **ProAgent**, a novel framework that harnesses large language models (LLMs) to fashion a *pro*active *agent* empowered with the ability to anticipate teammates' forthcoming decisions and formulate enhanced plans for itself. ProAgent excels at cooperative reasoning with the capacity to dynamically adapt its behavior to enhance collaborative efforts with teammates. Moreover, the ProAgent framework exhibits a high degree of modularity and interpretability, facilitating seamless integration to address a wide array of coordination scenarios. Experimental evaluations conducted within the framework of *Overcook-AI* unveil the remarkable performance superiority of ProAgent, attaining an average improvement of over 10% compared to the existing state-of-the-art methods. The advancement was consistently observed across diverse scenarios involving interactions with both AI agents of varying characteristics and human counterparts. These findings inspire future research for human-robot collaborations. For a hands-on demonstration, please visit https://pku-proagent.github.io.

Introduction

Large Language Models (LLMs) have rapidly emerged as powerful tools, achieving remarkable advancements across various domains, including long conversations (Ouyang et al. 2022), reasoning (Bubeck et al. 2023), and text generation (Brown et al. 2020). These models, by leveraging a vast amount of training data, can capture and embody a significant amount of common sense knowledge. Notable LLM-based agents like SayCan (Ahn et al. 2022), ReAct (Yao et al. 2022), DEPS (Wang et al. 2023), RAP (Hao et al. 2023), and Reflexion (Shinn et al. 2023) have demonstrated the ability to make decisions interactively through appropriate prompts or feedback. However, these works have primarily focused on exploring the potential of LLMs as individual agents, handling one task at a time. The untapped potential lies in investigating how LLMs can effectively cooperate with other agents, capitalizing on their robust reasoning and planning capabilities.

This research delves into the capabilities of LLMs in tackling the intricate challenges of multi-agent coordination (Yang and Wang 2020; Zhang, Yang, and Başar 2021; Gronauer and Diepold 2022), particularly in the realm of policy generalization (Strouse et al. 2021; Zhao et al. 2023; Li et al. 2023a,b). We present **ProAgent**, an innovative and adaptable framework specifically designed to excel in coordination scenarios alongside novel agents. ProAgent comprises three essential modules: `Planner`, `Verificator`, and `Memory`, along with the mechanism of `Belief Correction`. These modules synergistically enable ProAgent to actively predict teammates' intentions and achieve adaptive cooperative reasoning and planning without the need for prior training or finetuning.

To assess the adaptive cooperative capabilities of ProAgent, we conducted performance evaluations using the well-established multi-agent coordination testing suite, *Overcooked-AI* (Carroll et al. 2019). In this environment, two players must work together to maximize their score. The empirical findings from our evaluations reveal the following key insights: 1) ProAgent demonstrates remarkable proficiency in coordinating with various types of AI teammates across diverse scenarios. 2) ProAgent exhibits a notable preference for collaborating with rational teammates, such as the human proxy, which showcases human-like behavior and suggests its active effort in understanding teammates' intentions to enhance cooperation. 3) ProAgent achieves state-of-the-art performance in cooperation ability when compared to other AI agents and humans. These re-

[*]Equal contribution
[†]Work done when Ceyao Zhang visited Peking University.
[‡]Corresponding to: yaodong.yang@pku.edu.cn.

Preprint. Under review.

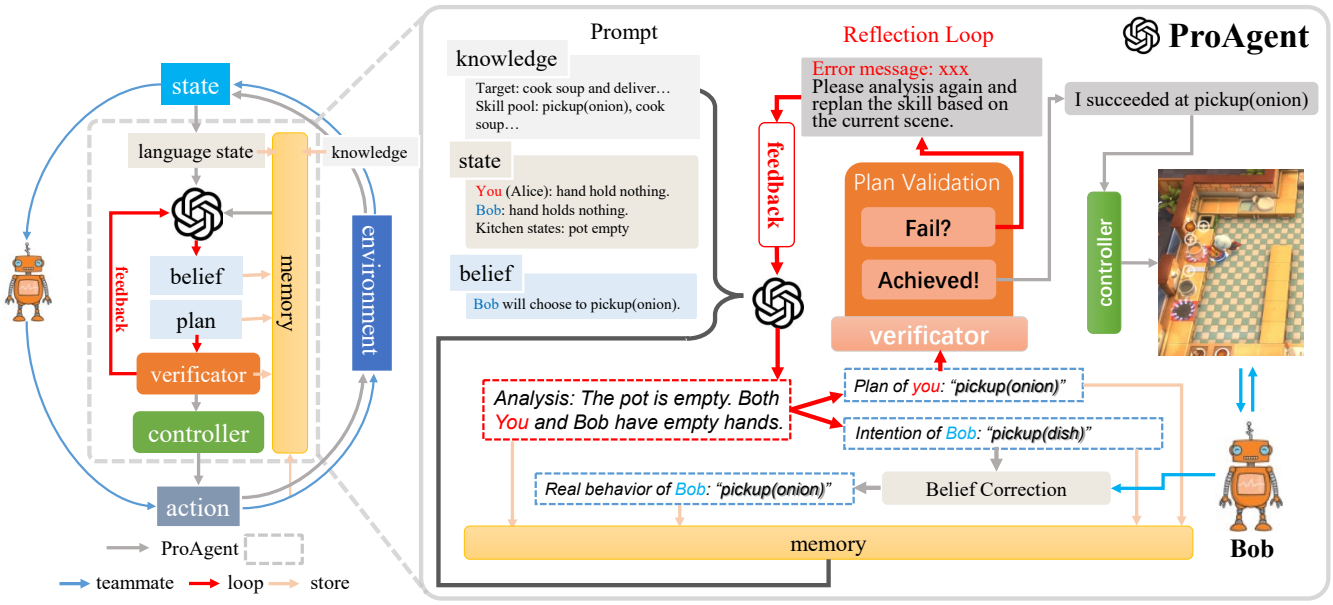


Figure 1: Overview of our proposed ProAgent framework including the coordination task workflow (**left**) and inner details of ProAgent pipeline (**right**). ProAgent commences its operation by translating the initial state into natural language. A large language model (`Planner`) adeptly analyzes the provided language state in conjunction with historical information stored in the `Memory`. This analytical process allows the model to discern the *intentions* of the teammate and devise a high-level *skill* for the agent accordingly. The predicted intention is validated through the `Belief Correction` mechanism, which involves comparing it with the ground truth behavior of the teammate agent. In case of skill failure, the `Verificator` is summoned to assess the skill's preconditions and provide a detailed explanation for the encountered issue. Should the need arise, ProAgent enters into a re-plan loop, initiating a recalibration process. On the other hand, if the skill is deemed viable, the `Controller` further dissects it into several executive actions, to be executed within the environment.

sults collectively highlight the effectiveness of ProAgent as a cooperative AI across a wide range of scenarios.

In summary, our work makes three key contributions: **Firstly**, we successfully integrate LLMs into the field of cooperative AI and propose the ProAgent framework, which serves as a comprehensive guideline for leveraging the powerful reasoning and planning capabilities of LLMs in cooperative settings. **Secondly**, we demonstrate the remarkable capability of our ProAgent to interpretably analyze the current scene, explicitly infer teammates' intentions and dynamically adapt its behavior accordingly. This proactive nature empowers ProAgent to actively collaborate with teammates, enabling more efficient cooperative scenarios. **Thirdly**, through a comprehensive series of experiments, we provide compelling evidence of ProAgent's superiority over other agents when engaging in cooperation with diverse types of teammates.

Related Works

Reasoning and Planning with Large Language Models. In the realm of LLMs (Huang and Chang 2022; Mialon et al. 2023; Bubeck et al. 2023), reasoning often entails decomposing intricate queries into sequential intermediate steps, referred to as Chain-of-Thought (CoT) (Wei et al. 2022; Kojima et al. 2022), to attain a final solution. Some research focuses on minimizing errors as the number of steps increases (Wang et al. 2022a), while others explore decomposition techniques that break down complex problems into simpler subproblems (Zhou et al. 2022). Recent endeavors have translated LLMs' reasoning capability into planning by constructing a monologue with feedback (Welleck et al. 2022; Shinn et al. 2023; Paul et al. 2023) to facilitate the reasoning and planning process. Notably, the challenge of open-ended long-term planning in the MineDojo environment (Fan et al. 2022) has been addressed by utilizing LLMs as central planners (Wang et al. 2023), thereby demonstrating the extensive capabilities of LLMs-based agents in overcoming complex decision-making tasks. However, none of the previously mentioned methods explicitly address the integration of reasoning and planning within the context of cooperative tasks, wherein the reasoning and planning capabilities of LLMs hold substantial potential.

Multi-agent Coordination. The goal of multi-agent coordination is to enable multiple autonomous agents to collaborate effectively towards a shared goal (Zhong et al. 2023; Rashid et al. 2018; Hu and Foerster 2019; Hu et al. 2021a; Yu et al. 2022). However, traditional approaches have limitations in fixed task settings and struggle to handle multiple tasks or unseen scenarios. One approach to address this challenge is to enable an agent to learn on multiple tasks concurrently (Hu et al. 2021b; Wen et al. 2022; Meng et al. 2021). However, these methods may still limit the agent's

cooperation ability in familiar tasks and fail to handle unseen tasks or new agent interactions. Another line of research focuses on zero-shot coordination (ZSC), utilizing Population-Based Training (PBT) (Strouse et al. 2021; Zhao et al. 2023; Lupu et al. 2021; Lucas and Allen 2022) and Theory of Mind (ToM) (Hu et al. 2021a; Wu et al. 2021; Wang et al. 2022b) to facilitate adaptive policy development for coordinating with various counterparts without prior coordination experience. However, these ZSC methods demand significant computational resources for data collection and model optimization, and the resulting policies often lack interpretability.

Method

The overview of our ProAgent framework, as is depicted in Fig. 1, involves constant interaction between agents and the environment. The interaction styles between the ProAgent and the teammate agent are different, where the teammate agent's decision-making loop is formed by the blue solid arrows in the outer circle, while the decision process of the ProAgent is formed by the middle gray dotted box and the outer gray solid arrows. The inference pipeline of ProAgent is a hierarchical process that involves multiple interactions between the LLMs and the task at hand. We break down the pipeline into five key stages:

Knowledge Library and State Alignment. The pipeline starts with acquiring `Knowledge Library` specific to the current task and transforming the raw state information into `Language-based State` description that the LLM can effectively comprehend.

Cooperative Reasoning and Planning. Receiving the aligned language-based state, the LLM then analyzes the current scene, infers the `Belief` about the teammate agent's intentions, and makes a skill `Plan` for the current agent.

Belief Correction and Skill Validation. The belief in the teammate agent's skill is further validated and corrected by the `Belief Correction` mechanism. Additionally, the selected skill is validated by the `Verificator` and repeatedly replanned until a legal skill is found.

Memory Storage. Throughout the pipeline, all relevant information involved in the prompt, reasoning process, and validation process is stored in the `Memory` module. This accumulated knowledge helps in making informed decisions and adjusting behavior over time.

Action Execution. Finally, a valid skill is selected, and the `Controller` module decomposes it into low-level actions, allowing ProAgent to effectively interact with the task or environment. The controller can be a rule-based, or hierarchical RL-based method.

We offer a pipeline outlined in pseudocode, available in the appendices, which delineates the procedural stages of ProAgent's cooperative reasoning and decision-making throughout the execution of a cooperative task.

By following this pipeline, ProAgent effectively incorporates a knowledge library, performs language-based reasoning, and adapts its behavior through continuous interaction and memory update.

Prompt Construction

Knowledge library The planning ability of LLMs is closely related to the prompt at the beginning, which is also the standard practice in automated planning. ProAgent is no exception, and the knowledge library should be fed into LLMs at the initial stage before the cooperation task begins. The main difficulty lies in how to build structured knowledge. In practice, we find that the best combination of knowledge library needs to be described from three perspectives, including `Task`, `Rules`, and `Demos`. We provide a template to construct the knowledge library:

```
### Task:
- The task requires two players player0 and player1 to
↪  work together as a team ...
- To get the points, the team needs to ...
...
### Rules:
In this task, each player can ONLY perform the
↪  following skills: [skill 1], [skill 2], ...
def skill_1(obj):
    [function detail]
def skill_2(obj, obj):
    [function detail]
...
Suppose you are an assistant who is proficient in the
↪  task. Your goal is to control player0 and cooperate
↪  with player1 who is controlled by a certain
↪  strategy in order to get a high score and should
↪  follow:
- [Rule 1].
- [Rule 2].
...
- Based on the current scene, you need to achieve
- [Target 1].
- [Target 2].
...
Your response should be in the following format:
- Analysis:[your analysis of the current scene]
- Plan for Player 0: [one skill in [skill 1, skill
↪  2...]]
...
### Demos:
Scene 0: [Player0 state 0]. [Player1 state 0]. [Other
↪  task information]
Analysis: Both Player0 and Player1 are [State
↪  description]. I guess two players will [Some
↪  skill].
[Target 1]: [Some skill].
[Target 2]: [Some skill].
...
Scene 39: [Player0 state 39]. [Player1 state 39].
↪  [Other task information]
Analysis: Player0 is [State description]. Player1 is
↪  [State description]. I guess ...
...
```

`Task` is for LLMs to understand the objective of the task and information about other cooperative agents. `Rules` is designed to regulate the planning pattern of the LLM, defining which skills are legal and which are not. In `Rules`, we can also enforce the format of LLMs' responses to follow the

CoT: output analysis and then plan according to the analysis instead of directly outputting a plan. `Demos` is an optional component of the three. Its main functionality is to provide real cases for LLMs to strengthen their memories and behave in accordance with the regulations set by `Rules`. Normally, `Demos` should contain a scene description followed by an analysis and the desired behavior, such as the selected skill. With these three parts, LLMs are able to understand the task and what is expected of them in the subsequent planning and reasoning stages.

State to language alignment In order to facilitate interaction between LLMs and the environment, it is essential to establish a bridge between the original symbolic state provided by the environment and the language-based state for LLMs. In most scenarios, the raw state is not directly applicable for LLMs' usage. Hence, finding an effective alignment between the original symbolic state and the language-based state is crucial to enhance LLMs' accurate understanding of the current situation. To illustrate this, we present a simplified example based on the Overcooked-AI environment, demonstrating how the state can be transformed into language within our ProAgent framework. With the knowledge library and initial state information prepared, ProAgent is equipped to tackle the cooperative task alongside its teammates. This marks the transition to the subsequent stage, where ProAgent engages in reasoning and planning, progressing step by step to achieve its objectives. Here is an illustrative instance of the alignment between state representation and natural language:

```
### Original State
        ___________________________________
        |X       X       P       X       X|
        |                                  |
        |O       →0o    ←1o           O|
        |                                  |
        |X                                 X|
        |                                  |
        |X       D       X       S       X|
        ___________________________________
### Language State (Layout)
Above is the layout of the kitchen: onion dispenser at
↪  (0, 1), onion dispenser at (4, 1), dish dispenser
↪  at (1, 3), pot at (2, 0), serving loc at (3, 3).
### Language State (Task state)
State: Player 0 holds one onion. Player 1 holds one
↪  onion. Kitchen states: Pot (2, 0) is empty.
```

Cooperative Reasoning and Planning

ProAgent is a specialized system tailored for cooperative tasks, where information from teammate agents plays a pivotal role in the coordination process. Existing works mainly utilize information in two ways: firstly, through explicit incorporation, involving communication and exchange of information before decision-making; secondly, through implicit modeling of teammate agents to facilitate cooperative learning. Each approach comes with its own set of advantages and disadvantages concerning cooperative reasoning and planning: The integration of teammate information can be achieved efficiently by sending teammate agent information to LLMs. However, this approach may jeopardize the overall generalization of ProAgent's reasoning capabilities. On the other hand, modeling the teammate agent offers a more flexible approach, while the modeling process is inherently unstable as the teammate agent's strategy may continuously evolve, demanding additional resources for maintenance.

In order to strike a balance between the generalization ability of built agents and the efficiency of incorporating teammate information, particularly for LLMs that possess excellent reasoning capabilities but face challenges in fine-tuning or learning extra belief modules, ProAgent introduces three core components along with a cooperative reasoning and planning mechanism. The three modules encompass: 1) The `Memory` module, which stores information about task trajectory and general knowledge in the task domain. 2) The `Verificator` module, consisting of one component for skill failure analysis and another for transforming skills into atomic actions. 3) The `Controller` module, dedicated to the transformation of skills into atomic actions. To further align the LLMs' belief regarding the teammate agent's intentions with actual behavior, and thereby continually enhance prediction accuracy, ProAgent implements the `Belief Correction` mechanism. This process effectively strengthens the LLMs' beliefs, leading to improved cooperative reasoning and planning.

In the following content, we will delve into the specifics of these modules and show how they enhance ProAgent's coordination with its teammates.

Memory Module: Leveraging History for Cooperative Behavior In ProAgent, the `Memory` module plays a crucial role in supporting information storage and retrieval processes. It consists of two components: `Knowledge Library` and `Trajectory`. The `Knowledge Library` acts as a persistent repository, retaining a comprehensive record of the task, including its layout, rules, and demonstrations throughout gameplay sessions. On the other hand, the `Trajectory` component serves as a temporary buffer with a fixed length, following a *First-In, First-Out* (FIFO) approach. It stores essential information, such as the latest `Language-based State`, `Analysis`, `Belief` of teammates' intentions, and the `Skill` used, while discarding the most outdated data. When needed, only specific parts of the `Memory` are retrieved, depending on the chosen strategy, such as the `recent-K` strategy[1]. This strategy focuses on the immediate context, facilitating efficient decision-making and planning during ongoing interactions. Overall, the `Memory` module significantly enhances ProAgent's capacity to access pertinent information and cooperate efficiently with teammate agents. By leveraging past experiences and learning from historical data, the `Memory` module empowers ProAgent to make informed decisions during cooperation tasks.

Planner Module: Reasoning with Chain of Thought With the history information and current state descrip-

[1] only retrieve the K most recent trajectories.

tion ready, ProAgent utilizes the strong reasoning ability of LLMs to make decisions in the current situation. The `Planner` module, which follows the Chain of Thought (CoT) approach commonly used in LLMs' reasoning and planning work (Yao et al. 2022; Hao et al. 2023; Shinn et al. 2023). Instead of directly outputting a plan, the `Planner` module makes the final decision step by step. The provided information is first thoroughly analyzed, and the intention of the teammate agent's plan for the current step is predicted. Based on this `Analysis` and the `Belief` about the teammate agent, LLMs formulate a plan that ensures it is the most reasonable and effective strategy for the given situation. In the experiment part, we conduct an ablation study to assess how this design enhances ProAgent's performance in a cooperative scenario.

Verificator Module: Analyzing Skill Failures With Multi-rounds Prompts In the cooperative setting, the `Verificator` module plays a crucial role in scrutinizing and identifying any unreasonable or flawed planning generated by the LLMs. Its primary function involves analyzing the underlying reasons for these inadequacies and providing valuable insights and suggestions for improvement. In the ProAgent framework, this process entails conducting a thorough investigation through multiple rounds of prompt and response between the agent and the LLMs.

To illustrate this process, we present an example based on Overcooked-AI, where we employ a `Preconditions Check` prompt and response. It's important to note that the number of rounds or the specific interaction style is not restricted, and the core idea behind the `Verificator` module remains focused on decomposing how to replan for the current agent when receiving negative feedback from external environments by checking and determination.

Preconditions Check: The `Preconditions Check` involves signaling the LLMs if the current plan is illegal due to internal checks before its actual execution. A robust internal checking mechanism can prevent failures when the LLMs haven't fully understood the consequences of their chosen skill under the current state. In the Overcooked-AI example, we design the condition check prompt by leveraging both the current scene and the failed skill as inputs. We employ a trigger prompt to enable the LLMs to individually verify each precondition of the skill and pinpoint the specific one that led to the failure. To aid in solving multi-step reasoning problems, prompting techniques like CoT are also adopted. An instance of the trigger prompt in Overcooked-AI could be: "*Analysis of why I cannot execute this skill in the current scene step by step.*" The preconditions of each skill can be expressed either in natural language or in pseudo-code form, which can be more effective as proposed in previous works (Liang et al. 2022; Singh et al. 2022). As an example, we construct the precondition for the skill `pickup(onion)` as follows:

```
# assert structure
def pickup(obj):
    assert object_in_hand() == "nothing", f"I cannot
    ↪  pickup(obj),
        since that hand is not empty."
    assert obj in ["onion", "dish"], f"I cannot
    ↪  pickup(obj),
        since this is an invalid object."
    return f"pickup(obj)"

# if...else structure
def pickup(obj):
    if object_in_hand() == "nothing":
        if obj in ["onion", "dish"]:
            return f"pickup(obj)"
        else:
            return f"I cannot pickup(obj),
            since that this is an invalid object."
    else:
        return f"I cannot pickup(obj), since that hand
        ↪  is not empty."
```

Belief Correction: Rectifying Predictions on Teammate Agents The `Belief Correction` mechanism plays a pivotal role in rectifying any incorrect beliefs during cooperation. ProAgent make predictions about their teammates' future behavior and store relevant analyses in their memory. In subsequent steps, ProAgent verify the accuracy of their predictions and correct any erroneous beliefs. Specifically, if the observed behavior of the teammate agent deviates from the assumed intentions recorded in `Memory`, the `Belief Correction` mechanism can take two approaches: 1) Replace the predicted intention with the actual behavior of the teammate. 2) Provide an annotation alongside the original prediction to flag it as incorrect. The replacement method enforces ProAgent to learn from ground truth, while the annotation method allows ProAgent to reason about the cause of the wrong belief, thereby avoiding similar mistakes in the future.

Additionally, the replan loop within the `Verificator` module serves as an indirect method for rectifying beliefs. With each query to the LLMs, ProAgent outputs new intentions on their teammate agent, which contributes to improving the accuracy of their predictions. This iterative process allows ProAgent to refine their beliefs over time and enhance their ability to make accurate predictions about their teammate's intentions.

In summary, the `Belief Correction` mechanism ensures that ProAgent maintains accurate and up-to-date information about their teammate agent's real behavior. By referencing the `Belief` part of `Memory` before making decisions, ProAgent continually improves the accuracy of their beliefs regarding their teammate's future behavior.

Controller: Grounding ProAgent High-Level Skill Plan to Low-Level Actions

Based on the modules and mechanisms as discussed above, ProAgent effectively engages in cooperative reasoning and plans a high-level skill. However, it is worth noting that there is a gap between the skill space and the environment's action space. Therefore, we also need a `Controller` module which is imperative, aiming to convert language-based

skills into low-level actions that can be executed in the environment. Although this transformation process is closely tied to the specific task at hand, making the `Controller` module highly flexible, it necessitates the establishment of fixed rules capable of decomposing the skill into multiple steps of low-level actions and providing a feedback signal to the reasoning component once the action is fully executed. The controller can be a rule-based path search algorithm or a policy trained by language-grounded reinforcement learning (Hanjie, Zhong, and Narasimhan 2021; Ding et al. 2023; Hu and Sadigh 2023; Du et al. 2023) methods. Considering that the controller is not our main concern, we choose the built-in controller in the overcooked-ai environment based on Best-First Search and a better controller can definitely reach better performance. An example of how the skill `fill_dish_with_soup()` is executed and completed in three timesteps can be found in the appendices.

Experiments

Experimental Settings

Following previous work on cooperative AI and human-AI cooperation, we choose Overcooked-AI as our test environment, in which two agents swiftly prepare and serve soups by placing up to three ingredients in a pot, cooking the soup, filling the soup with the dish, and delivering the soup. Agents must dynamically allocate tasks and collaborate effectively. Five classical layouts are used: *Cramped Room*, *Asymmetric Advantages*, *Forced Coordination*, *Coordination Ring*, and *Counter Circuit*. A detailed description of each layout can be found in the appendices.

Our primary concern behind this work is how well the agents developed so far based on ZSC methods can collaborate with diverse teammates, ranging from different AI agents to humans. In previous works on Overcooked-AI, the cooperative performance of an agent is often evaluated with two held-out populations: self-play (SP) agent and human proxy model. We conduct a comparative analysis between our proposed ProAgent and alternatives prevalent in the field including SP (Tesauro 1994; Carroll et al. 2019), PBT (Jaderberg et al. 2017), FCP (Strouse et al. 2021), MEP (Zhao et al. 2023), and COLE (Li et al. 2023a,b). Besides, we also select the human proxy model proposed by (Carroll et al. 2019) to test the agent's ability to cooperate with humans. We traversed each algorithm in combination with all other algorithms: we put an agent under one algorithm A and pair it with another algorithm B to form (A, B) pair and testing it with 10 episodes, collecting the mean and standard variation data of overall return. In addition, we switch positions and form a new pair (B, A) as the starting point of the two agents is not symmetrical.

Collaborating with AI Agents

Quantitative Results Table 1 illustrates the average performance of SP, PBT, FCP, MEP, COLE, and ProAgent when paired with all the others. For each layout, the first row represents the scenario where the agent takes the role of Player 0, and the AI partner takes the role of Player 1. The second row depicts the vice-versa scenario. The results indicate that ProAgent outperforms the baselines in all layouts when acting as Playe 0. Taking the role of Player 1, ProAgent only slightly underperforms FCP in cramped room layout and loses to PBT in forced coordination layout. We will examine this failure further in the appendices. In previous studies, it is rare to compare different AI agent combinations with each other, and our experimental results also reveal that none of the other ZSC methods is consistently better than other methods. Considering that ProAgent requires no specific training with distinct teammates and in distinct layouts, it presents a stronger adaptive ability than the other AI agents. These results show our LLM-based agent is a better cooperator.

Qualitative Results To gain deeper insights into the fundamental components of effective cooperation, we perform a qualitative examination of our ProAgent's behaviors exhibited during our experiments, leading us to identify several cooperative behaviors.

ProAgent showcases a remarkable capacity to execute high-level plans while maintaining adaptability. For instance, when ProAgent intends to deposit an onion into a pot, its underlying low-level controller identifies a blocked path caused by its teammate. Swiftly, the controller identifies an alternative interconnected route, skillfully bypassing any potential obstructions. This adaptive approach remains effective even in situations where an adversary might favor a counter-clockwise trajectory. ProAgent's controller dynamically adjusts its strategy based on the current scenario to uncover unhindered pathways. ProAgent is also able to effectively circumvent hindering crucial positions or pathways. For instance, ProAgent can issue a pause instruction, which is interpreted by `Controller` as a randomized movement. This dynamic maneuver ensures that ProAgent refrains from obstructing both pots, instead relinquishing its position.

ProAgent excels in rectifying errors. When erroneously instructing to retrieve a dish in the previous step, ProAgent can promptly recognize the inappropriateness, and query a new and correct planning. Moreover, ProAgent showcases an in-depth comprehension of the present situation. For example, when one pot is actively cooking and the other pot lacks an onion, ProAgent secures putting the onion into the pot with first priority, averting the potential conflicts stemming from fetching a dish. ProAgent exhibits its cooperative behavior in various scenarios, where it can adapt, prevent blocking, correct errors, and react intelligently to the current situation.

Collaborating with Humans

Apart from cooperation with AI agents, our concern also involves the generalization of novel human partners. Due to the limitation to collect human interaction data, we follow the previous work (Carroll et al. 2019) that uses a BC model trained on human data as a proxy of humans. Fig. 2 presents the average cumulative rewards achieved over a span of 400 timesteps by ProAgent when engaged in collaboration with BC. The reported outcomes encompass both the mean value and standard error across five distinct BC models. Analy-

Layout	Baseline AI Agents					ProAgent (ours)
	SP	PBT	FCP	MEP	COLE	
Cramped Room	168.5 ± 15.2	178.8 ± 16.5	196.3 ± 16.8	185 ± 15	163.8 ± 24.1	$\mathbf{197.3} \pm 6.1$
	172.8 ± 16.1	179.8 ± 26.8	$\mathbf{196} \pm 11.9$	178.2 ± 15.6	169.2 ± 16.8	194.2 ± 10.5
Asymmetric Advantages	183.3 ± 27.5	182.2 ± 27.9	185.7 ± 22.7	155.7 ± 63.9	201.3 ± 34.5	$\mathbf{228.7} \pm 23$
	177.8 ± 24.6	152.3 ± 64.5	167.8 ± 21.3	184 ± 41.8	165.5 ± 33.3	$\mathbf{229.8} \pm 21.9$
Coordination Ring	122 ± 17.2	141.3 ± 28	148.8 ± 19.4	167.2 ± 22.4	168.8 ± 26.1	$\mathbf{175.3} \pm 29$
	133.3 ± 23.7	141.3 ± 27.5	145.7 ± 17.1	159.3 ± 25.3	158.3 ± 27.1	$\mathbf{183} \pm 31.7$
Forced Coordination	6.7 ± 6.7	15.3 ± 17.1	44.7 ± 36.4	23.3 ± 19.8	24 ± 21.8	$\mathbf{49.7} \pm 33.1$
	30.2 ± 21.9	$\mathbf{61.7} \pm 46$	32.2 ± 30.2	39.3 ± 16.9	57.3 ± 36.4	31 ± 33.9
Counter Circuit	64.7 ± 45.8	64.7 ± 45.9	58.3 ± 37.5	74.3 ± 39.1	95.5 ± 25.2	$\mathbf{126.3} \pm 32.3$
	60.7 ± 40.8	54.3 ± 49.1	60 ± 38.3	81.5 ± 27.5	100.8 ± 31.1	$\mathbf{128.5} \pm 28.1$

Table 1: Performance for all AI agent pairs. Each column represents the average reward and standard error of one algorithm playing with all others. For each layout, the first row represents the scenario where the agent takes the role of Player 0, and the AI partner takes the role of Player 1. The second row depicts the vice-versa scenario. The best results for each layout are highlighted in bold.

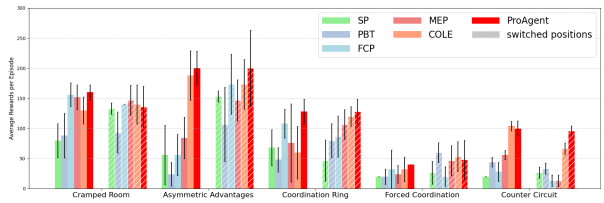


Figure 2: Performance with human proxy partners. In each layout, the reward bar represents the average performance of one algorithm collaborating with the unseen human proxy partners over 400 timesteps on five different random seeds, and the error lines represent the standard error. The hashed bars indicate the rewards obtained where the starting positions are switched. Zoom in for better visualization.

sis of the experimental findings reveals that across the five environments, ProAgent outperforms the baseline in four environments, exhibiting particularly noteworthy superiority when functioning as Player 0 in the context of *Forced Coordination*. Notably, the positioning discrepancy between the left and right starting positions had a negligible impact on ProAgent's performance. However, this difference led to substantial performance disparities among the baselines, particularly in asymmetric layouts, where the cumulative rewards achieved by all baselines were superior in the left position compared to the right position, consistent with the findings in COLE (Li et al. 2023a,b).

Discussion

Does analysis and belief help in Planner Module? To gauge the influence of *analysis* and *belief* on the accuracy and efficiency of decisions made by the `Planner` Module, we conducted an ablation study within the context of the *Cramped Room* layout. The experiment considered three distinct conditions and their respective scores were: 1) 204 for with both *analysis* and *belief*, 2) 184 without *belief*, and 3) 100 for no *analysis* and *belief*, and making a skill plan directly. We believe that the significance of analysis in the `Planner` Module lies in its provision of in-context for final planning just as CoT will improve the effect of reasoning. Moreover, the natural language analysis offers insights into the decision-making process of LLM-based agents, which is transparent and interpretable to humans. Additionally, inferring teammate intentions provides further improvements.

Is Verificator effective in feedback-based reasoning? Upon removing the `Verificator` Module and allowing LLMs to engage in planning without feedback, we computed success rates over 100 steps. Notably, the success rate dropped significantly to 20%, underscoring the critical role of our `Verificator` Module in furnishing feedback when the `Planner` Module generates inaccurate plans.

Failure Cases and Limitations

In the *forced coordination* layout, where players are separated by three counters, coordination is crucial for successful soup delivery. However, those ZSC methods consistently demonstrate specific conventions. For instance, in interactions between two FCP agents, they consistently exchange onions on the first counter and dishes on the third counter. Similarly, instances of two MEP agents exhibit a pattern of exclusive item exchange on the first counter, while instances of two COLE agents exchange dishes on the first counter and onions on either the first or second counter. These conventions, while effective for cooperating with themselves to achieve high scores, pose challenges for effective coordination with novel agents. For example, when collaborating with FCP, ProAgent (player 1) encounters issues as the FCP agent (player 0) inexplicably withholds putting onions into pots. In collaboration with MEP, ProAgent mistakenly interacts with the MEP agent, which fails to carry out essential actions correctly.

Conclusion

In this work, we have introduced ProAgent, a proactive framework grounded in LLMs, with the primary objective of addressing the multi-agent coordination predicament. By

leveraging the inherent faculties of LLMs encompassing common sense comprehension and language-centric task understanding, coupled with explicit mechanisms for reasoning and planning, ProAgent demonstrates remarkable performance within cooperative AI scenarios. Notably, it showcases a remarkable capacity to cooperate effectively with novel agents, dynamically tailoring its conduct to harmonize with its teammate's strategies. The pivotal reasoning and planning modules operate exclusively on human language, underpinning its transparency and interpretability. Experiments on cooperating with both AI agents and humans in the Overcooked-AI demonstrate the effectiveness of ProAgent over state-of-the-art methods. These encouraging results pave the way for further advancements in cooperative AI built upon LLMs.

References

Ahn, M.; Brohan, A.; Brown, N.; Chebotar, Y.; Cortes, O.; David, B.; Finn, C.; Fu, C.; Gopalakrishnan, K.; Hausman, K.; et al. 2022. Do as I can, not as I say: Grounding language in robotic affordances. *arXiv preprint arXiv:2204.01691*.

Brown, T.; Mann, B.; Ryder, N.; Subbiah, M.; Kaplan, J. D.; Dhariwal, P.; Neelakantan, A.; Shyam, P.; Sastry, G.; Askell, A.; et al. 2020. Language models are few-shot learners. *Advances in neural information processing systems*, 33: 1877–1901.

Bubeck, S.; Chandrasekaran, V.; Eldan, R.; Gehrke, J.; Horvitz, E.; Kamar, E.; Lee, P.; Lee, Y. T.; Li, Y.; Lundberg, S.; et al. 2023. Sparks of artificial general intelligence: Early experiments with GPT-4. *arXiv preprint arXiv:2303.12712*.

Carroll, M.; Shah, R.; Ho, M. K.; Griffiths, T.; Seshia, S.; Abbeel, P.; and Dragan, A. 2019. On the utility of learning about humans for human-ai coordination. *Advances in neural information processing systems*, 32.

Ding, Z.; Zhang, W.; Yue, J.; Wang, X.; Huang, T.; and Lu, Z. 2023. Entity divider with language grounding in multi-agent reinforcement learning. In *International Conference on Machine Learning*, 8103–8119. PMLR.

Du, Y.; Watkins, O.; Wang, Z.; Colas, C.; Darrell, T.; Abbeel, P.; Gupta, A.; and Andreas, J. 2023. Guiding pre-training in reinforcement learning with large language models. *arXiv preprint arXiv:2302.06692*.

Fan, L.; Wang, G.; Jiang, Y.; Mandlekar, A.; Yang, Y.; Zhu, H.; Tang, A.; Huang, D.-A.; Zhu, Y.; and Anandkumar, A. 2022. MineDojo: Building Open-Ended Embodied Agents with Internet-Scale Knowledge. In *NIPS Processing Systems Datasets and Benchmarks Track*.

Gronauer, S.; and Diepold, K. 2022. Multi-agent deep reinforcement learning: A survey. *Artificial Intelligence Review*, 1–49.

Hanjie, A. W.; Zhong, V. Y.; and Narasimhan, K. 2021. Grounding language to entities and dynamics for generalization in reinforcement learning. In *International Conference on Machine Learning*, 4051–4062. PMLR.

Hao, S.; Gu, Y.; Ma, H.; Hong, J. J.; Wang, Z.; Wang, D. Z.; and Hu, Z. 2023. Reasoning with language model is planning with world model. *arXiv preprint arXiv:2305.14992*.

Hu, H.; and Foerster, J. N. 2019. Simplified action decoder for deep multi-agent reinforcement learning. *arXiv preprint arXiv:1912.02288*.

Hu, H.; Lerer, A.; Cui, B.; Pineda, L.; Brown, N.; and Foerster, J. 2021a. Off-belief learning. In *International Conference on Machine Learning*, 4369–4379. PMLR.

Hu, H.; and Sadigh, D. 2023. Language instructed reinforcement learning for human-ai coordination. *arXiv preprint arXiv:2304.07297*.

Hu, S.; Zhu, F.; Chang, X.; and Liang, X. 2021b. Updet: Universal multi-agent reinforcement learning via policy decoupling with transformers. *arXiv preprint arXiv:2101.08001*.

Huang, J.; and Chang, K. C.-C. 2022. Towards Reasoning in Large Language Models: A Survey. *arXiv preprint arXiv:2212.10403*.

Jaderberg, M.; Dalibard, V.; Osindero, S.; Czarnecki, W. M.; Donahue, J.; Razavi, A.; Vinyals, O.; Green, T.; Dunning, I.; Simonyan, K.; et al. 2017. Population based training of neural networks. *arXiv preprint arXiv:1711.09846*.

Kojima, T.; Gu, S. S.; Reid, M.; Matsuo, Y.; and Iwasawa, Y. 2022. Large language models are zero-shot reasoners. *arXiv preprint arXiv:2205.11916*.

Li, Y.; Zhang, S.; Sun, J.; Du, Y.; Wen, Y.; Wang, X.; and Pan, W. 2023a. Cooperative Open-ended Learning Framework for Zero-shot Coordination. In *International conference on machine learning*. PMLR.

Li, Y.; Zhang, S.; Sun, J.; Zhang, W.; Du, Y.; Wen, Y.; Wang, X.; and Pan, W. 2023b. Tackling Cooperative Incompatibility for Zero-Shot Human-AI Coordination. *arXiv preprint arXiv:2306.03034*.

Liang, J.; Huang, W.; Xia, F.; Xu, P.; Hausman, K.; Ichter, B.; Florence, P.; and Zeng, A. 2022. Code as policies: Language model programs for embodied control. *arXiv preprint arXiv:2209.07753*.

Lucas, K.; and Allen, R. E. 2022. Any-play: An intrinsic augmentation for zero-shot coordination. *arXiv preprint arXiv:2201.12436*.

Lupu, A.; Cui, B.; Hu, H.; and Foerster, J. 2021. Trajectory diversity for zero-shot coordination. In *International conference on machine learning*, 7204–7213. PMLR.

Meng, L.; Wen, M.; Yang, Y.; Le, C.; Li, X.; Zhang, W.; Wen, Y.; Zhang, H.; Wang, J.; and Xu, B. 2021. Offline pre-trained multi-agent decision transformer: One big sequence model tackles all smac tasks. *arXiv preprint arXiv:2112.02845*.

Mialon, G.; Dessì, R.; Lomeli, M.; Nalmpantis, C.; Pasunuru, R.; Raileanu, R.; Rozière, B.; Schick, T.; Dwivedi-Yu, J.; Celikyilmaz, A.; et al. 2023. Augmented language models: A survey. *arXiv preprint arXiv:2302.07842*.

Ouyang, L.; Wu, J.; Jiang, X.; Almeida, D.; Wainwright, C.; Mishkin, P.; Zhang, C.; Agarwal, S.; Slama, K.; Ray, A.; et al. 2022. Training language models to follow instructions with human feedback. *Advances in Neural Information Processing Systems*, 35: 27730–27744.

Paul, D.; Ismayilzada, M.; Peyrard, M.; Borges, B.; Bosselut, A.; West, R.; and Faltings, B. 2023. REFINER: Reasoning Feedback on Intermediate Representations. *arXiv preprint arXiv:2304.01904*.

Rashid, T.; Samvelyan, M.; Schroeder, C.; Farquhar, G.; Foerster, J.; and Whiteson, S. 2018. QMIX: Monotonic value function factorisation for deep multi-agent reinforcement learning. In *International Conference on Machine Learning*, 4295–4304. PMLR.

Shinn, N.; Cassano, F.; Labash, B.; Gopinath, A.; Narasimhan, K.; and Yao, S. 2023. Reflexion: Language agents with verbal reinforcement learning. *arXiv preprint arXiv:2303.11366*.

Singh, I.; Blukis, V.; Mousavian, A.; Goyal, A.; Xu, D.; Tremblay, J.; Fox, D.; Thomason, J.; and Garg, A. 2022. Progprompt: Generating situated robot task plans using large language models. *arXiv preprint arXiv:2209.11302*.

Strouse, D.; McKee, K.; Botvinick, M.; Hughes, E.; and Everett, R. 2021. Collaborating with humans without human data. *Advances in Neural Information Processing Systems*, 34: 14502–14515.

Tesauro, G. 1994. TD-Gammon, a self-teaching backgammon program, achieves master-level play. *Neural computation*, 6(2): 215–219.

Wang, X.; Wei, J.; Schuurmans, D.; Le, Q.; Chi, E.; and Zhou, D. 2022a. Self-consistency improves chain of thought reasoning in language models. *arXiv preprint arXiv:2203.11171*.

Wang, Y.; Zhong, F.; Xu, J.; and Wang, Y. 2022b. ToM2C: Target-oriented Multi-agent Communication and Cooperation with Theory of Mind. In *International Conference on Learning Representations*.

Wang, Z.; Cai, S.; Liu, A.; Ma, X.; and Liang, Y. 2023. Describe, explain, plan and select: Interactive planning with large language models enables open-world multi-task agents. *arXiv preprint arXiv:2302.01560*.

Wei, J.; Wang, X.; Schuurmans, D.; Bosma, M.; Chi, E.; Le, Q.; and Zhou, D. 2022. Chain of thought prompting elicits reasoning in large language models. *arXiv preprint arXiv:2201.11903*.

Welleck, S.; Lu, X.; West, P.; Brahman, F.; Shen, T.; Khashabi, D.; and Choi, Y. 2022. Generating Sequences by Learning to Self-Correct. *arXiv preprint arXiv:2211.00053*.

Wen, M.; Kuba, J.; Lin, R.; Zhang, W.; Wen, Y.; Wang, J.; and Yang, Y. 2022. Multi-agent reinforcement learning is a sequence modeling problem. *Advances in Neural Information Processing Systems*, 35: 16509–16521.

Wu, S. A.; Wang, R. E.; Evans, J. A.; Tenenbaum, J. B.; Parkes, D. C.; and Kleiman-Weiner, M. 2021. Too Many Cooks: Bayesian Inference for Coordinating Multi-Agent Collaboration. *Topics in Cognitive Science*, 13(2): 414–432.

Yang, Y.; and Wang, J. 2020. An overview of multi-agent reinforcement learning from game theoretical perspective. *arXiv preprint arXiv:2011.00583*.

Yao, S.; Zhao, J.; Yu, D.; Du, N.; Shafran, I.; Narasimhan, K.; and Cao, Y. 2022. React: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629*.

Yu, C.; Velu, A.; Vinitsky, E.; Gao, J.; Wang, Y.; Bayen, A.; and Wu, Y. 2022. The surprising effectiveness of ppo in cooperative multi-agent games. *Advances in Neural Information Processing Systems*, 35: 24611–24624.

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

Zhao, R.; Song, J.; Yuan, Y.; Hu, H.; Gao, Y.; Wu, Y.; Sun, Z.; and Yang, W. 2023. Maximum entropy population-based training for zero-shot human-ai coordination. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, 6145–6153.

Zhong, Y.; Kuba, J. G.; Hu, S.; Ji, J.; and Yang, Y. 2023. Heterogeneous-Agent Reinforcement Learning. *arXiv preprint arXiv:2304.09870*.

Zhou, D.; Schärli, N.; Hou, L.; Wei, J.; Scales, N.; Wang, X.; Schuurmans, D.; Bousquet, O.; Le, Q.; and Chi, E. 2022. Least-to-most prompting enables complex reasoning in large language models. *arXiv preprint arXiv:2205.10625*.

Zhu, D.; Chen, J.; Shen, X.; Li, X.; and Elhoseiny, M. 2023. Minigpt-4: Enhancing vision-language understanding with advanced large language models. *arXiv preprint arXiv:2304.10592*.

Appendices

Environment Details

Layouts

Figure 3: Five classical Overcooked-AI (Carroll et al. 2019) environment layouts. From left to right: Cramped Room, Asymmetric Advantages, Coordination Ring, Forced Coordination, and Counter Circuit.

1. **Cramped Room**: The *Cramped Room* scenario maintains its simplicity, featuring two players confined to a compact space. This limited area contains a solitary pot (a black box with a gray base) and a lone serving spot (a light gray square). As a result, players are tasked with optimizing pot utilization and ensuring efficient soup delivery through basic coordination.
2. **Asymmetric Advantages**: In the context of the *Asymmetric Advantages* layout, two players find themselves situated in separate kitchens. As the name implies, the placement of onions, pots, and serving spots showcases deliberate asymmetry. Within the left kitchen, onions are positioned distantly from the pots, while serving spots are conveniently located near the center of the layout. Conversely, the right kitchen presents onions placed in proximity to the central area, accompanied by serving spots at a distance from the pots.
3. **Coordination Ring**: The *Coordination Ring* layout necessitates continuous movement from both players to avoid obstructing each other, particularly around the top-right and bottom-left corners where onions and pots are stationed. Effective collaboration demands the active engagement of both pots.
4. **Forced Coordination**: The *Forced Coordination* scenario deliberately separates the two agents. The left side lacks pots and serving spots, mirroring the absence of onions and pots on the right side. Consequently, successful task completion relies on seamless collaboration. The left player focuses on onion and plate preparation, while the right player handles cooking and serving.
5. **Counter Circuit**: The *Counter Circuit* introduces another ring-like layout, albeit with an expanded map size. Pots, onions, plates, and serving spots are strategically positioned in four distinct directions within this configuration. Due to the constraints of narrow passages, player movement can easily be impeded. Mastery of task execution in this environment demands intricate coordination. An advanced technique involves strategically placing onions in the central area to enable rapid exchange with the other player, resulting in heightened overall performance.

State Space

The Overcooked-AI environment provides both a grid world and a rendered image as the state. For example, the initial state of Cramped Room is a 4x5 grid world, for better visualization, we expand it to a 7x33 grid world, adding a blank line in the middle of each line and inserting 7 spaces between each column.

```
X       X       P       X       X

O                       ↑1      O

X       ↑0                      X

X       D       X       S       X
```

This grid world can also be transformed into a Python list of symbols, where 'E' represents EMPTY, "X" represents COUNTER, "O" represents ONION DISPENSER, "P" represents POT, "D" represents DISH DISPENSER, "S" represents SERVING LOC.

```
[['X', 'X', 'P', 'X', 'X'],
['O', 'E', 'E', '↑1', 'O'],
['X', '↑0', 'E', 'E', 'X'],
['X', 'D', 'X', 'S', 'X']]
```

Action Space

Players in the Overcooked-AI environment have only six actions that can be executed in each timestep:

1. NORTH: The player moves the player up one cell, and changes the direction up, unless blocked by a dispenser and counter or will collide with teammates, will cause to stay in place. It corresponds to "↑" on the keyboard when humans interact.
2. SOURTH: The player moves the player down one cell, and changes the direction down, unless blocked by a dispenser and counter or will collide with teammates, will cause to stay in place. It corresponds to "↓" on the keyboard when humans play.
3. WEST: The player moves the player left one cell, and changes the direction left, unless blocked by a dispenser and counter or will collide with teammates, will cause to stay in place. It corresponds to "←" on the keyboard when humans play.
4. EAST: The player moves the player right one cell, and changes the direction right, unless blocked by a dispenser and counter or will collide with teammates, will cause to stay in place. It corresponds to "→" on the keyboard when humans play.
5. INTERACT: The player interacts with the cell that they are facing. It corresponds to the "spacebar" on the keyboard when humans play.
   - If you are holding nothing and facing a specific object (onion dispense, tomato dispense, or dish dispense), you will get one item (onion, tomato, or dish)
   - If you are holding an onion/dish, and are facing a pot that is not full, you will put the onion/dish in the pot.
   - If you are holding one dish and facing the pot with the soup finished, you will get a dish with soup.
   - If you are holding one object and facing the counter, you will put the object on the counter.
   - Otherwise, nothing will happen.
6. STAY: The player stays in the same position without any interaction.

Agent Details

Algorithms Here we provide a pipeline presented in Alg. 1, describing the step-by-step process of ProAgent's cooperative reasoning and decision-making during a cooperative task:

LLMs within ProAgent We instantiate our framework with the recent LLM GPT-3.5[2] and its easy to extend to the other LLM. We access GPT-3.5 from the OpenAI API and use the parameter of temperature 0.0, top-p 1, and max tokens 256.

Controller Module Below is an example of how the skill `fill_dish_with_soup()` is executed and completed in three timesteps.

```
...
Intention for Player1: "put_onion_in_pot()".
Plan for Player0: "fill_dish_with_soup()".
====[End of reasoning]
Current skill: fill_dish_with_soup()
====[Start of controller]

--timestep: 38                    --timestep: 39                    --timestep: 40
X      X      P{øøø  X      X  |  X      X      P{øøø  X     X   |  X      X      P      X      X
|                                |
O             →1o           O  |  O      ←1o    ↑0d          O   |  O      ←1o    ↑0{øøø        O
→                                |  →
X             →0d           X  |  X                           X   |  X                           X
|                                |
X      D      X      S      X  |  X      D      X      S     X   |  X      D      X      S      X
action: P0 ↑                     |action: P0 interact               | (succeed in current skill)
r: 0 | total: 0                  |r: 0 | total: 0                   | r: 0 | total: 0
====[End of controller]
```

[2] gpt-3.5-turbo-0301

Algorithm 1: Cooperative Reasoning and Planning within ProAgent

Functions:
f_{ALIGN} (state to language); f_{PLAN} (planner); f_{CORR} (belief correction); f_{VALID} (skill validation); f_{VER} (verificator); f_{CONTR} (controller); f_{RET} (retrieval); f_{WAT} (watcher).
Initialization:
Initialize the knowledge library with instructions, allowed skills, and demonstrations: $knowledge = \text{CONCAT}(inst, skill\_pool, demo)$
Initialize the environment: env
Initialize the memory: $\mathbb{M} = knowledge$
Main Loop:
for *timestep* $t = 0, 1, \ldots$ **do**
  Get the raw state from the environment: $state_r = env.get\_state()$
  Transform raw state $state_r$ into a language-based state description: $state_l = f_{\text{ALIGN}}(state_r)$
  Get cached memory based on the current state: $\mathbb{M}_{cached} = f_{\text{RET}}(\mathbb{M}, state_l)$
  Form the prompt: $prompt = \text{CONCAT}(\mathbb{M}_{cached}, state_l)$
  Send the prompt to the LLMs and obtain: 1) the analysis of the current state, 2) a belief on the teammate's intention, and 3) the current skill plan: $analysis, belief, skill = f_{\text{PLAN}}(prompt)$
  Check whether the skill is legal to use: $flag_v = f_{\text{VALID}}(state_l, skill)$
  while $flag_v$ *is* ***False*** **do**
    Explain why this skill is not valid in the current state: $reason = f_{\text{VER}}(state_l, skill)$
    Replan with the explanation: $analysis, skill, belief = f_{\text{PLAN}}(\text{CONCAT}(prompt, reason))$
  Correct the belief by referring to the teammate's real behavior: $belief' = f_{\text{CORR}}(belief)$
  Update the memory: $\mathbb{M} = \text{CONCAT}(\mathbb{M}, state_l, analysis, skill, belief')$
  Convert the skill to a sequence of actions: $\mathbf{a} = f_{\text{CONTR}}(state_r, skill)$
  Interact with the environment using $\mathbf{a}$.

Baselines

We do not train any models. The baseline models, human proxy BC, SP, PBT, FCP (Strouse et al. 2021), MEP (Zhao et al. 2023), and COLE (Li et al. 2023a,b) are all provided from this repository [3]. For those interested in implementation details, please check their papers.

Prompt Design

Planner Task Prompt

```
Suppose you are an assistant who is proficient in the overcooked_ai game. Your goal is to control <Player 0> and
↪  cooperate with <Player 1> who is controlled by a certain strategy in order to get a high score.
- <Player 0> and <Player 1> cannot communicate.
- You cannot use move actions and don't use the location information in the observation.
- You cannot wait.
- <Pot> is full when it contains 3 onions.
- You must do deliver_soup() when you hold a soup!
- For each step, you will receive the current scene (including the kitchen, teammates' status).
- Based on the current scene, you need to
1. Describe the current scene and analysis it
2. Infer what <Player 1> will do. Format should not use natural language, but use the allowed functions, don't respond
↪  with a skill that is not in the allowed skills.
3. Plan ONLY ONE best skill for <Player 0>  to do right now. Format should not use natural language, but use the
↪  allowed functions, don't respond with a skill that is not in the allowed skills.
-Your response should be in the following format:
Analysis: [your analysis of the current scene]
Intention for Player 1: [one skill in
↪  [pickup(onion),put_onion_in_pot(),pickup(dish),fill_dish_with_soup(),deliver_soup(),place_obj_on_counter()]]
Plan for Player 0: [one skill in
↪  [pickup(onion),put_onion_in_pot(),pickup(dish),fill_dish_with_soup(),deliver_soup(),place_obj_on_counter()]]
```

[3] https://github.com/liyang619/COLE-Platform

Planner Rule Prompt

In this game, each player can ONLY perform the following 6 allowed skills: pickup, place_obj_on_counter, put_onion_in_pot, fill_dish_with_soup, deliver_soup, wait. And a warning: *Do not attempt to use any other skills that are not listed here.* We can define the skill by using the pseudocode or the text instruction:

```
def pickup(obj):
    if object_in_hand() == "nothing": # hand holds nothing
        if obj in ["onion", "dish"]:
            pass
def place_obj_on_counter(obj):
    if object_in_hand() == "obj":
        pass
def put_onion_in_pot(): # put one onion
    if object_in_hand() == "onion":
        if pot_onions_count() < 3:
            pass
def fill_dish_with_soup():
    if object_in_hand() == "dish":
        if soup_ready() or pot_started_cooking():
        # It is enough for one condition to be true
            pass
def deliver_soup():
    if object_in_hand("soup"):
        pass
def wait(num):  # wait positive num timesteps
    if type(num) == int and 0 < num <= 20:
        pass
```

```
- pickup(onion)
- I need to have nothing in hand.
- put_onion_in_pot()
- I need to have an onion in hand.
- pickup(dish)
- Need to have a soup cooking or ready in <Pot>.
- I need to have nothing in hand.
- If there isn't a  cooking or ready soup in the current scene, I shouldn't pickup(dish).
- fill_dish_with_soup()
- Need to have soup  cooking or ready in <Pot>.
- I need to pickup(dish) first or have a dish in hand.
- Then I must deliver_soup().
- deliver_soup()
- I must do deliver_soup() when I hold a dish with soup!
- I need to have soup in hand.
- The dish and soup will both disappear.
- place_obj_on_counter()
- I need to have something in hand.
- Don't  place_obj_on_counter() when I hold a dish with soup!
```

Planner Demo Prompt

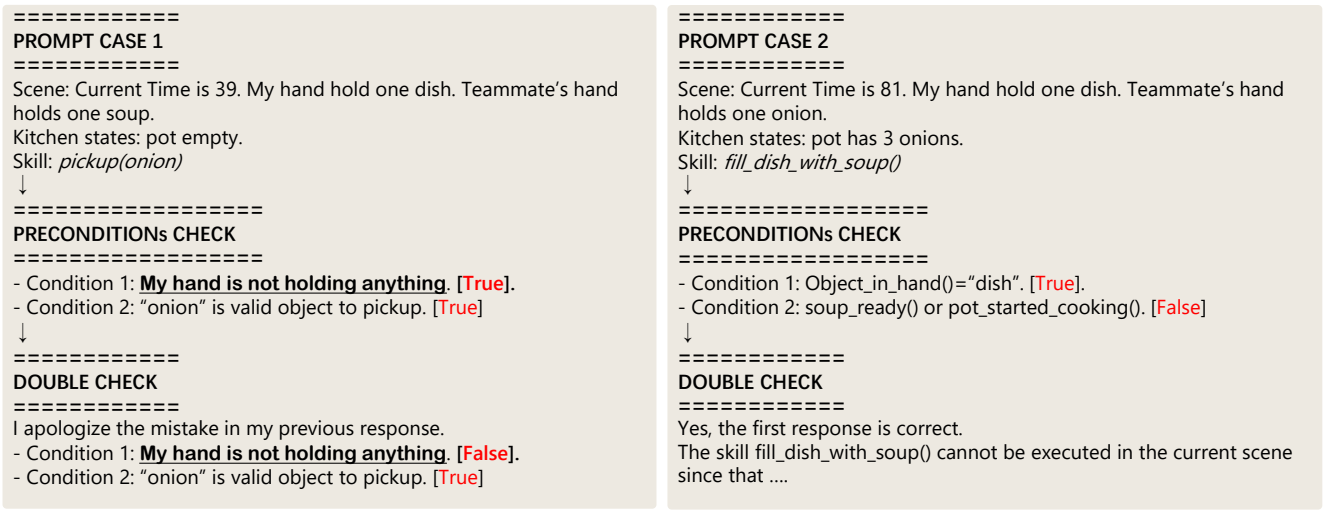
To help GPT better understand your task, it's highly recommended to use some demos as prompts in the interaction with GPT.

```
Examples:
###
Scene 999: <Player 0> holds one onion.<Player 1> holds nothing. Kitchen states: <Pot 0> is empty; <Pot 1> has 1 onion.

Analysis: <Pot 0> and <Pot 1> are not full. <Player 0> holds one onion, he should put the onion in the pot. <Player 1>
↪  holds nothing, he should pick up an onion.
Intention for Player 1: pickup(onion)
Plan for Player 0: put_onion_in_pot()
###
```

```
Scene 888: <Player 0> holds nothing.<Player 1> holds nothing. Kitchen states: <Pot 0> has 1 onion; <Pot 1> has 1 onion.

Analysis: <Pot 0> and <Pot 1> are not full. <Player 0> holds nothing, he should pick up an onion in order to cook.
↪  <Player 1> holds nothing, he should pick up an onion in order to cook.
Intention for Player 1: pickup(onion)
Plan for Player 0: pickup(onion)
###
```

Verificator Prompt

We provide Verificator Prompt in Fig. 4.



(a) The reasoning for the check was wrong, and the double-check stage found the error.

(b) The reasoning for the check was correct, and the double-check stage confirmed it.

Figure 4: Two demonstrations of the process of "check preconditions" and "double-check". The context in the figure is modified for better visualization.

Additional Results

Cross-agent collaborative evaluation

In collaboration with AI experiments, we traversed every AI algorithms pair, and only show the averaged results for each AI algorithm. As shown in Figure 5, we use the heatmap to demonstrate the total reward of each AI pair cooperation within 400 timesteps in each layout, where the row represents the AI algorithm as Player 0 and the column represents the AI algorithm as Player 1. Except for the non-ZSC results on the diagonal, we calculate the average results of each algorithm and the other five algorithms: the calculation along the horizontal axis is the result of the algorithm as Player 0, and the calculation along the vertical axis is the result of the algorithm as Player 1. We organize the results as the table in the paper's main part and as a bar graph in Figure 6.

Failure Cases and Limitations

The raw state of the Overcooked-AI is a grid world, which could be represented by either an image or a matrix of symbols. It is elegant to make decisions directly from vision-based input with LLM with multimodal capability. We tried using MiniGPT4 (Zhu et al. 2023) to make planning directly based on the grid-world image. Performance is poor because it lacks high-level reasoning capabilities. We also tried to use the matrix of symbols as the input of ProAgent, but soon found that even GPT3.5 could not clarify the distance between the positions represented by different symbols in the matrix, let alone directly plan low-level actions. This LLM's lack of spatial awareness is also proven in the recent work (Bubeck et al. 2023).

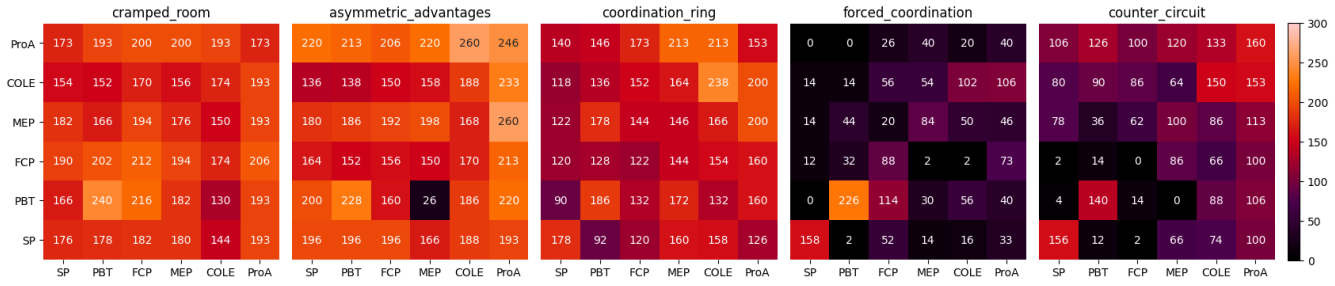


Figure 5: The heatmaps illustrate the performance of different algorithm combinations for player 0 and player 1 in five layouts of the Overcook environment from left to right: *Cramped Room, Asymmetric Advantages, Forced Coordination, Coordination Ring, and Counter Circuit*. Each heatmap corresponds to a specific layout, showing the performance of various algorithm combinations in terms of scores. The color intensity represents the score, with hotter colors indicating higher scores.

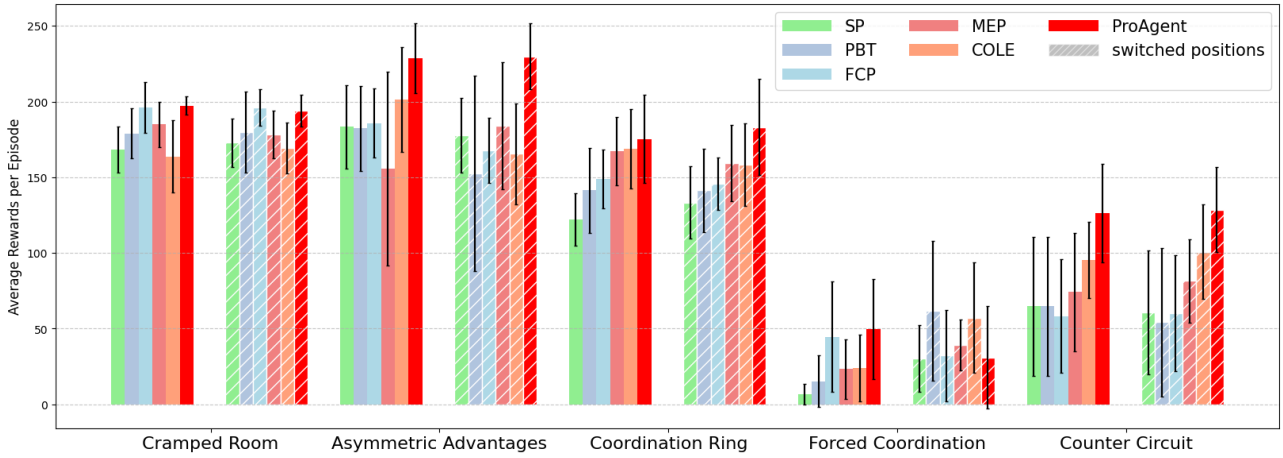


Figure 6: Cross-agent collaborative evaluation: the ZSC performance of ProAgent, COLE, FCP, MEP, PBT, and SP when paired with all the held-out populations. In each layout, the reward bar represents the average performance of one algorithm collaborating with all other algorithms, and the error lines represent the standard deviation. The gray and hashed bars indicate the rewards obtained where the starting positions are switched.