
Multi-Agent Credit Assignment with Pretrained Language Models

Wenhai Li

Tongji University

Dan Qiao

The Chinese University of
Hong Kong, Shenzhen

Baoxiang Wang

The Chinese University of
Hong Kong, Shenzhen

Xiangfeng Wang

East China Normal University

Wei Yin

Bank of Communications

Hao Shen

Fortiss GmbH

Bo Jin*

Tongji University
SRIAS

Hongyuan Zha

The Chinese University of
Hong Kong, Shenzhen

Abstract

The difficulty of appropriately assigning credit is particularly heightened in cooperative MARL with sparse reward, due to the concurrent time and structural scales involved. Automatic subgoal generation (ASG) has recently emerged as a viable MARL approach inspired by utilizing subgoals in intrinsically motivated reinforcement learning. However, end-to-end learning of complex task planning from sparse rewards without prior knowledge, undoubtedly requires massive training samples. Moreover, the diversity-promoting nature of existing ASG methods can lead to the “over-representation” of subgoals, generating numerous spurious subgoals of limited relevance to the actual task reward and thus decreasing the sample efficiency of the algorithm. To address this problem and inspired by the disentangled representation learning, we propose a novel “disentangled” decision-making method, **Semantically Aligned task decomposition in MARL (SAMA)**, that prompts pretrained language models with chain-of-thought that can suggest potential goals, provide suitable goal decomposition and subgoal allocation as well as self-reflection-based replanning. Additionally, SAMA incorporates language-

grounded MARL to train each agent’s subgoal-conditioned policy. SAMA demonstrates considerable advantages in sample efficiency compared to state-of-the-art ASG methods, as evidenced by its performance on two challenging sparse-reward tasks, *Overcooked* and *MiniRTS*. The code is available at <https://anonymous.4open.science/r/SAMA/>.

1 Introduction

Credit assignment problem is notably exacerbated in cooperative multi-agent reinforcement learning (MARL) with sparse-reward compared to the single-agent context (Hernandez-Leal et al., 2019; Zhou et al., 2020; Li et al., 2022). This issue arises not only on temporal scales, as observed in assigning sparse credits within a trajectory, but also in structural scales, regarding credit assignment among agents (Agogino and Turner, 2004). *Value decomposition* serves as the primary paradigm for addressing structural one (Devlin et al., 2014; Nguyen et al., 2018; Foerster et al., 2018; Sunehag et al., 2018; Rashid et al., 2020b; Son et al., 2019; Rashid et al., 2020a; Wang et al., 2021a; Böhmer et al., 2020; Kang et al., 2022). Consequently, most studies tackling temporal and structural credit assignment problems follow the “*value decomposition+x*” framework.

In directly learning of assigning credits along a trajectory in sparse environments, the primary challenge lies in the inherent difficulty each agent faces while attempting to acquire a substantial number of beneficial trajectories through random exploration (Mesnard et al., 2021). One prevalent example of “x” involves

learning with efficient exploration, which shows efficacy in task selection (Mahajan et al., 2019; Yang et al., 2020; Zheng et al., 2021; Li et al., 2021a; Zhang et al., 2021; Gupta et al., 2021). Nevertheless, solely employing exploration still falls short of ascertaining which actions yield rare non-zero rewards (Jeon et al., 2022; Hao et al., 2023). Subgoal-based methodologies have recently emerged as a viable alternative inspired by approaches utilizing subgoals in single-agent RL (Kulkarni et al., 2016; Bellemare et al., 2016; Pathak et al., 2017; Burda et al., 2019; Ecoffet et al., 2021; Guo et al., 2022) (also referred to as intrinsically motivated RL (Colas et al., 2022)). Intuitively, these methods decompose a task into a series of goals while concurrently breaking down each goal into subgoals that require completion by agents. Both time and structural scales could receive dense and goal-directed rewards, which mitigates the credit assignment dilemma.

While promising, four primary problems must be addressed within subgoal-based MARL: **a)** sequential goal generation; **b)** rational decomposition of goals into subgoals; **c)** accurate subgoal allocation to agents; and **d)** agents’ efficient achievement of subgoals. Early solutions for the above issues primarily rely on artificially predefined rules (Becht et al., 1999; Lhaksmana et al., 2018; Min et al., 2018; Grote et al., 2020), which limits their dynamism and adaptability in new environments. Consequently, *automatic subgoal generation* (ASG) serves as a more practical implementation of “x”, permitting application to various tasks without relying on domain knowledge.

Predominant ASG methods typically incorporate solutions for the four identified problems within a two-stage, end-to-end learning procedure. Initially, subgoals for each agent’s completion are generated, followed by the learning of policies that enable the achievement of these subgoals. These approaches can be broadly classified into two categories depending on the representation of subgoals. Some methods employ embeddings for encoding subgoals, allowing their generation to be based on local observations (Wang et al., 2020, 2021b), local message-passing (Li et al., 2021b; Shao et al., 2022), or selection from a pre-generated subgoal set using global information (Yang et al., 2022a). Subsequently, goal-conditioned policies are learned through standard reward maximization. In contrast, other techniques implicitly transform subgoals into intrinsic rewards, optimizing policies for subgoal completion by maximizing the shaped reward (Phan et al., 2021; Jeon et al., 2022; Nguyen et al., 2022; Yang et al., 2022b; Li et al., 2023).

Nevertheless, the end-to-end learning of complex task planning (**a–d**), with sparse rewards and without a priori knowledge, undoubtedly necessitates massive training samples. Furthermore, existing ASG methods

often incorporate representation learning techniques promoting diversity to learn effective subgoal embeddings or intrinsic rewards. Given the rich sources of novelty in complex environments (Burda et al., 2019), this may result in the “over-representation” of subgoals, generating numerous redundant subgoals of limited relevance to the task reward and decreasing the algorithm’s sample efficiency. In contrast, humans do not explore goal spaces uniformly; instead, they rely on commonsense to generate plausibly functional goals (Du et al., 2023). Take the *Overcooked* (Carroll et al., 2019) as an example, humans can readily infer that ingredient preparation must precede cooking, while utilizing utensils is necessary for food handling before serving.

In disentangled representation learning (DRL), the objective is to separate the underlying factors of variation into semantically meaningful variables. It leads to the acquisition of representations that mimic the comprehension process of humans grounded in commonsense (Bengio et al., 2013; Wang et al., 2022b). As a semantically aligned learning scheme, DRL has demonstrated its efficacy in enhancing model effectiveness, explainability, controllability, robustness, and generalization capacity, all of which are essential for general decision-making. Motivated by DRL, we switch from the “*value decomposition+x*” and propose a method for *disentangled* decision-making, which requires learning semantically aligned and disjointed subgoals in cooperative MARL with long horizon and sparse reward. The proposed method leverages pre-trained language models (PLMs) as a trove of priors and commonsense for the automatic generation of credit-assigned subgoals. As generative models trained on extensive open-world corpora, PLMs embody rich data on human knowledge (Bommasani et al., 2021; Yang et al., 2023).

The proposed **Semantically Aligned Task Decomposition in MARL (SAMA**, Figure 1) utilizes PLM prompting, combined with the chain-of-thought (Wei et al., 2022) (CoT), to suggest potential goals and facilitate appropriate goal decomposition and allocation. To accommodate non-textual input, SAMA introduces a general preprocessing stage that employs the code understanding for translating states and actions. SAMA integrates language-grounded MARL (Hanjie et al., 2021; Ding et al., 2023) to train each agent’s goal-conditioned policy, allowing agents to understand the natural language subgoals generated by PLMs. To reduce the computational overhead associated with frequent PLM calls, the preprocessing stage also utilizes the code understanding for pre-generating the training set, which comprises diverse goals and corresponding reward functions. Furthermore, recognizing the existing limitations of PLMs, such as hallucinations, SAMA employs the *self-reflection* mechanism (Yao et al., 2023;

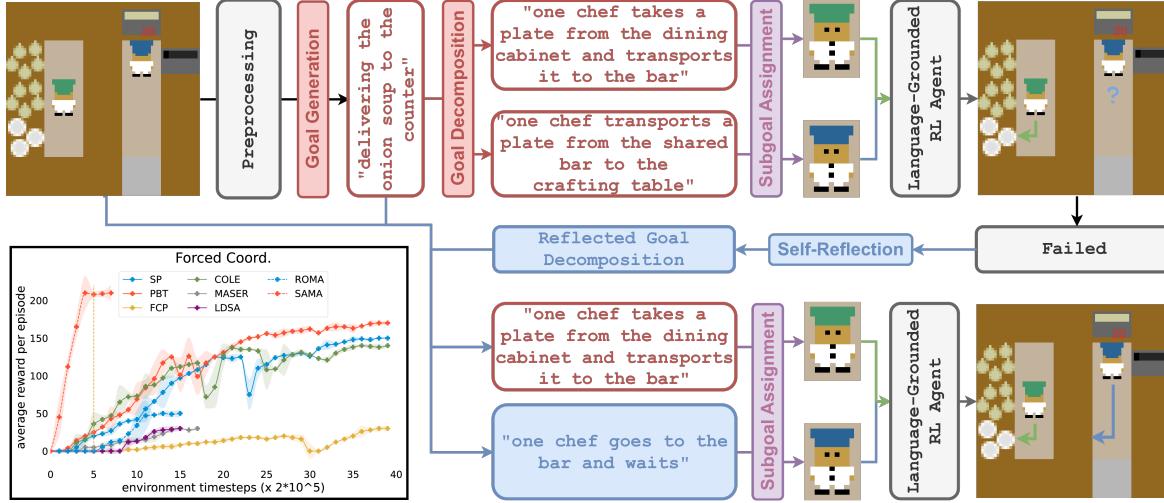


Figure 1: SAMA’s proficiency progression and experimental trials on the **Forced Coordination** within **Overcooked**. SAMA necessitates merely approximately 10% of the training instances to approach the SOTA performance. In executing the task, SAMA initially prompts the PLM to generate goals, decompose the goal to subgoals, and apportion subgoals based on the individual agent’s status, semantically aligned. Next, the cultivated language-grounded MARL policy adheres to the designated subgoals and engages with the environment. In the event of unsuccessful sub-objective execution, SAMA prompts the PLM again to re-strategize, drawing upon the self-reflection mechanism.

Shinn et al., 2023; Madaan et al., 2023; Chen et al., 2023) to prompt PLM re-planning of tasks when errors are encountered.

In summary, our main contributions are: **1)** the development of a novel framework, SAMA, which implements disentangled, commonsense-driven automatic subgoal generation by prompting PLMs to alleviate the credit assignment problem in MARL; **2)** the introduction of preprocessing based on code understanding, enabling SAMA to achieve end-to-end, environment-agnostic task decomposition; **3)** the integration of a language-grounding mechanism that allows each agent to learn an MARL policy conditioned on a natural language subgoal, facilitating efficient collaboration between MARL and PLM; and **4)** the demonstration of SAMA’s considerable advantage in sample efficiency compared to state-of-the-art subgoal-based MARL methods and language agents, as evidenced by its performance on **Overcooked** and **MiniRTS**.

2 Problem Formulation

This paper examines a fully cooperative multi-agent setting which can be characterized by a decentralized, partially observable Markov decision process (DEC-POMDP, (Peshkin et al., 2000; Bernstein et al., 2002)) $\langle \mathcal{I}, \mathcal{S}, \{\mathcal{A}_i\}_{i=1}^N, \{\mathcal{O}_i\}_{i=1}^N, \mathcal{P}, \mathcal{E}, \{\mathcal{R}_i\}_{i=1}^N \rangle$, wherein \mathcal{I} signifies the domain of N agents. The environmental state is denoted by $s \in \mathcal{S}$. Agent i is solely able to access a localized observation $o_i \in \mathcal{O}_i$ following the emission func-

tion $\mathcal{E}(o_i | s)$. At each discrete timestep, an individual agent i opts for an action $a_i \in \pi_i(a | o_i)$, originating a jointly executed action $\mathbf{a} = \langle a_1, \dots, a_n \rangle \in \times \mathcal{A}_i$. This action consequently leads to the subsequent state s' according to the transition function $P(s' | s, \mathbf{a})$ and procures a mutually experienced reward $r = \mathcal{R}(s, \mathbf{a})$.

We aim to develop an agent capable of addressing long-horizon sparse reward tasks by utilizing linguistic task manuals. To accomplish this, we consider employing planning, integrating language-grounded MARL policies designed to achieve subgoals with a planner that suggests semantically aligned goals, provide suitable goal decomposition, and distributes subgoal allocations, taking into account the environment’s current state, agents’ local contexts, and linguistic task manuals. We assume that task planning adheres to Markov properties, implying that the planner only requires performing the planning under the current contexts.

Concretely, we utilize a PLM planner, i.e., GPT-3.5 (OpenAI, 2022) and GPT-4 (OpenAI, 2023), to leverage its encoded commonsense knowledge. At each round k , the PLM planner suggests the next semantically aligned goal, g_k , that the multi-agent system must accomplish, and strives to decompose it into N distinct subgoals $\langle g_k^1, \dots, g_k^N \rangle$. Subsequently, the PLM planner assigns the N subgoals to the N agents respectively. For each agent and its associated subgoal $g_{t_k}^i$ and the task manual z at each round k , its language-grounded MARL policy $\pi(a_{t_k}^i | o_{t_k}^i, z, g_k^i)$ samples an action $a_{t_k}^i$ based on the current local observation $o_{t_k}^i$, where t_k

represents the timesteps in round k . Since completing a multi-agent task requires achieving multiple sequential goals, an episode contains multiple rounds.

3 Semantically Aligned Task Decomposition

This section provides an outline (Figure 1) of the SAMA, which comprises 3 components: The task decomposition handles credit assignment on temporal and structural scales; language-grounded MARL builds upon task decomposition, interacting with the environment to achieve goals; the self-reflection, grounded in a trial-and-error paradigm akin to RL, assimilates interaction outcomes to optimize the task decomposition. Section 3.1 addresses the necessary preprocessing for the smooth operation of SAMA’s 3 core components. Section 3.2 details the implementation of task decomposition by utilizing multi-stage prompting on PLM; Section 3.3 investigates the training of language-grounded MARL agents, ensuring that the agent’s policy aligns with textual sub-goals while fostering cooperation. Lastly, we elucidate the self-reflection mechanism’s optimization of the task decomposition, enhancing its effectiveness based on the language-grounded policy execution. Since this section primarily draw upon Reflexion (Shinn et al., 2023), we have included it in Appendix C.1.

3.1 Preprocessing

As will be demonstrated, task decomposition and self-reflection are achieved through PLM prompting, which relies on states, agents’ local contexts, and task manuals, all represented in natural language. However, it is essential to develop linguistic task manuals for a broader range of multi-agent tasks. Moreover, apart from a few text-based tasks, most environments do not supply text-based information. As a result, the translation of tasks, states, and actions becomes crucial. To enhance SAMA’s adaptability, we utilize the PLM again for translation purposes.

Task Manual Generation. Drawing inspiration from SPRING (Wu et al., 2023), we generate the language task manuals directly from the L^AT_EX code of the original paper, as illustrated in Figure 2. Initially, we compose gameplay-specific prompts and obtain the task manual by prompting the PLM with L^AT_EX files. Considering that a substantial portion of the paper is unrelated to gameplay, we employ the relevant prompt set Q_{rel} to determine relevance and the game prompt collection Q_{game} to summarize pertinent information regarding gameplay and action space. We subsequently divide the entire L^AT_EX file into paragraphs $\{\mathcal{S}_{\text{para}}^i\}$ to avoid exceeding input length constraints for most

PLMs. For each paragraph $\mathcal{S}_{\text{para}}^i$, we filter for relevance, retaining only those deemed relevant by at least one prompt from Q_{rel} . We designate P_{rel} as the set of relevant paragraphs. Following this, for each prompt in Q_{game} and each paragraph in P_{rel} , we prompt the PLM to generate corresponding answers. Finally, we prompt the PLM to deduplicate and summarize all answers, thereby producing the textual task manual.

State and Action Translation. As states, which encompass both the environmental conditions and the agent’s local context, and actions are represented by programming language rather than written text, we employ source codes for state and action translation. However, due to the non-contiguous semantics of codes, they cannot be segmented and processed individually like L^AT_EX files. Fortunately, several successful code understanding applications exist, such as Github Copilot, Code Interpreter, and the open-source tools like LangChain (Chase, 2022). Consequently, we perform state translation based on LangChain (Figure 2). Specifically, LangChain initially divides the entire code project into multiple documents according to functions and classes, leveraging context-aware splitting. Each document is subsequently encoded and stored in a vector database. Ultimately, PLM is enhanced with the ability to answer relevant questions through retrieval from the database.

Training Dataset Generation. Two primary challenges must be addressed when integrating language-grounded techniques: 1) Online subgoal generation is computationally expensive, as continuous PLM queries during language-grounded MARL training lead to significant temporal and financial costs. 2) A reward function capable of indicating subgoal completion is absent. In tasks with sparse rewards, rewards are assigned only upon completion, such as onion soup delivery in *Overcooked* or team victory in *MiniRTS*. To mitigate the cost associated with PLM queries and reduce language-grounded MARL training durations, we generate the training dataset in advance. This dataset comprises diverse subgoals and corresponding reward functions. For subgoals, we repurpose the technology used in *state and action translation* to convert PLM prompt from “translation” to the “generation” of a diverse set of states, $\mathcal{D}_{\text{state}}$. Subsequently, we apply the process prompt PLM described in Section 3.2 to generate a corresponding subgoal set $\mathcal{D}_{\text{subgoal}}$ for the derived states. For reward functions, we leverage the PLM to design the reward function, minimizing human intervention within the SAMA and enhance generalizability. Prior works have validated the feasibility of this approach (Kwon et al., 2023; Yu et al., 2023; Xie et al., 2024; Ma et al., 2024). Initially, we de-duplicate subgoals in $\mathcal{D}_{\text{subgoal}}$ by prompting the PLM, akin to

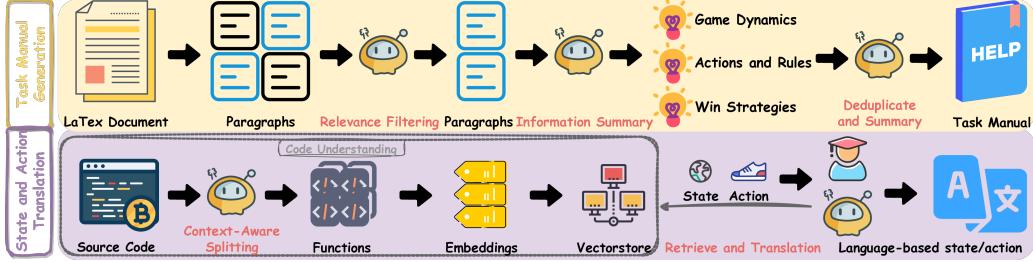


Figure 2: Illustration of preprocessing of SAMA. SAMA uses a hierarchical framework for task planning. The higher-level is based on goal generation, decomposition and subgoal assignment of PLMs (🤖), and the lower-level is based on subgoal achievement of language-grounded MARL agent. To complete the upper-level tasks, SAMA needs to generate task manuals and translate states and actions based on PLMs in advance. To complete lower-level tasks, SAMA also needs to generate diversified subgoals and design reward functions. SAMA is highly automated and requires only a few-shot example designs (สอน), which greatly improves its generalization ability for different tasks.

the *task manual generation* process, to obtain a refined subgoal set $\mathcal{D}'_{\text{subgoal}}$. Subsequently, inspired by existing prompt engineering (Yu et al., 2023), we employ the technology from *state and action translation* for PLM prompting to generate the corresponding Python code snippet capable of evaluating subgoal completion. The reward function within the code snippet set $\mathcal{D}_{\text{code}}$ accept the current state as input and return either 0 or 1. The returned values serve as binary rewards during language-grounded MARL training.

3.2 Semantically Aligned Task Decomposition

Task decomposition can also be achieved by PLM prompting, which can be broadly categorized into goal generation, decomposition, and subgoal assignment. Although exploring a goal space non-uniformly with meaningful semantics may provide valuable insights, it can be prohibitively expensive due to the highly abstract nature of open-ended tasks. Therefore, we focus on more straightforward tasks for an initial investigation. In line with most multi-agent environments and existing literature on PLM-based decision-making (Du et al., 2023; Wang et al., 2023), our approach, SAMA, primarily addresses tasks requiring continuous interactions with specific objects.

As task completion necessitates interactions with environmental objects, transition goals must incorporate these objects. Prior to goal generation, we prompt the PLM to extract interactive objects from the task manual, following a similar approach to (Varshney et al., 2023). Subsequently, we construct few-shot examples and prompt the PLM to generate rational transition goals, decomposing them into a number of subgoals corresponding to the agent count. This is based on the extracted objects and current environmental state information, facilitating temporal scale credit assignment. It is worth noting that object extraction is not manda-

tory, as goals can be generated solely from the task manual. However, our experiments show that object extraction significantly constrains the goal space generated by the PLM, improving the training efficiency of language-grounded MARL and consequently enhancing the overall performance.

After decomposing the transition goal into several subgoals equal to the number of agents, we prompt the PLM to establish a one-to-one correspondence between the subgoals and the agents. This is based on the task manual, decomposed subgoals, and each agent’s local context, resulting in the completion of the structural scale credit assignment.

3.3 Language-Grounded MARL

Following the assignment of subgoals, an efficient policy is needed to guide the agent in achieving the assigned subgoal. Considering the limitations imposed by PLMs on fine-grained decision-making (Du et al., 2023; Wang et al., 2023), we utilize a goal-conditioned MARL policy for this task. However, traditional MARL tasks mainly operate in non-textual observation spaces, relying on image- or numeric-based information. Consequently, MARL agents need to establish a relationship between text-based objectives and non-textual observations to effectively accomplish subgoals.

To address this challenge, we employ language-grounded MARL for training the goal-conditioned policy (Figure 3), which facilitates the final stage of task planning, specifically, the efficient achievement of subgoals. As described in Section 2, after completing the subgoal assignment, agent i receives the manual z and a subgoal g_k^i at round k . Meanwhile, at each timestep t_k , agent i obtains an observation o_{i,t_k} and generates a distribution over action $\pi(a_{i,t_k} | o_{i,t_k}, z, g_k^i)$. The core of language-grounded MARL involves parameterizing

policy π by incorporating a language grounding module to produce a representation $X = \text{Ground}(o_{i,t_k}, z, g_k^i)$, which reflects the relationship among the goal, manual, and observation. Agents strive to understand game dynamics and subgoals by interacting with environmental entities, ultimately updating the policy using a standard method (such as MAPPO; (Yu et al., 2022)).

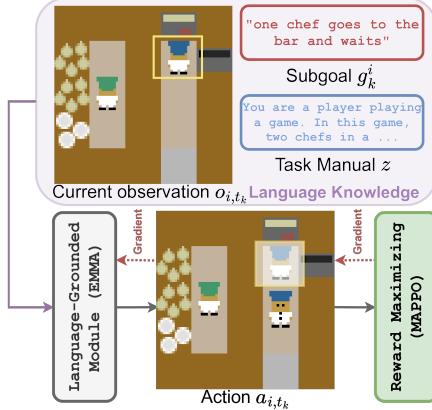


Figure 3: Language-grounded MARL.

Remarks. To monitor the training progress, we evaluate the agents at a very low frequency (Figure 5 is drawn according to the evaluation results). In light of the potential emergence of unaccounted environmental states during the evaluation process, periodic inclusion of the encountered states and corresponding PLM-generated subgoals to the dataset $\mathcal{D}_{\text{subgoal}}$ is undertaken. Moreover, an advantage of introducing PLM as a higher-level task planner is that we do not need to learn how to split subgoals, as in EDTI (Wang et al., 2020). This allows us to significantly reduce the training difficulty and sample complexity when training the language-grounded MARL algorithm. Additionally, it should be noted that SAMA’s compatibility extends to any language grounding module. Given the absence of prior testing on the *Overcooked* task for extant language-grounded RL techniques, we opted for the facile implementation of the EMMA model (Hanjie et al., 2021; Ding et al., 2023). As for MiniRTS, a corresponding language-grounded RL algorithm (Xu et al., 2022) exists; thus, we proceeded to retrain using its publicly available pretrained model. See Appendix E for more details.

4 Experiments

4.1 Case Study: Overcooked

In this section, we conduct a series of experiments in the *Overcooked* environment (Carroll et al., 2019; Charakorn et al., 2020; Knott et al., 2021; Li et al., 2021c), specifically designed to address sparse-reward,

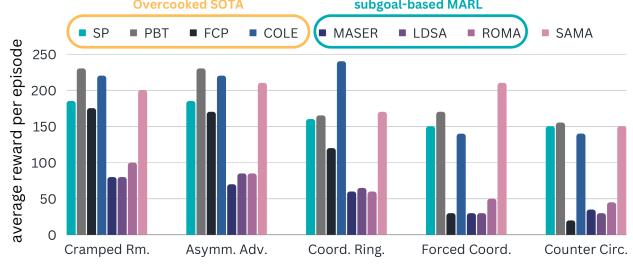


Figure 4: The mean self-play rewards of episodes over 400 timesteps under 10 random seeds. SAMA can approach or even exceed the performance of SOTA algorithm. Refer to learning curves for variances.

long-horizon, and coordination challenges (Figure 9). In this dyadic common payoff game, each player controls a chef within a culinary context, collaboratively preparing and delivering soup, thereby earning a shared reward of 20 for the team.

We evaluate our approach in comparison with alternative methods, including the SOTA technique for the *Overcooked* task, namely, selfplay (Tesauro, 1994; Carroll et al., 2019), PBT (Jaderberg et al., 2017; Carroll et al., 2019), FCP (Strouse et al., 2021), and COLE (Li et al., 2021c), all of which utilize PPO (Schulman et al., 2017) as the RL backbone. Furthermore, we examine the ASG method, MASER (Jeon et al., 2022), LDSA (Yang et al., 2022a), and ROMA (Wang et al., 2020). Considering that the current SOTA methods for the *Overcooked* task primarily focus on zero-shot coordination problems, we employ self-play rewards to demonstrate their performance.

Figure 4 (see Figure 12 in Appendix E for additional information) displays the mean rewards per episode over 400 timesteps of gameplay using 10 random seeds. Figure 5 depicts learning curves. A synthesis of these figures leads to the following observations: (1) The current SOTA in *Overcooked* requires a significant number of training samples; (2) The existing ASG methods demonstrate suboptimal performance in long-horizon, sparse-reward, and cooperative tasks, as exemplified by *Overcooked*; (3) By leveraging human commonsense embedded within PLM, SAMA can approximate or even surpass the performance of the SOTA using merely five to one-tenth of the training samples, thanks to semantically aligned task planning; (4) Due to the inherent limitations of the current PLM in long-horizon reasoning, SAMA is unable to reach optimal results. It is worth noting that enhancing the existing ASG method with approaches such as self-play and population-based training, as showcased by algorithms like COLE, could result in a substantial improvement in performance. However, this semantically non-aligned, end-to-end training without prior knowledge would also

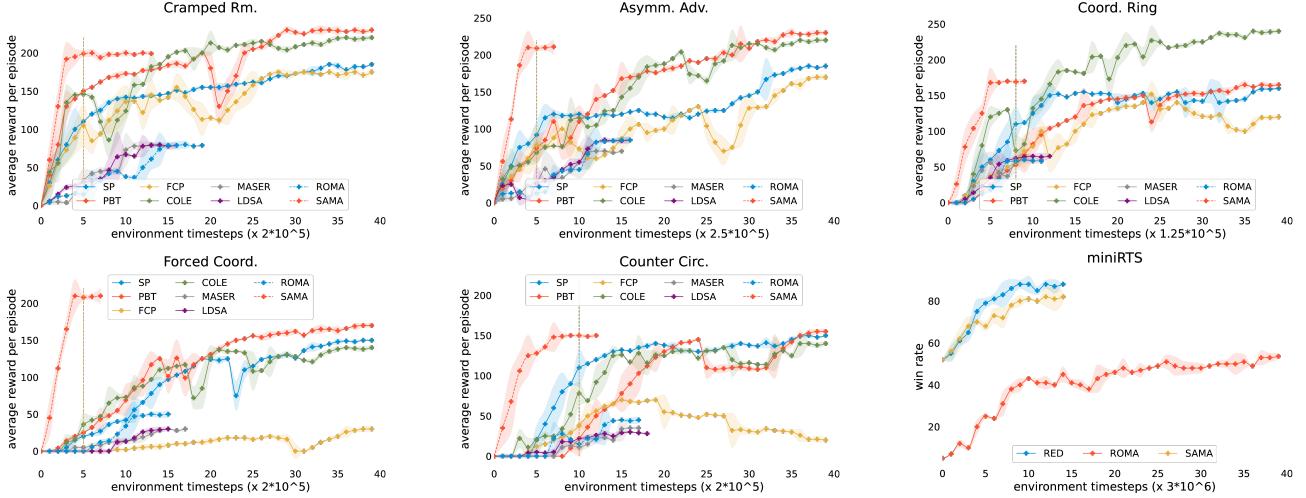


Figure 5: Learning curves of each methods in five *Overcooked*’s layouts and *MiniRTS*.

necessitate a considerable amount of training data.

4.2 Case Study: MiniRTS

This section demonstrates the effectiveness of SAMA in addressing more complex tasks. *MiniRTS* (Hu et al., 2019), a grid-world environment, embodies key aspects of sophisticated real-time strategy games (see Figure 10). The configuration involves two entities: a player (blue) controlled by human/policy intervention, positioned against a built-in script AI (red). Primary goals for the player encompass resource gathering, construction, and either eliminating enemy units or destroying the enemy base to achieve victory. 7 distinct unit types form a rock-paper-scissors attack dynamic (Figure 11), with the environment supporting a sparse reward mechanism. Upon a game’s completion, the reward equals 1 if the agent wins and -1 in case of a loss. During other timesteps, 0 reward can be obtained.

MiniRTS provides a variety of built-in script AIs, and we select the medium as the opponent. Initially, this script sends all three available peasants to mine the closest resource unit. It then randomly chooses one of the seven army unit types, determining an army size n ranging from 3 to 7. A building unit corresponding to the chosen army is constructed, followed by training n units of the selected army category for deployment in attacks. The script continues to train army units, maintaining an army size of n .

Unlike *SC II* (Samvelyan et al., 2019; Ellis et al., 2022), where individual unit control strategies can be assigned, *MiniRTS* utilizes one strategy to manage all units. To transform the latter into a multi-agent task, we implement a simple modification of its environment from *RED* (Xu et al., 2022). Two agents arise in the modified environment, each controlling half of the units.

Similarly, we also slightly altered the built-in medium-level AI, allowing the random selection of two types of army units in each round.

Considering that the built-in AI constructs only 2 army unit types per game, we devise an oracle prompt design strategy based on the ground truth of the attack graph. In the early stages of the game, the PLM guides the agent to progressively build appropriate army units. Subsequently, it sends NA and encourages the policy to function autonomously, akin to *RED*. An effective language-grounded RL policy adheres to these instructions, assembling the correct army units and performing actions independently, resulting in a higher win rate.

The proposed agent starts with *RED*’s pretrained policy, which represents the SOTA in *MiniRTS*. Besides *RED*, we investigate *ROMA* as an alternative baseline. Win rates in Figure 5 stem from 100 test games, repeated across 3 random seeds. *ROMA* is not a language-grounded MARL algorithm, so *RED*’s pretrained model is unsuitable and must be learned from scratch. Observations similar to *Overcooked* arise. Furthermore, while *RED*, capitalizing on oracle commands, exhibits unparalleled performance, *SAMA* approaches *RED* due to the employment of PLM.

4.3 PLM-based Baselines

In this section, we further compare the performance of alternative PLM-based task planners on *Overcooked* and *MiniRTS*. Specifically, we choose two algorithms, DEPS (Wang et al., 2023) and Plan4MC (Yuan et al., 2023), for comparison with *SAMA*. We select these algorithms because both necessitate planning based on existing goals or skill sets. *SAMA* must generate a diverse set of goals during the language-grounded MARL training stage, making it highly suitable for

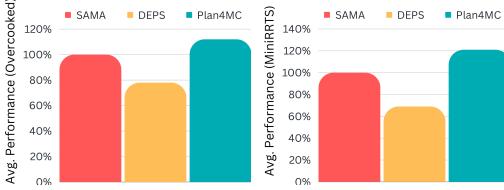


Figure 6: **Left:** Average performance of PLM-based planners in all tasks in Overcooked. **Right:** Average win rate of PLM-based planners in MiniRTS.

replacing the SAMA process with DEPS or Plan4MC.

In particular, DEPS generates a complete goal sequence at the onset of the task. Subsequently, the reflection mechanism adjusts the goal sequence based on the policy execution outcomes. Plan4MC determines the goal to be generated by searching the pre-constructed skill graph at each round of goal generation, a paradigm also adopted by SAMA. It is important to note that both baselines were designed for single-agent tasks (i.e., Minecraft). Hence, to ensure a fair comparison, we employ the goal decomposition in SAMA and subsequent processes after goal generation in DEPS or Plan4MC. In other words, we only replace the goal generation procedure in SAMA with DEPS or Plan4MC.

As illustrated in Figure 6, DEPS generates an entire goal sequence at once, which significantly increases the likelihood of PLM errors, leading to a decline in performance. Plan4MC, which generates goals round-by-round and incorporates graph search technology, exhibits a slight performance improvement. However, it is crucial to recognize that within SAMA’s framework, goal generation constitutes only one of several modules.

4.4 Additional Experimental Environments

To further validate the generality of our proposed framework (SAMA), we expanded our experimental evaluation by incorporating two additional environments: Pig Chase from the Malmo Collaborative AI Challenge and Pressure Plate. These environments complement the existing tasks by introducing different types of cooperation and task decomposition challenges.

4.4.1 Pig Chase

Originally designed as a mixed-motive social dilemma (a variant of the stag hunt), Pig Chase involves two Minecraft agents collaborating to corner a pig within a small meadow. In our adaptation, we modified the environment to better align with fully cooperative tasks. Our modification assigns each of the two agents a separate pig to catch simultaneously, effectively removing the competitive aspect of the original design. Furthermore, we removed the exit gates to emphasize task

Table 1: Success Rate Comparison

Method	Pig Chase	Pressure Plate
SAMA	100%	100%
MAPPO	100%	60%

Table 2: Training Epochs Comparison

Method	Pig Chase	Pressure Plate
SAMA	~1,560	~670
MAPPO	~2,120	~4,300

allocation and cooperative behavior between agents. In this setup, Symbolic View (a grid-based global state) is used as input for the LLM, while First-Person View (3D imagery) is used as input for MARL agents. Since no formal paper exists for this environment, we based our task manual on the official README file¹.

4.4.2 Pressure Plate

This multi-agent environment² challenges agents to traverse a gridworld partitioned into rooms. Each room contains a plate and a locked door, and agents must cooperate to progress. The environment implements a unique cooperation mechanism where each agent is assigned a specific plate that only they can activate. The core challenge lies in the requirement that to open the door to the next room, one agent must remain behind to activate their assigned plate while others proceed forward. Success in this environment is achieved when the agents collectively reach the final goal, represented by a treasure chest. We evaluated SAMA using the default four-player map, which involves four agents and requires sophisticated task allocation strategies.

4.4.3 Performance Analysis

We evaluated SAMA against MAPPO, a representative MARL baseline, using success rate over 10 rounds as the primary metric. In addition, we compared the training efficiency by analyzing the number of epochs required to achieve optimal performance.

These results (Table 1 and 2) demonstrate that SAMA achieves superior performance with significantly fewer training epochs, highlighting its sample efficiency and ability to generalize.

¹https://github.com/microsoft/malmo-challenge/blob/master/ai_challenge/pig_chase/README.md.

²<https://github.com/ueo-agents/pressureplate>.

Table 3: Performance Comparison and Optimality Gaps Across Different PLMs

Model	Cramped Rm.	Asymm. Adv.	Coord. Ring.	Forced Coord.	Counter Circ.
GPT-4	200	210	165	210	150
OpenAI-o1-mini	210	210	190	210	150
SOTA	230	230	240	170	155

4.5 Analysis of Performance Bounds

Impact of PLM Capabilities on Optimality To quantify the relationship between PLM capabilities and performance optimality, we conducted comparative experiments using OpenAI-o1-mini in Overcooked-AI. Table 3 demonstrates that while more powerful PLMs can reduce the optimality gap, they still fall short of SOTA results in most scenarios, indicating that PLM capabilities represent a significant constraint on achieving optimal performance.

Reasoning Conflicts and Optimality Constraints Through detailed case studies of incomplete sub-goals after three iterations of self-reflection, we identified a fundamental constraint in SAMA’s optimization process. The PLM’s inherent common-sense reasoning, particularly regarding real-world cooking scenarios, occasionally conflicts with the task-specific rules defined in the environment. This misalignment can lead to the generation of sub-optimal goals that, while logical in a real-world context, prove inefficient or infeasible within the constraints of the Overcooked-AI environment.

These findings provide insights into both the current performance bounds of our approach and potential pathways for optimization. The interplay between PLM capabilities and task-specific constraints emerges as a crucial factor in determining system optimality, suggesting that advances in either area could lead to significant improvements in overall system performance.

4.6 Runtime and Economic Cost

Runtime. In SAMA evaluation, a PLM query is made to generate the next goal and assign subgoals. Self-reflection may require multiple queries, but the average number per episode is relatively small (about 20). Wall time has increased by 1-2x compared to baselines.

Economic Cost for 1 Inference Episode. For both Overcooked-AI and MiniRTS, each PLM query requires around 3,000 tokens. Thus, Overcooked-AI needs about 60,000 tokens per episode ($20 \times 3,000$) at a cost of \$0.06 (GPT-3.5-turbo) or \$1.8 (GPT-4). MiniRTS needs about 3,000 tokens per episode ($1 \times 3,000$) at a cost of \$0.003 (GPT-3.5-turbo) or \$0.09 (GPT-4). The PLM output is around 1,500 tokens,

with consistent costs.

Economic Cost for Training. We averaged ten evaluations for different tasks, with each result being the average of 10 random seeds. For Overcooked-AI, each episode requires about 10 PLM queries, totaling 1,000 queries ($10 \times 10 \times 10$) and 3,000,000 tokens. The cost is \$3 (GPT-3.5-turbo) or \$90 (GPT-4). For MiniRTS, each episode requires 1 PLM query, totaling 100 queries ($10 \times 10 \times 1$) and 300,000 tokens. The cost is \$0.3 (GPT-3.5-turbo) or \$9 (GPT-4). The PLM output is around 1,500 tokens, with consistent costs.

5 Closing Remarks and Limitations

This paper presents SAMA to address the “sample scarcity” and “over-representation of goals” challenges when dealing with MARL credit assignment. SAMA leverages PLMs, enabling the generation of semantically aligned goals, supporting proper decomposition and allocation. Additionally, SAMA integrates language-grounded MARL to train lower-level policies. Our results show that SAMA is beneficial for agents involved in long-horizon, sparse-reward, and highly collaborative tasks, such as Overcooked and MiniRTS, which require common-sense behaviors that cannot extract through end-to-end learning. However, it may be less favorable in environments—where human reasoning is irrelevant or inexpressible in language.

Furthermore, the effectiveness of PLMs is dependent on prompt selection. Despite carefully crafted prompts and self-reflective mechanisms, PLMs may occasionally falter due to missing domain-specific knowledge. Multiple strategies can be employed to overcome this limitation, such as integrating domain expertise or fine-tuning on task-specific data. Moreover, the quality of recommendations significantly improves with larger model sizes. However, the repeated prompting of large-scale PLMs may be time-consuming and financially impractical. As generative models advance in fields beyond text, SAMA-inspired algorithms could be utilized to generate plausible visual goals. Therefore, SAMA has the potential to serve as a foundation for future research, fostering the development of increasingly adaptive and versatile approaches for incorporating human contextual understanding into MARL.

Acknowledgements

This work is supported by National Natural Science Foundation of China (62406270, 72495131, 62106213, 72394361), STCSM “Science and Technology Innovation Action Plan” (22QB1402100, 24YF2748800), the “Sino-German Cooperation 2.0” Funding Program of Tongji University, and an extended support project from the Shenzhen Science and Technology Program.

References

- Agogino, A. K. and Tumer, K. (2004). Unifying temporal and structural credit assignment problems. In *AAMAS*.
- Becht, M., Gurzki, T., Klarmann, J., and Muscholl, M. (1999). Rope: Role oriented programming environment for multiagent systems. In *IFCIS*.
- Bellemare, M., Srinivasan, S., Ostrovski, G., Schaul, T., Saxton, D., and Munos, R. (2016). Unifying count-based exploration and intrinsic motivation. In *NeurIPS*.
- Bengio, Y., Courville, A., and Vincent, P. (2013). Representation learning: A review and new perspectives. *IEEE transactions on pattern analysis and machine intelligence*, 35(8):1798–1828.
- Bernstein, D. S., Givan, R., Immerman, N., and Zilberstein, S. (2002). The complexity of decentralized control of markov decision processes. *Mathematics of operations research*, 27(4):819–840.
- Böhmer, W., Kurin, V., and Whiteson, S. (2020). Deep coordination graphs. In *ICML*. PMLR.
- Bommasani, R., Hudson, D. A., Adeli, E., Altman, R., Arora, S., von Arx, S., Bernstein, M. S., Bohg, J., Bosselut, A., Brunskill, E., et al. (2021). On the opportunities and risks of foundation models. *arXiv preprint arXiv:2108.07258*.
- Burda, Y., Edwards, H., Storkey, A., and Klimov, O. (2019). Exploration by random network distillation. In *ICLR*.
- 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. In *NeurIPS*.
- Charakorn, R., Manoonpong, P., and Dilokthanakul, N. (2020). Investigating partner diversification methods in cooperative multi-agent deep reinforcement learning. In *ICONIP*.
- Chase, H. (2022). Langchain. <https://github.com/hwchase17/langchain>. Released on 2022-10-17.
- Chen, X., Lin, M., Schärli, N., and Zhou, D. (2023). Teaching large language models to self-debug. *arXiv preprint arXiv:2304.05128*.
- Colas, C., Karch, T., Sigaud, O., and Oudeyer, P.-Y. (2022). Autotelic agents with intrinsically motivated goal-conditioned reinforcement learning: a short survey. *Journal of Artificial Intelligence Research*, 74:1159–1199.
- Devlin, S., Yliniemi, L., Kudenko, D., and Tumer, K. (2014). Potential-based difference rewards for multiagent reinforcement learning. In *AAMAS*.
- 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 *ICML*.
- Du, Y., Han, L., Fang, M., Liu, J., Dai, T., and Tao, D. (2019). Liir: Learning individual intrinsic reward in multi-agent reinforcement learning. In *NeurIPS*.
- Du, Y., Watkins, O., Wang, Z., Colas, C., Darrell, T., Abbeel, P., Gupta, A., and Andreas, J. (2023). Guiding pretraining in reinforcement learning with large language models. In *ICML*.
- Ecoffet, A., Huizinga, J., Lehman, J., Stanley, K. O., and Clune, J. (2021). First return, then explore. *Nature*, 590(7847):580–586.
- Ellis, B., Moalla, S., Samvelyan, M., Sun, M., Mahajan, A., Foerster, J. N., and Whiteson, S. (2022). Smacv2: An improved benchmark for cooperative multi-agent reinforcement learning. *arXiv preprint arXiv:2212.07489*.
- Foerster, J., Farquhar, G., Afouras, T., Nardelli, N., and Whiteson, S. (2018). Counterfactual multi-agent policy gradients. In *AAAI*.
- Grote, E.-M., Pfeifer, S. A., Röltgen, D., Kühn, A., and Dumitrescu, R. (2020). Towards defining role models in advanced systems engineering. In *ISSE*.
- Guo, Z., Thakoor, S., Pišlar, M., Avila Pires, B., Altché, F., Tallec, C., Saade, A., Calandriello, D., Grill, J.-B., Tang, Y., et al. (2022). Byol-explore: Exploration by bootstrapped prediction. In *NeurIPS*.
- Gupta, T., Mahajan, A., Peng, B., Böhmer, W., and Whiteson, S. (2021). Uneven: Universal value exploration for multi-agent reinforcement learning. In *ICML*.
- Hanjie, A. W., Zhong, V. Y., and Narasimhan, K. (2021). Grounding language to entities and dynamics for generalization in reinforcement learning. In *ICML*.

- Hao, J., Yang, T., Tang, H., Bai, C., Liu, J., Meng, Z., Liu, P., and Wang, Z. (2023). Exploration in deep reinforcement learning: From single-agent to multiagent domain. *IEEE Transactions on Neural Networks and Learning Systems*.
- Hernandez-Leal, P., Kartal, B., and Taylor, M. E. (2019). A survey and critique of multiagent deep reinforcement learning. *Autonomous Agents and Multi-Agent Systems*, 33(6):750–797.
- Hu, H., Yarats, D., Gong, Q., Tian, Y., and Lewis, M. (2019). Hierarchical decision making by generating and following natural language instructions. In *NeurIPS*.
- Hua, Y., Gao, S., Li, W., Jin, B., Wang, X., and Zha, H. (2023). Learning optimal “pigovian tax” in sequential social dilemmas. In *AAMAS*.
- 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*.
- Jeon, J., Kim, W., Jung, W., and Sung, Y. (2022). Maser: Multi-agent reinforcement learning with sub-goals generated from experience replay buffer. In *ICML*.
- Kang, Y., Wang, T., Yang, Q., Wu, X., and Zhang, C. (2022). Non-linear coordination graphs. In *NeurIPS*.
- Knott, P., Carroll, M., Devlin, S., Ciosek, K., Hofmann, K., Dragan, A., and Shah, R. (2021). Evaluating the robustness of collaborative agents. In *AAMAS*.
- Kulkarni, T. D., Narasimhan, K., Saeedi, A., and Tenenbaum, J. (2016). Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation. In *NeurIPS*.
- Kwon, M., Xie, S. M., Bullard, K., and Sadigh, D. (2023). Reward design with language models. In *ICLR*.
- Lhaksmana, K. M., Murakami, Y., and Ishida, T. (2018). Role-based modeling for designing agent behavior in self-organizing multi-agent systems. *International Journal of Software Engineering and Knowledge Engineering*, 28(01):79–96.
- Li, C., Wang, T., Wu, C., Zhao, Q., Yang, J., and Zhang, C. (2021a). Celebrating diversity in shared multi-agent reinforcement learning. In *NeurIPS*.
- Li, J., Kuang, K., Wang, B., Liu, F., Chen, L., Fan, C., Wu, F., and Xiao, J. (2022). Deconfounded value decomposition for multi-agent reinforcement learning. In *ICML*.
- Li, W., Wang, X., Jin, B., Lu, J., and Zha, H. (2023). Learning roles with emergent social value orientations. *arXiv preprint arXiv:2301.13812*.
- Li, W., Wang, X., Jin, B., Sheng, J., Hua, Y., and Zha, H. (2021b). Structured diversification emergence via reinforced organization control and hierachical consensus learning. In *AAMAS*.
- Li, Y., Zhang, S., Sun, J., Du, Y., Wen, Y., Wang, X., and Pan, W. (2021c). Cooperative open-ended learning framework for zero-shot coordination. In *ICML*.
- Ma, Y. J., Liang, W., Wang, G., Huang, D.-A., Bastani, O., Jayaraman, D., Zhu, Y., Fan, L., and Anandkumar, A. (2024). Eureka: Human-level reward design via coding large language models. In *ICLR*.
- Madaan, A., Tandon, N., Gupta, P., Hallinan, S., Gao, L., Wiegreffe, S., Alon, U., Dziri, N., Prabhumoye, S., Yang, Y., et al. (2023). Self-refine: Iterative refinement with self-feedback. *arXiv preprint arXiv:2303.17651*.
- Mahajan, A., Rashid, T., Samvelyan, M., and Whiteson, S. (2019). Maven: Multi-agent variational exploration. In *NeurIPS*.
- Mao, H., Gong, Z., and Xiao, Z. (2020). Reward design in cooperative multi-agent reinforcement learning for packet routing. *arXiv preprint arXiv:2003.03433*.
- Mesnard, T., Weber, T., Viola, F., Thakoor, S., Saade, A., Harutyunyan, A., Dabney, W., Stepleton, T. S., Heess, N., Guez, A., et al. (2021). Counterfactual credit assignment in model-free reinforcement learning. In *ICML*.
- Min, H., Jung, J., Kim, S., Kim, B., and Heo, J. (2018). Role-based automatic programming framework for interworking a drone and wireless sensor networks. In *SAC*.
- Nguyen, D., Nguyen, P., Venkatesh, S., and Tran, T. (2022). Learning to transfer role assignment across team sizes. In *AAMAS*.
- Nguyen, D. T., Kumar, A., and Lau, H. C. (2018). Credit assignment for collective multiagent rl with global rewards. In *NeurIPS*.
- OpenAI (2022). Introducing ChatGPT. <https://openai.com/blog/chatgpt/>. Accessed on March 11, 2025.
- OpenAI (2023). GPT-4. <https://openai.com/research/gpt-4>. Accessed on March 11, 2025.

- Pathak, D., Agrawal, P., Efros, A. A., and Darrell, T. (2017). Curiosity-driven exploration by self-supervised prediction. In *ICML*.
- Peshkin, L., Kim, K.-E., Meuleau, N., and Kaelbling, L. P. (2000). Learning to cooperate via policy search. In *UAI*.
- Phan, T., Ritz, F., Belzner, L., Altmann, P., Gabor, T., and Linnhoff-Popien, C. (2021). Vast: Value function factorization with variable agent sub-teams. In *NeurIPS*.
- Rashid, T., Farquhar, G., Peng, B., and Whiteson, S. (2020a). Weighted qmix: Expanding monotonic value function factorisation for deep multi-agent reinforcement learning. In *NeurIPS*.
- Rashid, T., Samvelyan, M., De Witt, C. S., Farquhar, G., Foerster, J., and Whiteson, S. (2020b). Monotonic value function factorisation for deep multi-agent reinforcement learning. *The Journal of Machine Learning Research*, 21(1):7234–7284.
- Samvelyan, M., Rashid, T., Schroeder de Witt, C., Farquhar, G., Nardelli, N., Rudner, T. G., Hung, C.-M., Torr, P. H., Foerster, J., and Whiteson, S. (2019). The starcraft multi-agent challenge. In *AAMAS*.
- Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. (2017). Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*.
- Shao, J., Lou, Z., Zhang, H., Jiang, Y., He, S., and Ji, X. (2022). Self-organized group for cooperative multi-agent reinforcement learning. In *NeurIPS*.
- Shinn, N., Cassano, F., Gopinath, A., Narasimhan, K. R., and Yao, S. (2023). Reflexion: Language agents with verbal reinforcement learning. In *NeurIPS*.
- Son, K., Kim, D., Kang, W. J., Hostallero, D. E., and Yi, Y. (2019). Qtran: Learning to factorize with transformation for cooperative multi-agent reinforcement learning. In *ICML*.
- Strouse, D., McKee, K., Botvinick, M., Hughes, E., and Everett, R. (2021). Collaborating with humans without human data. In *NeurIPS*.
- Sunehag, P., Lever, G., Gruslys, A., Czarnecki, W. M., Zambaldi, V., Jaderberg, M., Lanctot, M., Sonnerat, N., Leibo, J. Z., Tuyls, K., et al. (2018). Value-decomposition networks for cooperative multi-agent learning based on team reward. In *AAMAS*.
- Tesauro, G. (1994). Td-gammon, a self-teaching backgammon program, achieves master-level play. *Neural computation*, 6(2):215–219.
- Varshney, N., Yao, W., Zhang, H., Chen, J., and Yu, D. (2023). A stitch in time saves nine: Detecting and mitigating hallucinations of llms by validating low-confidence generation. *arXiv preprint arXiv:2307.03987*.
- Wang, J., Ren, Z., Han, B., Ye, J., and Zhang, C. (2021a). Towards understanding cooperative multi-agent q-learning with value factorization. In *NeurIPS*.
- Wang, L., Zhang, Y., Hu, Y., Wang, W., Zhang, C., Gao, Y., Hao, J., Lv, T., and Fan, C. (2022a). Individual reward assisted multi-agent reinforcement learning. In *ICML*.
- Wang, T., Dong, H., Lesser, V., and Zhang, C. (2020). Roma: multi-agent reinforcement learning with emergent roles. In *ICML*.
- Wang, T., Gupta, T., Mahajan, A., Peng, B., Whiteson, S., and Zhang, C. (2021b). {RODE}: Learning roles to decompose multi-agent tasks. In *ICLR*.
- Wang*, T., Wang*, J., Wu, Y., and Zhang, C. (2020). Influence-based multi-agent exploration. In *ICLR*.
- Wang, X., Chen, H., Tang, S., Wu, Z., and Zhu, W. (2022b). Disentangled representation learning. *arXiv preprint arXiv:2211.11695*.
- 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. In *NeurIPS*.
- Wei, J., Wang, X., Schuurmans, D., Bosma, M., Xia, F., Chi, E. H., Le, Q. V., Zhou, D., et al. (2022). Chain-of-thought prompting elicits reasoning in large language models. In *NeurIPS*.
- Wu, Y., Min, S. Y., Prabhumoye, S., Bisk, Y., Salakhutdinov, R., Azaria, A., Mitchell, T., and Li, Y. (2023). SPRING: Studying papers and reasoning to play games. In *NeurIPS*.
- Xie, T., Zhao, S., Wu, C. H., Liu, Y., Luo, Q., Zhong, V., Yang, Y., and Yu, T. (2024). Text2reward: Dense reward generation with language models for reinforcement learning. In *ICLR*.
- Xu, S., Wang, H., and Wu, Y. (2022). Grounded reinforcement learning: Learning to win the game under human commands. In *NeurIPS*.
- Yang, J., Jin, H., Tang, R., Han, X., Feng, Q., Jiang, H., Yin, B., and Hu, X. (2023). Harnessing the power of llms in practice: A survey on chatgpt and beyond. *arXiv preprint arXiv:2304.13712*.

- Yang, M., Zhao, J., Hu, X., Zhou, W., Zhu, J., and Li, H. (2022a). Ldsa: Learning dynamic subtask assignment in cooperative multi-agent reinforcement learning. In *NeurIPS*.
- Yang, Q., Dong, W., Ren, Z., Wang, J., Wang, T., and Zhang, C. (2022b). Self-organized polynomial-time coordination graphs. In *ICML*.
- Yang, Y., Wen, Y., Wang, J., Chen, L., Shao, K., Mguni, D., and Zhang, W. (2020). Multi-agent determinantal q-learning. In *ICML*.
- Yao, S., Zhao, J., Yu, D., Shafran, I., Narasimhan, K. R., and Cao, Y. (2023). React: Synergizing reasoning and acting in language models. In *ICLR*.
- 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. In *NeurIPS*.
- Yu, W., Gileadi, N., Fu, C., Kirmani, S., Lee, K.-H., Arenas, M. G., Chiang, H.-T. L., Erez, T., Hasenclever, L., Humplik, J., et al. (2023). Language to rewards for robotic skill synthesis. In *CoRL*.
- Yuan, H., Zhang, C., Wang, H., Xie, F., Cai, P., Dong, H., and Lu, Z. (2023). Plan4mc: Skill reinforcement learning and planning for open-world minecraft tasks. *arXiv preprint arXiv:2303.16563*.
- Zhang, T., Li, Y., Wang, C., Xie, G., and Lu, Z. (2021). Fop: Factorizing optimal joint policy of maximum-entropy multi-agent reinforcement learning. In *ICML*.
- Zheng, L., Chen, J., Wang, J., He, J., Hu, Y., Chen, Y., Fan, C., Gao, Y., and Zhang, C. (2021). Episodic multi-agent reinforcement learning with curiosity-driven exploration. In *NeurIPS*.
- Zhou, M., Liu, Z., Sui, P., Li, Y., and Chung, Y. Y. (2020). Learning implicit credit assignment for cooperative multi-agent reinforcement learning. In *NeurIPS*.

Checklist

1. For all models and algorithms presented, check if you include:
 - (a) A clear description of the mathematical setting, assumptions, algorithm, and/or model. **Yes. Please refer to Section 2, Section 3 and Appendix F.**
 - (b) An analysis of the properties and complexity (time, space, sample size) of any algorithm. **Yes. Please refer to Section 4.6 and Appendix E.**
 - (c) (Optional) Anonymized source code, with specification of all dependencies, including external libraries. **Yes. The code is available at <https://anonymous.4open.science/r/SAMA/>.**
2. For any theoretical claim, check if you include:
 - (a) Statements of the full set of assumptions of all theoretical results. **Not Applicable.**
 - (b) Complete proofs of all theoretical results. **Not Applicable.**
 - (c) Clear explanations of any assumptions. **Not Applicable.**
3. For all figures and tables that present empirical results, check if you include:
 - (a) The code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL). **Yes. Please refer to Appendix D, Appendix E, Appendix F and the code repository <https://anonymous.4open.science/r/SAMA/>.**
 - (b) All the training details (e.g., data splits, hyperparameters, how they were chosen). **Yes. Please refer to Appendix E.**
 - (c) A clear definition of the specific measure or statistics and error bars (e.g., with respect to the random seed after running experiments multiple times). **Yes. Please refer to Section 4, Appendix C, Appendix D, Appendix E, Appendix G, Appendix H and Appendix I.**
 - (d) A description of the computing infrastructure used. (e.g., type of GPUs, internal cluster, or cloud provider). **Yes. Please refer to Section 4.6 and Appendix E.**
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets, check if you include:
 - (a) Citations of the creator If your work uses existing assets. **Yes. Please refer to Appendix D and Appendix E.**
 - (b) The license information of the assets, if applicable. **Yes. Please refer to Appendix D and Appendix E.**
 - (c) New assets either in the supplemental material or as a URL, if applicable. **Yes. The code is available at <https://anonymous.4open.science/r/SAMA/>.**
 - (d) Information about consent from data providers/curators. **Not Applicable.**
 - (e) Discussion of sensible content if applicable, e.g., personally identifiable information or offensive content. **Not Applicable.**
5. If you used crowdsourcing or conducted research with human subjects, check if you include:
 - (a) The full text of instructions given to participants and screenshots. **Not Applicable.**
 - (b) Descriptions of potential participant risks, with links to Institutional Review Board (IRB) approvals if applicable. **Not Applicable.**
 - (c) The estimated hourly wage paid to participants and the total amount spent on participant compensation. **Not Applicable.**

Supplementary Material

A Related Work

A.1 Tackeling Credit Assignment in MARL

The challenge of credit assignment arises not only on temporal scales but also on structural scales. *Value decomposition*³ serves as the primary paradigm for addressing structural credit assignment (Devlin et al., 2014; Nguyen et al., 2018; Foerster et al., 2018; Sunehag et al., 2018; Rashid et al., 2020b; Son et al., 2019; Rashid et al., 2020a; Wang et al., 2021a; Böhmer et al., 2020; Kang et al., 2022). Consequently, most studies tackling temporal and structural credit assignment problems employ the “value decomposition+x” framework.

One prevalent example of “x” involves learning with efficient exploration, which shows efficacy in task selection (Mahajan et al., 2019; Yang et al., 2020; Zheng et al., 2021; Li et al., 2021a; Zhang et al., 2021; Gupta et al., 2021). Subgoal-based methodologies have recently emerged as a viable alternative inspired by approaches utilizing subgoals in single-agent RL (Kulkarni et al., 2016; Bellemare et al., 2016; Pathak et al., 2017; Burda et al., 2019; Ecoffet et al., 2021; Guo et al., 2022) (also referred to as intrinsically motivated RL (Colas et al., 2022)). Intuitively, these methods decompose a task into a series of goals while concurrently breaking down each goal into subgoals that require completion by agents. In doing so, both time and structural scales receive dense, goal-directed rewards, mitigating the credit assignment dilemma.

A.2 Subgoal-based Multi-Agent Reinforcement Learning

Numerous antecedent studies have delved into subgoal-based MARL characterized by sparse rewards. Early solutions primarily rely on artificially predefined rules (Becht et al., 1999; Lhaksmana et al., 2018; Min et al., 2018; Grote et al., 2020), which limits their dynamism and adaptability in new environments. Consequently, *automatic subgoal generation* (ASG) serves as a more practical implementation of “x,” permitting application to various tasks without relying on domain knowledge.

The early endeavors of ASG primarily revolved around a semi-end-to-end framework, wherein the generation of subgoals occurred automatically, but the lower-level policies fulfilling these subgoals were pre-trained. HD-MARL (Tang et al., 2018) introduces a hierarchical deep MARL framework devised to tackle sparse-reward challenges via temporal abstraction. HDMARL selects a singular subgoal from a predefined assortment contingent upon domain-specific knowledge; consequently, such subgoal construction proves inapplicable across disparate tasks. Works by these papers (Xiao et al., 2020, 2022) impose further constraints by necessitating the pre-definition of goal-conditioned policies corresponding with each respective goal.

Predominant ASG methodologies predominantly integrate resolutions for the four identified problems ((**a-d**) in Section 1) within a bifurcated, end-to-end learning process. Initially, agent-specific subgoals are conceived, succeeded by learning policies conducive to realizing these subgoals. These stratagems can be broadly compartmentalized into dual classifications contingent upon subgoal representation. Specific methods employ embeddings to encode subgoals, facilitating their generation via local observations (Yang et al., 2020a; Wang et al., 2020, 2021b), local message-passing (Li et al., 2021b; Shao et al., 2022), or selecting from a pre-formulated subgoal compendium utilizing global information (Yang et al., 2022a). After this, goal-conditioned policies are learned through standard reward maximization. Conversely, alternative approaches implicitly convert subgoals into intrinsic rewards, refining policies endorsing subgoal fruition by maximizing these shaped rewards (Phan et al., 2021; Jeon et al., 2022; Nguyen et al., 2022; Yang et al., 2022b; Li et al., 2023).

³In the study by this paper (Wang et al., 2021a), a theoretical demonstration is provided regarding the implicit nature of counterfactual difference rewards in individual value decomposition. Owing to this, we propose to include such classic mechanisms for managing structural credit assignment challenges under the broad classification of value decomposition, albeit with some degree of flexibility.

A.3 Language Grounded Reinforcement Learning

Language grounding encompasses the acquisition of natural languages unit comprehension, such as utterances, phrases, or words, by capitalizing on non-linguistic contexts. Numerous antecedent studies have addressed the conveyance of goals or instructions to agents through text, eliciting agent behavior in response to language grounding (Branavan et al., 2012; Janner et al., 2018; Wang et al., 2019; Blukis et al., 2020; Küttler et al., 2020; Tellex et al., 2020; Xu et al., 2022), thus establishing a robust correlation between the provided instructions and the executed policy. Recently, a plethora of research has delved into generalizations from multifaceted vantage points. Series of works (Hill et al., 2020a,b, 2021) scrutinized the generalization concerning novel entity combinations, extending from synthetic template commands to human-generated natural instructions and object quantities. Choi et al. (Choi et al., 2021) introduced a language-guided policy learning algorithm, facilitating expeditious task learning through linguistic adjustments. Moreover, this paper (Co-Reyes et al., 2019) advocated using language guidance for policies to achieve generalization in novel tasks by employing meta-learning techniques. An alternative line of inquiry has emphasized the incorporation of task manuals as supplementary information to bolster generalization (Narasimhan et al., 2018; Zhong et al., 2020; Hanjie et al., 2021; Zhong et al., 2021). In contrast to the research mentioned above, EnDi (Ding et al., 2023) advances the discourse by examining language grounding at the entity level within multi-agent environments.

A.4 Foundation Models for Decision Making

Foundation models, trained through extensive datasets, have demonstrated remarkable aptitudes in conjunction with fast adaptability for a plethora of downstream tasks in diverse fields such as vision (Yuan et al., 2021), language (Kenton and Toutanova, 2019; Brown et al., 2020), and cross-modalities (Ramesh et al., 2021; Jiang et al., 2022; Alayrac et al., 2022). Capitalizing on such capabilities, these models have been employed to furnish RL agents with rewards (Gupta et al., 2022; Fan et al., 2022; Kwon et al., 2023; Xie et al., 2024; Ma et al., 2024; Rocamonde et al., 2024); a burgeoning line of research is now investigating using foundation models (particularly PLMs) for refining agent policies. In instances where agent actions are delineated through natural language, PLMs may be employed to devise more sophisticated strategies for long-horizon tasks, as linguistic descriptions of actions are anticipated to exhibit enhanced generalization than low-level controls (Huang et al., 2022b,a; Brohan et al., 2023; Wang et al., 2023; Dasgupta et al., 2023; Carta et al., 2023; Gur et al., 2024; Anonymous, 2024b,a; Zhang et al., 2024b; Hong et al., 2024; Tan et al., 2024; Dalal et al., 2024; Chan et al., 2024; Chen et al., 2024; Zhang et al., 2024a). Furthermore, when agent observations encompass imagery and textual descriptions, vision-language captioning models can augment agent observations with linguistic delineations (Tam et al., 2022; Du et al., 2023; Driess et al., 2023; Zheng et al., 2024; Rocamonde et al., 2024; Du et al., 2024). Remarkably, even in situations where agent states, actions, and rewards do not encompass visual or textual elements, PLMs have been identified as advantageous policy initializers for both offline (Reid et al., 2022; Shi et al., 2024) and online RL (Li et al., 2022; Lin et al., 2024).

B Intrinsically Motivated Goal-Conditioned RL

Owing to the intimate correlation between the ASG framework and intrinsically motivated goal-conditioned RL (IMGC-RL; Colas et al., 2022), a succinct exposition of the IMGC-RL is presented herein. Furthermore, this section delves into the distinctions and connections between the SAMA framework and the IMGC-RL paradigm. The proposal of the IMGC-RL framework primarily endeavors to address tasks characterized by temporal-extended, sparse or delayed rewards. By incessantly generating transitional phase subgoals and devising the corresponding intrinsic reward functions, the framework dissects the original task into a succession of subordinate tasks with shorter-horizons and dense rewards, thereby simplifying problem resolution. However, it is imperative to underscore that the IMGC-RL pertains to a single-agent setting, thereby exhibiting a considerable discrepancy when compared to MARL.

From Figure 7, it is discernible that the IMGC-RL framework primarily consists of two modules: the goal generator and the goal-conditioned reinforcement learning. The former may be further subdivided into three submodules, namely goal representation learning, goal support set learning, and goal sampling; the latter encompasses goal-conditioned reward design and goal-conditioned RL policy learning. For an in-depth examination of each module and its submodules, along with related works, kindly refer to this comprehensive review paper (Colas et al., 2022). Furthermore, in categorizing existing MARL methodologies based on the ASG framework from the perspective

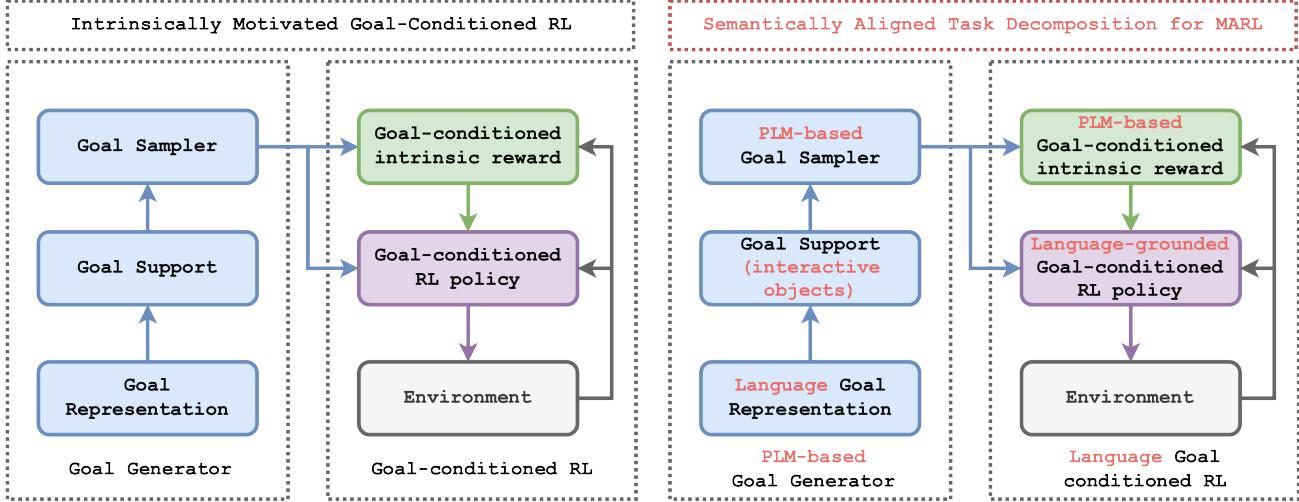


Figure 7: **Left:** The illustration diagram of intrinsically motivated goal-conditioned reinforcement learning (IMGC-RL); **Right:** The proposed SAMA framework under the IMGC-RL context.

of IMGC-RL, one may broadly distinguish two types according to the presence of goal-conditioned RL: one in which a pre-trained policy stemming from imitation learning replaces the goal-conditioned RL module (Tang et al., 2018; Xiao et al., 2020, 2022), and the other prevalent methodologies incorporates both, employing either a two-phase or end-to-end learning paradigm (Yang et al., 2020a; Wang et al., 2020, 2021b; Yang et al., 2022a; Phan et al., 2021; Jeon et al., 2022; Nguyen et al., 2022; Yang et al., 2022b; Li et al., 2023).

Connection with the SAMA. SAMA bears a remarkably close connection to IMGC-RL. Figure 7 delineates their intricate correspondence in detail. Initially, SAMA renders subgoals as text whilst constraining the subgoal space or support by extracting interactive entities from the environment. Subsequently, SAMA implements goal sampling and the design of goal-conditioned reward functions by prompting PLMs. Ultimately, SAMA accomplishes the training of the goal-conditioned RL policy by incorporating language grounding techniques.

C Missing Discussions

C.1 Self-Reflection

Prevalent PLMs frequently yield to typical errors, including hallucination or arbitrary decision-making, attributable to the incapacity to assimilate knowledge from long trajectories to refine subsequent planning. We adopt a self-reflection mechanism for discerning hallucination and suboptimal planning (Shinn et al., 2023). Employing this technique, the PLM planner will rectify (as necessitated) the semantically aligned goal generation, decomposition, or subgoal allocation based on the outcomes derived from the subgoal achievement of language-grounded MARL.

As shown in Figure 1 and Figure 8, the self-reflection module contains two core steps, namely evaluation and reflection. Concretely, after the goal generation, decomposition and subgoal assignment, the language-grounded agent i carries out a sequence of actions in response to the assigned subgoal g_k^i at each round k . At each timestep t_k , the agent i executes an action a_{i,t_k} and the state transits to o_{i,t_k+1} . The evaluator, or the code snippet in \mathcal{D}_{code} corresponding to the subgoal g_k^i , will determine whether g_k^i has been completed. SAMA then calculates a heuristic h predicated on the binary return value, potentially eliciting self-reflection. The application of heuristic h in this paper is uncomplicated: if the return values *all* equal 1 (indicating successful execution of the subgoal), self-reflection will not be prompted; conversely, it will be invoked.

If h suggests self-reflection, SAMA prompts the PLM planner to deliberate on its extant semantically aligned goal generation, decomposition, or assignment following self-reflective memory and environmental state. Since PLM needs to regenerate goals or subgoals during the self-reflection, we need to recovery the environment to the previous state. For this, we record the subgoals generated sequentially during the task planning. In this way, the state can be recovered by resetting the environment and calling the lower-level language-grounded

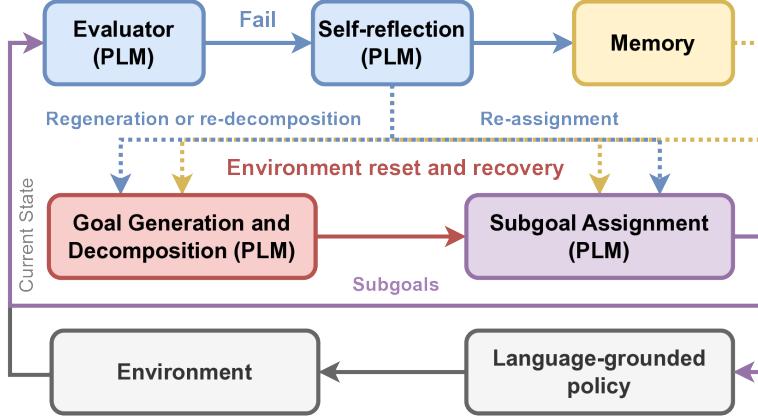


Figure 8: One self-reflection trail.

policy conditioned on the logged subgoal sequence. The above “reset-recovery” procedure is denominated as a *self-reflection trial*. Otherwise, SAMA prompts the PLM planner for subsequent semantically aligned planning. Since most MARL tasks are episodic and language-grounded MARL policy is fixed in evaluation, the above reset-recovery process is doable. In practice, we set a hyperparameter limit of a maximum of 3 reflections retained within the agent’s history. The trial is discontinued if the agent surpasses the utmost quantity (we also set it to 3 in the following experiments). In quantifying the empirical performance, the instances encountered during self-reflection trials are not incorporated.

C.2 PLM-based Baselines

There are some PLM-based decision-making methods bear the closest resemblance to our work: DEPS (Wang et al., 2023), ELLM (Du et al., 2023), Plan4MC (Yuan et al., 2023) and GITM (Zhu et al., 2023). DEPS (Wang et al., 2023) engenders the whole goal sequence via prompt PLM, and the goal-conditioned policy is ascertained through imitation learning in advance. In contrast, ELLM (Du et al., 2023), Plan4MC (Yuan et al., 2023) and GITM (Zhu et al., 2023), akin to SAMA, prompts the PLM to generate the subsequent goal contingent on the current state. ELLM (Du et al., 2023) trains goal-conditioned RL policy through synthetically devised goal-oriented, dense intrinsic rewards, Plan4MC (Yuan et al., 2023) and GITM (Zhu et al., 2023), similar with DEPS, adopt the pre-trained or pre-defined goal-conditioned policy to achieve goals.

SAMA transcends a mere augmentation of these works within multi-agent contexts, showcasing marked innovation in following three aspects:

Refined and Automated Preprocessing. In contrast to prevailing approaches reliant on manual translations or those confined to specific translated environments, such as Minedojo (Fan et al., 2022), SAMA presents a comprehensive procedural framework for environmental translation, incorporating the generation of task manuals, state, and action translations. Furthermore, by incorporating code understanding technology (Chase, 2022), SAMA alleviates the training complexities of language-grounded MARL and enhances self-reflection quality. This refined and automated preprocessing enables SAMA to transition to novel tasks at a less expense.

Introspective Goal Decomposition. Devising explicit subgoals for singular agents in sparse-reward multi-agent tasks optimally remains an untenable endeavor for humans. Generally, human aptitude is demonstrated in devising strategies for the aggregate system. Consequently, it can be logically deduced that PLMs, endowed with human knowledge, exhibit similar tendencies. Thus, SAMA deliberately orchestrates a *goal decomposition* (over multiple agents) phase. Moreover, SAMA incorporates a self-reflection mechanism to ameliorate PLM limitations in long-horizon reasoning.

Automated Goal-conditioned Policy Optimization. DEPS, Plan4MC and GITM, reliant on imitation-learning-based or predefied skills, or goal-conditioned policy, necessitates extensive expert data encompassing an array of achieved goals. However, the unbounded nature of the goal space emitted by PLM renders it challenging for these method to be directly applied to alternative tasks; ELLM, contingent on the synthetic crafting of intrinsic rewards, engenders its complexities when extending to a multi-agent context. Reward design constitutes

a longstanding and contentious issue in MARL, as both global rewards and local rewards have been shown to be inherently flawed: the former may foster indolence in agents, while the latter might yield egoistic agents (Du et al., 2019; Mao et al., 2020; Wang et al., 2022a; Hua et al., 2023). In contrast, SAMA leverages language-grounded MARL for the *automatic* optimization of a (language-based) goal-conditioned policy.

D Environment Details

D.1 Overcooked

In this section, we carry out an array of experiments in the **Overcooked** environment (Carroll et al., 2019; Charakorn et al., 2020; Knott et al., 2021; Li et al., 2021c), specifically devised to address sparse-reward, long-horizon, and coordination conundrums. In this dyadic common payoff game, each participant manipulates a chef within a culinary setting, collaboratively preparing and delivering soup, thereby accruing a shared reward of 20 for the team. The **Overcooked** domain we employed encompasses five distinct configurations (refer to Figure 9), namely **Cramped Room**, **Asymmetric Advantages**, **Coordination Ring**, **Forced Coordination**, and **Counter Circuit**. The comprehensive elucidation of the five configurations is presented below (Li et al., 2021c):

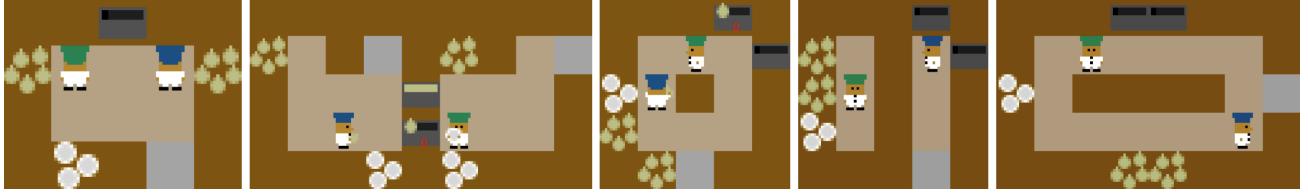


Figure 9: Screenshots of the **Overcooked** layout. From left to right: Cramped Room, Asymmetric Advantages, Coordination Ring, Forced Coordination, and Counter Circuit.

- **Cramped Room.** The cramped room constitutes a rudimentary setting wherein two participants are confined to a diminutive space possessing a singular pot (black box with gray base) and one serving point (light gray square). Consequently, individuals are anticipated to exploit the pot to its fullest and effectually dispense soup, even with elementary collaboration.
- **Asymmetric Advantages.** Within this arrangement, two players are situated in separate, isolated kitchens. As the appellation implies, the locations of onions, pots, and serving points exhibit asymmetry. In the left kitchen, onions are remote from the pots, while serving points are proximal to the central region of the configuration. In contrast, within the right kitchen, onions are positioned near the central area, and the serving points are distant from the pots.
- **Coordination Ring.** This annular configuration necessitates both participants to maintain constant motion to avoid impeding one another, particularly in the top-right and bottom-left vertices where the onions and pots are situated. To achieve optimal synergy, both pots should be employed.
- **Forced Coordination.** The Forced Coordination layout represents another separation of the two agents. The left side lacks pots or serving points, while the right side is devoid of onions or pots. As a result, the pair must synchronously orchestrate their actions to accomplish the task. The left player is expected to prepare onions and plates, while the right player oversees cooking and serving responsibilities.
- **Counter Circuit.** The Counter Circuit is an additional annular configuration but with an expanded cartographical scope. Within this layout, pots, onions, plates, and serving points occupy four distinct orientations. Constrained by the constricted passageways, players are prone to obstruction. Therefore, coordination and task execution prove challenging in this environment. Individuals must acquire the advanced technique of positioning onions in the central zone to facilitate rapid exchange, thereby augmenting performance.

We selected **Overcooked** as a focal case due to the facile transformation of its state space, action space, and reward function into comprehensible natural language text via pre-established scripts. Furthermore, the environment state information readily permits the discernment of successful subgoal completion. Consequently, this facilitates the expeditious training of language-grounded MARL policies and the integration of self-reflection mechanisms.

D.2 MiniRTS



Figure 10: Screenshots of the MiniRTS (Hu et al., 2019; Xu et al., 2022), which is a real-time strategy game where the player in blue needs to control its units to kill the enemy units in red.

The ensuing content is gleaned from MiniRTS (Hu et al., 2019) and RED (Xu et al., 2022); for a comprehensive elucidation, one is encouraged to consult the source manuscript.

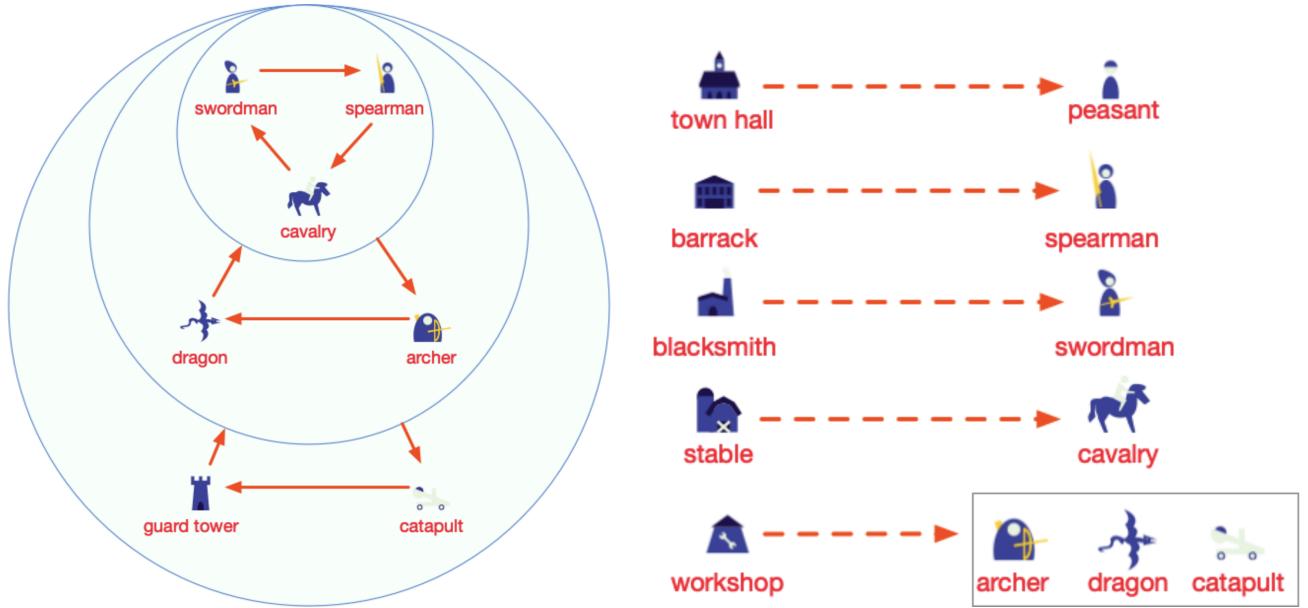


Figure 11: **Left:** Depicted in the screenshots (of RED) are the intricate attack dynamics within MiniRTS. Utilizing a rock-paper-scissors attack graph schema, each army type exhibits distinct strengths and vulnerabilities against opposing units. For instance, the “swordsman” efficaciously subdues the “spearman”; conversely, the former is susceptible to the prowess of the “cavalry.” The triumvirate of “swordsman,” “spearman,” and “cavalry” maintains a superior offensive stance against the “archer.” **Right:** Various building units generate diverse army units employing resource utilization (in RED). The “workshop,” for example, engenders the creation of “archer,” “dragon,” and “catapult” units, while alternative buildings yield a singular unit type. Exclusive to the “peasant” is the capability to extract resources from resource units and erect additional building units.

Units and Map. MiniRTS encompasses three unit classifications: resource, building, and army units, as shown in Figure 10. *Resource* units are stationary, invariably present at the game’s commencement, and not producible by any entity. Solely “peasants” can extract resources from these resource units for building specific buildings or army units. Moreover, six stationary *Building* unit categories exist. Five building types can generate specific army units (Figure 11). Conversely, the “guard tower” neither spawns army units nor can launch assaults upon adversaries. “Peasants” erect building units on any vacant map location. Lastly, seven army unit variants can

traverse and assail opponents. Except for the “peasant”, the remaining six army units and “guard tower” employ a rock-paper-scissors paradigm.

The game map comprises a discrete 32×32 grid partitioned into grass and river cells. Grass cells permit building unit construction and passage for army units, whereas exclusively “dragon” may cross river cells. The initialized map contains one “town hall,” three “peasants,” and randomly positioned river cells and resource units.

Observation Space. An agent’s observation encompasses a 32×32 map, supplemented by additional game states (e.g., observable units’ health points and resource unit quantities). Regions not traversed are masked, and unobserved enemy units are eliminated.

Action Space. We follow the variant action space designed by (Xu et al., 2022). Two agents govern half of the total units. A 0/1 action is appended to each unit, signifying whether it will act or remain *IDLE*. Specifically, agents generate a collective action and a 0/1 flag for every units. For a unit designated with a 1, it executes the collective action; for a unit labeled 0, the action *CONTINUE* is implemented. After discerning which units will react, the agent then predicts an action type (e.g., *MOVE*, *ATTACK*) and subsequently anticipate action outputs based on the said action type. We collate all accessible action types and their outputs as follows.

- **IDLE:** *None* (the elected units remain stagnant).
- **CONTINUE:** *None* (Units persist with their prior actions).
- **GATHER:** *The target resource unit’s ID.*
- **ATTACK:** *The enemy unit’s ID.*
- **TRAIN:** *The army type to be trained.*
- **BUILD:** *The building type and the designated (x, y) construction position.*
- **MOVE:** *The designated (x, y) movement position.*

Reward Function. Sparse rewards define this environment. The agent garners a reward of 1 if victorious and -1 if defeated. For all other timesteps, the environment yields a reward of 0.

E Implementation and Training Details

All code will be released soon, licensed under the MIT license (with *Overcooked*, *MiniRTS*, and *RED* licensed under their respective licenses). Concerning the PLM specifically *gpt-3.5-turbo*, we employ OpenAI’s APIs for facilitation. The PLM is exclusively utilized during the pretraining phase of the language-grounded MARL policy (limited to *Overcooked*) and the evaluation processes, thus preventing any undue financial obligations to SAMA’s implementation. We incorporate the technique of batch prompting⁴ (Cheng et al., 2023) to simultaneously minimize token and temporal expenditures when engendering pretraining subgoals and appraising SAMA’s efficacy under varied random seeds. The computing hardware comprises two servers, each with 256GB of memory, and a pair of NVIDIA GeForce RTX 3090s, outfitted with 24GB of video memory. The training time (plus PLM query time) for *Overcooked-AI* is around $43 + 3$ hours and $22 + 3$ hours for *MiniRTS*.

Required Resources at Different Procedures Table 4 shows the resources required by SAMA.

E.1 Overcooked

E.1.1 The Proposed SAMA

At each timestep t , every agent i acquires an $h \times w$ grid observation o_t^i and yields a distribution over the action $\pi(a_t^i | o_t^i, z, g_t^i)$. The action is characterized by a one-hot matrix mirroring the observation’s dimensions, signifying the agent’s targeted relocation. It is imperative to note that, besides relocation, *Overcooked* agents’ actions also enable manipulation of environmental objects, such as onions and pots. We augment the original *Overcooked*

⁴<https://github.com/HKUNLP/batch-prompting>.

Table 4: The resources needed at different procedures.

Procedures	Required resources	MARL or not
Preprocessing	Papers and codes	no MARL
Semantically-aligned task decomposition	Rollout samples	no MARL
Language-grounded MARL	Rollout samples	MARL
Self-reflection	Rollout samples	no MARL

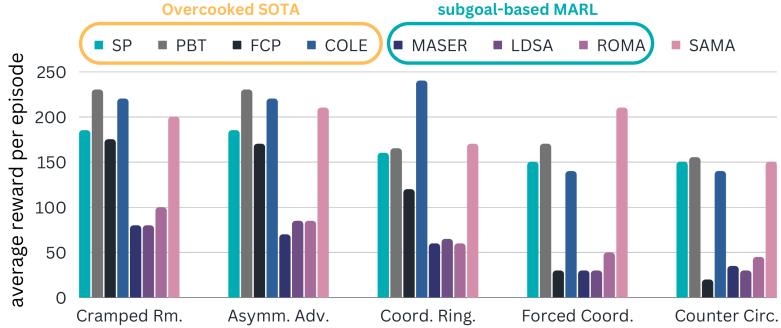


Figure 12: The mean self-play rewards of episodes over 400 timesteps under 10 random seeds. SAMA can approach or even exceed the performance of SOTA algorithm.

map to reduce the action space, thus facilitating the agent’s co-location with the object. Consequently, the action of operating is seamlessly converted into a relocation action. We shall excise the subscript t for expediency.

Possessing an encompassing linguistic knowledge, the agent initially aligns this language with environmental observations, employing a pre-existing language grounding module to engender a language-based representation $X = \text{Ground}(o, z, g) \in \mathbb{R}^{h \times w \times d}$. This conduit captures the intricate connections among the subgoal, the task manual, and the observation. Such a representation is harnessed to generate the policy. It is essential to highlight that SAMA remains congruent with any language grounding module. Specifically, we construct our framework upon EMMA⁵ (Hanjie et al., 2021), embracing their grounding modules - the multi-modal attention.

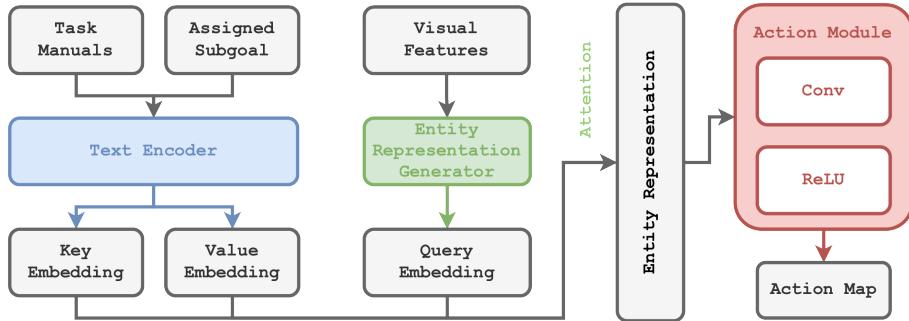


Figure 13: The 3 components of the EMMA: Text Encoder, Entity Representation Generator, and Action Module. The Text Encoder uses a pretrained BERT-base model to encode entity descriptions within a grid observation, which are then processed by the Entity Representation Generator to position entity representations in a tensor. Finally, the Action Module derives action distributions based on the processed entity representation tensor.

The architecture of EMMA is illustrated in Figure 13. The EMMA model encompasses three components: Text Encoder, Entity Representation Generator, and Action Module (Hanjie et al., 2021; Ding et al., 2023). The input for the Text Encoder constitutes a $h \times w$ grid observation imbued with entity descriptions. EMMA encodes each description utilizing a pretrained and fixed BERT-base model. Subsequently, the key and value

⁵<https://github.com/ahjwang/messenger-emma>, MIT License.

embeddings are extracted from the encoder. Within the Entity Representation Generator, EMMA embeds each entity’s symbol into a query embedding, attending to the descriptions accompanied by their respective key and value embeddings. For each entity e in the observation, EMMA situates its representation x_e within a tensor $X \in \mathbb{R}^{h \times w \times d}$, precisely aligning with the entity’s position in the observation to preserve comprehensive spatial information. The agent’s representation is merely a learned embedding of dimension d . In the Action Module, to furnish temporal information that facilitates grounding movement dynamics, EMMA concatenates the outputs of the representation generator from the most recent observations, procuring a tensor $X' \in \mathbb{R}^{h \times w \times d}$. EMMA conducts a 2D convolution on X' over the h, w dimensions, deriving a distribution over the actions with an identical shape to the grid observation. For further information regarding EMMA, the reader is directed to the original paper (Hanjie et al., 2021).

The EMMA model exhibits end-to-end differentiability and is trained using the MAPPO⁶ (Yu et al., 2022) algorithm with $\gamma = 0.99, \epsilon = 0.01$, along with the **Adam** optimizer, featuring a learning rate of $\alpha = 5 \times 10^{-5}$. Each episode is delimited to encompass 400 timesteps. The validation games are employed to preserve the model parameters with the highest reward per episode throughout the training, subsequently utilizing these parameters to evaluate the models on the test games under 10 random seeds.

E.1.2 Baselines

We train and evaluate the self-play and PBT approaches utilizing the Human-Aware Reinforcement Learning repository⁷ (Carroll et al., 2019), implementing PPO (Schulman et al., 2017) as the RL algorithm. We train FCP in accordance with the FCP publication (Strouse et al., 2021) and employ COLE’s implementation (referenced below). The COLE agent, aligned with the manuscript (Li et al., 2021c), adapts a population size of 5 and incorporates the original implementation⁸. Furthermore, the MASER⁹, LDSA¹⁰, and ROMA¹¹ agents maintain the utilization of their original implementations.

E.2 MiniRTS

Except for ROMA, all policies commence from a warm-start employing the RED pretrained model¹².

E.2.1 SAMA and RED Implementations

The training approach for SAMA and RED remains identical, with SAMA’s required natural language instructions (i.e., subgoals) adhering to the oracle script furnished by (Xu et al., 2022). The sole distinction arises during testing, wherein SAMA receives instructions via PLM prompting, whereas RED acquires instructions through Oracle scripts. Given our adaptation of MiniRTS to accommodate multi-agent algorithms, we conformed to the RED implementation and training methodology to refine the pretrained language-grounded MARL policy.

The policy architecture parallels that of MiniRTS (Hu et al., 2019), albeit with the incorporation of an ancillary value head as the critic, as proposed by (Xu et al., 2022). We refine the policy training employing a concurrent MAPPO process. Synchronous parallel workers gather transitions (s, a, r, s') , with the amassed data subdivided into four distinct batches to execute MAPPO training, utilizing a discount factor of 0.999. Generalized Advantage Estimation (GAE) advantage is applied to each data point, culminating in 100 training epochs per iteration. The Behaviour Cloning (BC) training procedure bears resemblance to MiniRTS (Hu et al., 2019), introducing an additional 0/1 action to govern the controllable unit’s behavior. These supplementary actions derive from prevalent action type commonalities.

We employ the **Adam** optimizer. A total of 15 training iterations, equivalent to 1,500 MAPPO epochs, is employed. We establish $\beta = (0.9, 0.999)$ for both phases, accompanied by a constant learning rate of 2×10^{-4} for BC training. The MARL training’s learning rate is adapted across 1,500 MAPPO epochs, commencing linearly from 0 to

⁶<https://github.com/marlbenchmark/on-policy>, MIT License.

⁷https://github.com/HumanCompatibleAI/human_aware_rl/tree/neurips2019.

⁸<https://sites.google.com/view/cole-2023>

⁹<https://github.com/Jiwonjeon9603/MASER>

¹⁰https://openreview.net/attachment?id=J5e13zmpj-Z&name=supplementary_material, Apache-2.0 License.

¹¹<https://github.com/TonghanWang/ROMA>, Apache-2.0 License.

¹²<https://drive.google.com/file/d/1dEjG9I0CwunUjVUq00dDxaYr01mg9Ll1/view?usp=sharing>, MIT License.

5×10^{-5} within the initial 300 epochs and progressively diminishing from 5×10^{-5} to 0. For further elucidation, please consult the original paper (Xu et al., 2022).

E.2.2 ROMA Implementation

We develop ROMA agents employing the original implementation¹³. To accommodate the original implementation within the state and action space of the MiniRTS, we supplanted its network structure with the policy architecture employed by SAMA and RED.

F Prompt Engineering

All the prompts are placed in the code repository¹⁴. Below, we briefly introduce the details of prompt engineering.

F.1 Task Manual Generation

Upon paragraphing the LATEX document, we employ question sets Q_{rel} and Q_{game} , as utilized in (Wu et al., 2023), to filter for relevance and extract pivotal information respectively.

We use four papers (Carroll et al., 2019; Charakorn et al., 2020; Knott et al., 2021; Li et al., 2021c) for **Overcooked**, and another two papers (Hu et al., 2019; Xu et al., 2022) for **MiniRTS**.

F.2 Goal Generation, Decomposition and Subgoal Assignment

F.2.1 Overcooked

Figure 14 and Figure 15 are illustrate the colored prompts for better understanding.

F.2.2 MiniRTS

MiniRTS furnishes an array of built-in script AIs, and we choose the medium-level AI as the opponent to facilitate expeditious prompt design for PLMs. Initially, this script dispatches all three available peasants to mine the nearest resource unit. It then randomly selects one of the seven army unit types, determining an army size n ranging between 3 and 7. A building unit corresponding to the selected army is constructed, followed by training n units of the elected army category for deployment in attacks. The script perpetuates army unit training, maintaining an army size of n .

Given that the built-in script AI constructs only two army unit types per game, we establish an oracle prompt design strategy following the ground truth of enemy units and the attack graph (Figure 11), similar with (Xu et al., 2022). In the game’s nascent stages, the PLM instructs the agent to construct suitable army units progressively according to prompts similar with **Overcooked**.

G Missing PLM Outputs

This section presents selected intermediate results of randomly chosen SAMA instances in two benchmark tasks.

Listing 1: Task Manual for **Overcooked**

```
You are a player playing a game.  
In this game, two chefs in a restaurant serve onion soup.  
This game aims to serve as many orders as possible before the end of time.  
To complete an order, the following steps need to be completed in order:  
1. Take three onions from the onion storage room and put them on the crafting table;  
2. It takes 20 seconds for the crafting table to automatically make an onion soup;
```

¹³<https://github.com/TonghanWang/ROMA>, Apache-2.0 License.

¹⁴<https://anonymous.4open.science/r/SAMA/>.

Task Manual

You are a player playing a game. In this game, two chefs in a restaurant serve onion soup. This game aims to serve as many orders as possible before the end of time. **To complete an order, the following steps need to be completed in order:** 1. Take three onions from the onion storage room and put them on the crafting table; 2. It takes 20 seconds for the crafting table to automatically make an onion soup; 3. Take a plate from the sideboard to the crafting table to serve the onion soup; 4. Take the onion soup from the crafting table, put it on the serving counter, and an order is completed now.

The above steps require two chefs to cooperate to complete. In addition to the above steps, **there are the following rules in the game:** 1. Each chef can only transport one onion at a time; 2. The crafting table starts working if and only if there are three onions on the crafting table; 3. The finished onion soup must be served on a plate taken from the sideboard before being brought to the counter.

A brief description of the game map is as follows: two chefs are in separate rooms. The room on the left contains only the onion storage room and the sideboard, while the room on the right contains two crafting tables and a serving counter. There are two crafting tables, one above the right room and one to the right of the right room. There is a shared bar between the two rooms, which can be used to pass items.

Instruction for Goal Generation & Decomposition

Valid elements: onion, onion storage room, shared bar, dining cabinet, plate, crafting table, onion soup, counter.

Your task is first to give the goal that must be completed according to the current situation by operating valid elements.

Then you need to decompose the goal into two sub-goals selected from valid sub-goals so that the two chefs can complete it separately.

One-shot Example & Chain-of-Thought Prompting

The current situation is: there are no onions on the crafting table and shared bar. To formulate goals, think step by step: To fill as many orders as possible, we must constantly make onion soup and deliver it to the serving counter. There are currently no onions on the crafting table, meaning no ready-made onion soup exists.

So the current goal by operating valid elements is: making onion soup.

Further, to decompose the goal into two sub-goals, let the two chefs complete it separately, let us think step by step:

since the current goal is to make onion soup, we need to transport three onions from the storage room to the crafting table. No onions are currently on the crafting table, so three more are needed. Since the onion storage room and the crafting table are in two separate rooms, we need one chef to carry three onions from the onion storage room to the shared bar and another chef to carry three onions from the shared bar to the crafting table. Now there are no onions on the shared bar, and each chef can only carry one onion at a time, **so the two sub-goals by operating valid elements are: 1. one chef takes an onion from the onion storage room and transports it to the shared bar; 2. one chef goes to the bar and waits.**

Instruction for Subgoal Assignment

You are a goal assigner playing a game. In this game, two restaurant players (i.e., chefs) serve onion soup. This game aims to serve as many orders as possible before the end of time.

Your task is to assign two current sub-goals to each player one-to-one according to the current status of the game and two chefs. You only can assign one of two current sub-goals to chefs.

One-shot Example & Chain-of-Thought Prompting

The two current sub-goals: 1. one chef takes an onion from the onion storage room and transports it to the shared bar; 2. one chef transports an onion from the shared bar to the crafting table.

The current status of the game: one crafting table is making onion soups, two onions are on the other crafting table, and one onion is on the shared bar.

The current status of the two chefs: Chef A is idle in the left room; Chef B is idle in the right room.

To allocate sub-goals reasonably and efficiently, let us think step by step: the current two sub-goals are one chef takes an onion from the onion storage room and transports it to the shared bar, and one chef transports an onion from the shared bar to the crafting table. We can assign two sub-goals to the two chefs since they are both idle. Note that the onion storage room is in the left room, where Chef A is, so only Chef A can take the onion. Similarly, the crafting tables are in the right room, where Chef B is located, so only Chef B can access the crafting table and transport the onion from the shared bar to the crafting table. **Therefore, the result of the two current sub-goals assignment is as follows: Chef A: one chef takes an onion from the onion storage room and transports it to the shared bar; Chef B: one chef transports an onion from the shared bar to the crafting table.**

Figure 14: The prompt designed for the **Forced Coordination** layout in Overcooked.

3. Take a plate from the sideboard to the crafting table to serve the onion soup;
4. Take the onion soup from the crafting table, put it on the serving counter, and an ↪ order is completed now.

The above steps require two chefs to cooperate to complete.

Instruction for Self-reflection

As a decision inspector, you are playing a game where two restaurant players (chefs) are serving onion soup, with the intention of fulfilling as many orders as possible before the time runs out.

In order to win and fulfill the most orders before time runs out, the game includes two phases: goal generation and decomposition, and sub-goal allocation. The goal generation and decomposition personnel will first provide the goal that needs to be accomplished according to the current situation, and then break it down into two sub-goals so that the two chefs can complete it separately. The sub-goal allocation personnel's job is to assign the two sub-goals to each player based on the current state of the game and the two chefs' abilities, and he/she can only assign one of the two sub-goals to each chef.

Valid phase: goal generation and decomposition, subgoal assignment.

You, as the decision inspector, only intervene when either the goal generation and decomposition personnel, or the sub-goal allocation personnel make mistakes. Your job consists of two parts. Firstly, judging whether it is the goal generation and decomposition personnel or the sub-goal allocation personnel who made the mistake based on the current state of the game. You can only choose from the valid phases.

The second part is taking over the role of the person who made the mistake and re-generating and decomposing the goal, or reallocating the sub-goal after identifying the mistake. **Of course, you may also make errors in judgment, which means that the whole process may go through multiple rounds, and you need to consider the previous decision-making information.**

One-shot Example and Chain-of-Thought Prompting for Step 1

The current situation is: there are no onions on the crafting table and shared bar. **The generated goal from the goal generation and decomposition personnel is:** making onion soup. **The decomposed two sub-goals from the goal generation and decomposition personnel are:** one chef takes an onion from the onion storage room and transports it to the shared bar; one chef transports an onion from the shared bar to the crafting table. **The current status of the game:** there are no onions on the crafting table and shared bar. **The current status of the two chefs:** Chef A is idle in the left room; Chef B is idle in the right room. **The result of the two current sub-goals assignment decided by the sub-goal allocation personnel is as follows:** Chef A: one chef takes an onion from the onion storage room and transports it to the shared bar; Chef B: one chef transports an onion from the shared bar to the crafting table. **The failed decision inspection history:** None.

To determine whether any errors occurred during the phase of goal generation and decomposition, or during the subgoal allocation stage, let us think step by step. Firstly, we will concentrate on the goal generation and decomposition phase. Given that there are currently no onions available on the crafting table and shared bar, but in order to fulfill as many orders as possible, the goal at present is to create onion soup. Hence, there seems to be no issue with goal generation. In the next step, we must subdivide the above goal into two subgoals. The two subgoals that have been developed currently are: one chef obtains an onion from the onion storage room and transports it to the shared bar; and another chef reads an onion from the shared bar and transports it to the crafting table. Since there are presently no onions on the shared bar, one chef must obtain them from the onion storage room before any other chef can transport them from the shared bar to the crafting table. Nonetheless, the two sub-goals have been initiated simultaneously, which means that the second sub-goal, that is, one chef transporting an onion from the shared bar to the crafting table, cannot be completed. Based on the above analysis, it appears that the problematic stage is: **goal generation and decomposition.**

One-shot Example and Chain-of-Thought Prompting for Step 2

Previous failed goals: None.

Previous failed subgoals: one chef takes an onion from the onion storage room and transports it to the shared bar; one chef transports an onion from the shared bar to the crafting table.

For more reasonable goal generation and decomposition, the goals you generate or the subgoals you decompose should no longer be consistent with previously failed goals or subgoals.

[Similar with instructions for goal generation and decomposition]

Previous failed subgoal assignments: Chef A: one chef takes an onion from the onion storage room and transports it to the shared bar; Chef B: one chef transports an onion from the shared bar to the crafting table.

For more reasonable subgoal assignment, the assignment you make should no longer be consistent with previously failed subgoal assignments.

[Similar with instructions for subgoal assignment]

Figure 15: The prompt of self-reflection designed for the **Forced Coordination** layout in Overcooked. The prompts are composed of task manual in Figure 14, instruction and one-shot CoT example.

In addition to the above steps, there are the following rules in the game:

1. Each chef can only transport one onion at a time;
2. The crafting table starts working if and only if there are three onions on the ↗ crafting table;
3. The finished onion soup must be served on a plate taken from the sideboard before ↗ being brought to the counter.

A brief description of the game map is as follows:

two chefs are in separate rooms.
The room on the left contains only the onion storage room and the sideboard, while the
→ room on the right contains two crafting tables and a serving counter.
There are two crafting tables, one above the right room and one to the right of the
→ right room.
There is a shared bar between the two rooms, which can be used to pass items.

Listing 2: Task Manual for MiniRTS

MiniRTS is a game environment designed to encompass the fundamental complexities of
→ real-time strategy games.
Consisting of two playing sides- a user-run unit governed by a pre-set policy and an
→ in-built AI enemy, players control their units to gather resources, manage
→ constructions, and defeat enemy units by destroying their base or eliminating
→ all opposition on the battlefield.
MiniRTS contains 6 unique building types, each fulfilling a specific role in-game.
→ Using allocated resources, the PEASANT unit type can construct any building type
→ in any available area on the map.
The constructed buildings can then be utilized to construct various unit types. While
→ most building types generate up to one distinct unit type, the WORKSHOP can
→ produce 3 different unit types.
For further specifics, please refer to the following table:

Building name	Description
TOWN HALL	The main building of the game, it allows a player to train PEASANTS and → serves as a storage for mined resources.
BARRACK	Produces SPEARMEN.
BLACKSMITH	Produces SWORDMEN.
STABLE	Produces CAVALRY.
WORKSHOP	Produces CATAPULT, DRAGON and ARCHER. The only building that can produce → multiple unit types.
GUARD TOWER	A building that can attack enemies, but cannot move.

A total of 7 unit types form a rock-paper-scissors attacking dynamics.
For further specifics, please refer to the following table:

Unit name	Description
PEASANT	Gathers minerals and constructs buildings, not good at fighting.
SPEARMAN	Effective against cavalry.
SWORDMAN	Effective against spearmen.
CAVALRY	Effective against swordmen.
DRAGON	Can fly over obstacles, can only be attacked by archers and towers.
ARCHER	Great counter unit against dragons.
CATAPULT	Easily demolishes buildings.

Kindly note that the maximum number of each building cannot exceed 1 and likewise the
→ maximum number of each type of unit cannot exceed 6.

Listing 3: Some Translated (Environment and Agent) States in Overcooked

1. Both crafting tables are making onion soups, and there is one plate on the shared
→ bar.
2. There are three onions on one crafting table, one onion on the shared bar, and one
→ plate on the sideboard.
3. One onion is on the shared bar, and there are no onions or plates on either of the
→ crafting tables or the serving counter.
4. Two completed onion soups are on one crafting table, and three onions are on the
→ other crafting table.
5. One chef is holding an onion, and the other chef is holding a plate, with one onion
→ on one crafting table and two onions on the other crafting table.
6. Both crafting tables have three onions each, and both chefs are holding a plate.
7. One crafted onion soup is on the shared bar, waiting for a chef to serve it on a
→ plate, and the other crafting table has two onions.
8. Two onions are on one crafting table, one onion is on the shared bar, and a
→ completed onion soup is on a plate on the serving counter.
9. One chef is holding an onion, one onion is on the shared bar, and both crafting
→ tables have two onions each.
10. A crafted onion soup is on a plate on one crafting table, while the other crafting
→ table is making onion soup with three onions on it.

Listing 4: Some Translated (Environment and Agent) States in MiniRTS

1. The enemy has 4 swordman and 3 cavalry; our side has 2 archer, 1 workshop building,
→ 1 peasant, limited resources.
2. The enemy has 2 dragon and 5 spearman; our side has 3 archer, 1 stable building, 2
→ peasant, sufficient resources.
3. The enemy has 6 spearmen and 1 dragon; our side has 4 cavalry, 1 blacksmith
→ building, 2 peasant, limited resources.
4. The enemy has 3 dragon and 4 archer; our side has 5 peasant, 1 town hall building,
→ limited resources.
5. The enemy has 2 cavalry and 3 catapult; our side has 1 guard tower, 1 stable
→ building, 3 peasant, sufficient resources.
6. The enemy has 3 archer and 4 swordman; our side has 4 spearman, 1 blacksmith
→ building, 1 peasant, limited resources.
7. The enemy has 1 dragon, 4 spearman, and 2 catapult; our side has 2 archer, 1
→ workshop building, 2 peasant, sufficient resources.
8. The enemy has 4 cavalry and 3 swordman; our side has 4 spearmen, 1 barrack building,
→ 1 peasant, limited resources.
9. The enemy has 2 archer, 3 spearman, and 1 dragon; our side has 1 guard tower, 1
→ stable building, 1 peasant, sufficient resources.
10. The enemy has 5 spearmen and 2 swordman; our side has 3 peasant, 1 town hall
→ building, limited resources.

To monitor the training progress, we evaluate the agents at a very low frequency (Figure 5 is drawn according to the evaluation results). In light of the potential emergence of unaccounted environmental states during the evaluation process, periodic inclusion of the encountered states and corresponding PLM-generated subgoals to the dataset $\mathcal{D}_{\text{subgoal}}$ is undertaken. However, determining whether a goal or subgoal generated by a PLM exists in the offline dataset is not a simple task. One approach is to obtain the embeddings of the two goals to be

compared using an open-source PLM and then assess their similarity (e.g., cosine similarity) to see if it exceeds a predefined threshold. Nevertheless, this method can be influenced by the performance of the open-source PLM and the chosen threshold. In our paper, we adopted an alternative method, using a query to GPT-4 to discern if a goal is classified as a new one. According to our experimental statistics, over 97% of the goals were covered during the offline generation phase.

Listing 5: Some Generated Goals in Overcooked

1. Create onion soup.
2. Deliver onion soup to counter.
3. Cook onion soup.
4. Complete onion soup making process.
5. Prepare onion soup.
6. Serve onion soup.
7. Make onion soup.
8. Transfer onion soup to counter.
9. Craft onion soup.
10. Serve cooked soup to customers.

Listing 6: Some Generated SubGoals in Overcooked

1. one chef takes an onion from the onion storage room and transports it to the shared
 ↪ bar;
2. one chef goes to the bar and waits.
3. Another chef picks up the onion from the shared bar and places it on the crafting
 ↪ table;
4. The second chef returns to the onion storage room to get another onion;
5. One chef takes a plate from the sideboard and transports it to the shared bar;
6. The other chef picks up the plate from the shared bar and sets it down next to the
 ↪ crafting table;
7. A chef takes the second onion from the onion storage room and brings it to the
 ↪ shared bar;
8. The other chef collects the second onion from the shared bar and places it on the
 ↪ crafting table;
9. The first chef retrieves the third onion from the onion storage room and delivers
 ↪ it to the shared bar;
10. The other chef obtains the third onion from the shared bar and places it on the
 ↪ crafting table, triggering the crafting process;
11. One chef goes to the shared bar and waits for the onion soup to finish;
12. The second chef picks up the plate from the sideboard and brings it to the shared
 ↪ bar;
13. The other chef collects the plate from the shared bar and places it near the
 ↪ crafting table;
14. One chef moves to the crafting table and waits for the onion soup to finish;
15. The first chef takes the completed onion soup from the crafting table and sets it
 ↪ on the plate;
16. A chef transports the plated onion soup from the crafting table to the serving
 ↪ counter;
17. The other chef retrieves another onion from the onion storage room and brings it
 ↪ to the shared bar;
18. One chef picks up the onion from the shared bar and places it on the second

```

    ↳ crafting table;
19. A chef takes an onion from the onion storage room and places it on the shared bar;
20. The second chef picks up the onion from the shared bar and places it on the second
    ↳ crafting table.

```

Listing 7: Some Generated Goals in MiniRTS

1. mine with all idle peasant
2. build one guard tower
3. build one blacksmith
4. build one barrack
5. build one stable
6. build one workshop
7. build 3 cavalry
8. build 2 catapult
9. build 5 archer
10. build 4 swordman

Listing 8: Reward Function Snippet for “make onion soup”

```

if num_onions < 0:
    return 0
if num_onions > Recipe.MAX_NUM_INGREDIENTS:
    return 0
if cooking_tick >= 0 and num_onions == 0:
    return 0
if finished and num_onions == 0:
    return 0
if finished:
    return 1

```

Listing 9: Reward Function Snippet for “place one onion on the crafting table”

```

if not ingredient.name in Recipe.ALL_INGREDIENTS:
    return 0
if self.is_full:
    return 0
else:
    return 1

```

Listing 10: Reward Function Snippet for “build [#number] [unit_name]”

```
build_type = build_cmd['target_type']
```

```

for my_unit in current['my_units']:
    unit_type = my_unit['unit_type']
    if unit_type != build_type:
        continue

    new_unit = True
    unit_id = my_unit['unit_id']
    for prev_unit in previous['my_units']:
        if unit_id == prev_unit['unit_id']:
            new_unit = False
    if new_unit:
        return 1

return 0

```

Listing 11: Self-Reflection Process in Overcooked-AI

The current situation is: there are no onions on the crafting table and shared bar.

The generated goal from the goal generation and decomposition personnel is: making onion soup.

The decomposed two sub-goals from the goal generation and decomposition personnel are: one chef takes an onion from the onion storage room and transports it to the shared bar; one chef transports an onion from the shared bar to the crafting table.

The current status of the game: there are no onions on the crafting table and shared bar.

The current status of the two chefs: Chef A is idle in the left room; Chef B is idle in the right room.

The result of the two current sub-goals assignment decided by the sub-goal allocation personnel is as follows: Chef A: one chef takes an onion from the onion storage room and transports it to the shared bar; Chef B: one chef transports an onion from the shared bar to the crafting table.

The failed decision inspection history: None.

To determine whether any errors occurred during the phase of goal generation and decomposition, or during the subgoal allocation stage, let us think step by step.

Firstly, we will concentrate on the goal generation and decomposition phase.

Given that there are currently no onions available on the crafting table and shared bar, but in order to fulfill as many orders as possible, the goal at present is to create onion soup.

Hence, there seems to be no issue with goal generation.

In the next step, we must subdivide the above goal into two sub-goals.

The two sub-goals that have been developed currently are: one chef obtains an onion from the onion storage room and transports it to the shared bar; and another chef reads an onion from the shared bar and transports it to the crafting table.

Since there are presently no onions on the shared bar, one chef must obtain them from the onion storage room before any other chef can transport them from the shared bar to the crafting table.

Nonetheless, the two sub-goals have been initiated simultaneously, which means that the second sub-goal, that is, one chef transporting an onion from the shared bar

→ to the crafting table, cannot be completed.

Based on the above analysis, it appears that the problematic stage is: goal generation and decomposition.

Previous failed sub-goals: one chef takes an onion from the onion storage room and
 → transports it to the shared bar; one chef transports an onion from the shared
 → bar to the crafting table.

For more reasonable goal generation and decomposition, the goals I generate or the sub-
 → goals I decompose should no longer be consistent with previously failed goals or
 → sub-goals.

The current situation is: there are no onions on the crafting table and shared bar.

To formulate goals, think step by step:

To fill as many orders as possible, we must constantly make onion soup and deliver it
 → to the serving counter.

There are currently no onions on the crafting table, meaning no ready-made onion soup
 → exists.

So the current goal by operating valid elements is: making onion soup.

Further, to decompose the goal into two sub-goals, let the two chefs complete it

→ separately, let us think step by step:

since the current goal is to make onion soup, we need to transport three onions from
 → the storage room to the crafting table.

No onions are currently on the crafting table, so three more are needed.

Since the onion storage room and the crafting table are in two separate rooms, we need
 → one chef to carry three onions from the onion storage room to the shared bar
 → and another chef to carry three onions from the shared bar to the crafting table
 → .

Now there are no onions on the shared bar, and each chef can only carry one onion at a
 → time, so the two sub-goals by operating valid elements are: 1. one chef takes
 an onion from the onion storage room and transports it to the shared bar; 2. one
 chef goes to the bar and waits.

H Missing Ablation Studies

Is the learned policy grounded? Observe that Figure 9 (Right) and Figure 10 (Right) have already furnished ample evidence supporting the command-following performance of the SAMA policy, as the attained rewards and win rates proximate to SOTA methods can solely be accomplished by adhering to the Oracle commands. Consequently, we introduce an ancillary metric, a modification of the criterion postulated by RED: intrinsically, as a command-following policy, its reward or win rate ought to diminish if it receives “inferior” commands; conversely, if the reward or win rate remains stable, the policy must inadequately adhere to the commands. Thus, we interpolate between the PLM-generated subgoal and the wholly random subgoal, subsequently evaluating disparate policies’ reward and win rates with varying proportions of random commands. The outcomes are delineated in Figure 16. The findings corroborate our preliminary analysis.

Does self-reflection prove advantageous? To ascertain whether the self-reflection stimulates PLMs to rectify logical or factual inaccuracies they commit during semantically aligned goal generation, decomposition, and allocation, we examined the efficacy of SAMA incorporating diverse quantities of self-reflection trials. Gleaning from Figure 16 (Right), we garner two notable observations: (1) The self-reflection mechanism engenders a marked enhancement in performance; (2) In most instances, exceeding a single self-reflection trial elicits a conspicuous elevation in performance.

Is self-reflection generalizable? To further test the generality of the self-reflection component, we conducted

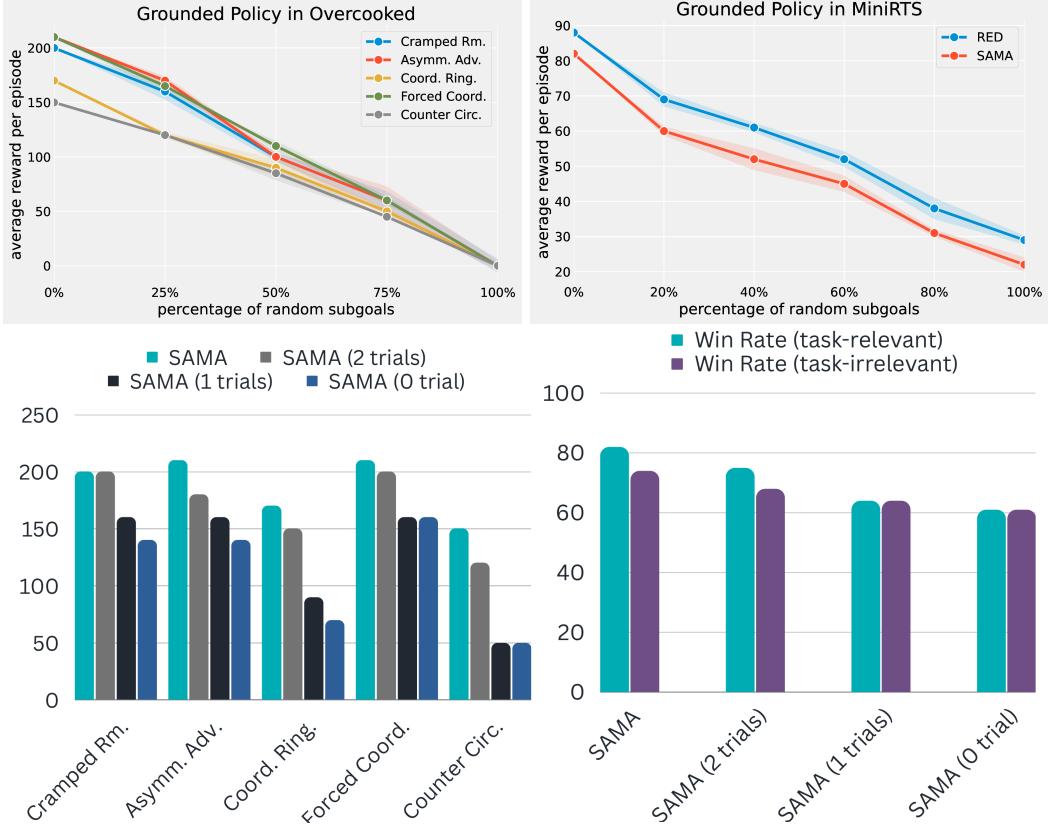


Figure 16: **Top:** Rewards per episode and win rates with an increasing amount of random commands. **Bottom:** Effect of different number of self-reflection trials. “0 trial” means no self-reflection. Generally, only more than 1 trial can bring about significant performance improvement. The in-context examples that are not task-relevant also demonstrate decent performance, which suggests that the self-reflection mechanism has a certain level of generalizability. This can help to some extent in reducing the amount of prompt engineering required.

an additional quick experiment. Specifically, inspired by this paper (Min et al., 2022), we tried applying the **Overcooked-AI**-designed self-reflection in-context example to the **MiniRTS** task. A plausible explanation for promising results lies in the evidence presented by aforementioned paper (Min et al., 2022), suggesting that in-context examples can improve PLM’s classification ability even without providing correct labels for classification tasks. One potential reason is that these examples constrain the input and output spaces, guiding the PLM’s behavior. Similarly, in our case, while the **Overcooked-AI**-designed self-reflection chain of thought cannot directly guide **MiniRTS** task decomposition, it does specify the “chain-of-thought space” from input (current state, failure flag, etc.) to output (regenerated goals, subgoals, etc.). As shown in Figure 16, even non-task-relevant in-context examples exhibit decent performance, indicating a certain level of generalizability in the self-reflection mechanism, which could help reduce the extent of prompt engineering required.

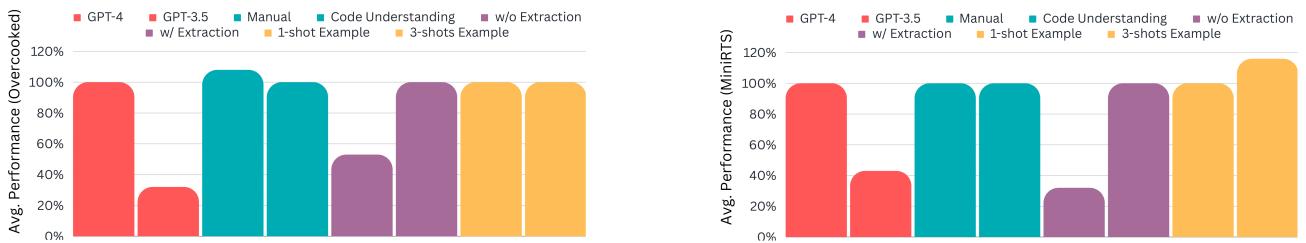


Figure 17: **Left:** the average performance of different ablations in all tasks in **Overcooked**. **Right:** the average win rate of different ablations in **MiniRTS**.

Do different versions of GPT have a big impact on performance? We tested the average performance of GPT-3.5 and GPT-4 on all tasks, as shown in the Figure 17. The vertical axis shows the percentage of different algorithms compared to SAMA. As can be seen from the figure, there is a huge gap in performance between GPT-3.5 and GPT-4 (default in SAMA). The poorer performance of GPT-3.5 is also consistent with some existing findings (Chen et al., 2023b).

Can PLM design high-quality reward functions? When pre-training language-grounded MARL agents, intrinsic rewards need to be designed to indicate whether the subgoal is completed. Since the state fed back by `Overcooked` and the `MiniRTS` contains high-level semantic information, it is easy for humans to write a judgment function for whether the corresponding subgoal is completed. For example, in `Overcooked`, you can directly read from the state whether there are onions on the cooking table, how many onions there are, etc.; in `MiniRTS`, you can also read from the state the type and quantity of the currently manufactured building or army. Then we can use a manually designed reward function to evaluate the performance of the reward function automatically generated by PLM. As can be seen from Figure 17, in the `Overcooked` environment, the reward function automatically generated by PLM is slightly weaker than the manual design; but it shows comparable performance in `MiniRTS`. A simple analysis shows that the latter subgoals are easier to understand by PLM, such as confirming the type of building, type of army and number of troops built. The former subgoal requires a certain level of understanding. For example, for the subgoal of “putting onions from the storage room to the cooking table”, the storage room is actually meaningless. The key is to detect whether there is an extra onion on the cooking table. And here it’s no longer a simple counting task, but a comparison with the number of onions currently on the cooking table.

Does the extraction of interactive objects effectively limit the goal space generated by PLM and further improve performance? We tested not performing the preprocessing step of interactive objects extraction during the goal generation, decomposition and subgoal assignment phase, but directly prompting PLM to generate goals and subgoals according to the task manual. As can be seen from the Figure 17, this brings a significant performance degradation. The fundamental reason is that unbounded goal space causes language input to contain a large amount of redundant information, which greatly affects the pre-training and performance of language-grounded policy. Poor language-grounded policy ultimately leads to reduced task completion.

Can more hand-designed few-shot examples improve performance? Few-shot, or in-context examples, are considered critical to improving PLM performance. But more examples mean more labor costs and worse generalization capabilities. We evaluate the performance of the algorithm with 1 and 3 few-shot examples in all sessions that require prompt PLM. As can be seen from Figure 17, more few-shot examples will not bring performance improvements in all tasks. Considering the balance between performance and cost, 1-shot example is used as the default setting in SAMA.

I More Environments

This section mainly wants to verify two points: first, whether SAMA can handle tasks with more agents; second, whether SAMA can handle tasks where human common sense plays little role.

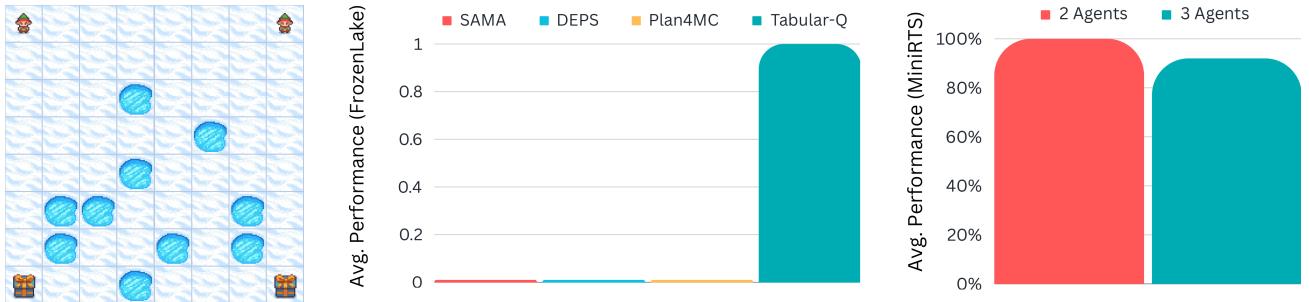


Figure 18: **Left:** The map of multi-agent version `FrozenLake` task.. **Middle:** the average success rate of different PLM-based task planner in multi-agent version `FrozenLake`. **Right:** the average win rate of SAMA in `MiniRTS` with different number of agents.

For the first point, since there are currently few environments in MARL that can better adapt to language

models and be solved by human commonsense like `Overcooked` or `MiniRTS`, we made simple modifications to the latter to enable it to accommodate more agents at the same time. Similar to modifying the environment from a single-agent to a 2-agents environment, we can also use a similar method to extend it to N agents. Here we set N to 3. Specifically, 3 agents emerge in the modified environment, each governing $1/3$ of the units. Analogously, we also modestly modified the built-in medium-level AI script, enabling the random selection of 3 types of army units in each iteration. Given that the built-in script AI constructs only 3 army unit types per game, we establish an oracle prompt design strategy following the ground truth of enemy units and the attack graph.

When the number of agents is larger, feasible plans will grow exponentially as the task horizon increases, which will be more challenging for current PLM. As shown in Figure 18 (Right), the performance of SAMA does not drop significantly in scenarios with more agents. This shows that the SAMA framework itself has certain scalability.

For the second point, we have modified the classic `FrozenLake` environment¹⁵ in OpenAI Gym and expanded it into a task for two agents, which can be regarded as a discrete version of the Spread task in MPE¹⁶. In this task, two agents need to avoid holes in the ice and each reach a target location (Figure 18 Left). The difficulty with this task is that each agent’s observation space contains only integers representing its own coordinates and those of another agent. Other than that, it contains no information and can only be remembered through continuous exploration to remember all possible situations. In this case, human commonsense cannot give the optimal planning in advance.

We verify the performance of SAMA and other PLM-based task planners (see details below) on the multi-agent version `FrozenLake`. Figure 18 (Middle) shows the average rewards of different algorithms over 20 episodes of the game. Note that `FrozenLake` is a sparse reward task, and there is a +1 reward only when the agent reaches the target point. As can be seen from the figure, the PLM-based task planner we selected, including SAMA, is unable to solve this type of problem. How we empower PLM to solve such tasks will be left to further exploration later.

There could be two potential approaches to enable language agents, such as SAMA, to handle this type of problem. The first approach is lightweight, in which the language agents can learn through trial and error, similar to how MARL agents explore their environment, gather experience, and assist with subsequent decision-making. The feasibility of this approach has been preliminarily verified in a recent work (Anonymous, 2023). The second approach is a heavier one involving knowledge editing (De Cao et al., 2021), where task-specific knowledge is injected or modified in a task-oriented manner without altering the PLM’s general capabilities. This second method is gradually becoming popular in the NLP field, with many papers published (Cao et al., 2023).

J Other Limitations

The Correctness of the Generated Task Manual and Reward Function In the current version of SAMA, we have not designed additional mechanisms to ensure the correctness of the generated task manual and the generated reward function. From the experimental results, this does not affect SAMA’s ability to approach SOTA baselines and achieve a leading sample efficiency. In Figure 17, we have also compared the performance impact of the human-designed reward function with that of the PLM-generated reward function. It can be seen from the figure that both methods show comparable performance levels. We plan to introduce a meta-prompt mechanism (Yang et al., 2024; Ma et al., 2024) to adjust the output of the PLM during the task manual and reward function generation stages based on the performance of the final algorithm, thus further improving the performance of the SAMA algorithm. However, as this mechanism may bring significant framework changes to the SAMA method, and the current version of SAMA is mainly designed for proof-of-concept purposes, we plan to explore the meta-prompt mechanism in more depth in our subsequent work.

Document-Free Preprocessing Although the vast majority of tasks are usually accompanied by academic papers, technical reports, or documentation, there may still be a small number of boundary tasks that do not meet these requirements. There are several potential technical approaches to generate task manuals and carry out state/action translations in different ways. These approaches have their respective advantages and disadvantages when compared to the method employed by SAMA.

Firstly, we can have language agents create task manuals incrementally by continually interacting with the

¹⁵https://gymnasium.farama.org/environments/toy_text/frozen_lake/.

¹⁶<https://github.com/openai/multiagent-particle-envs>.

environment, receiving feedback, and summarizing their experiences without relying on assumed resources like research papers or technical reports. The feasibility of this approach has already been preliminarily validated in some studies (Chen et al., 2023a). However, a limitation of this approach is that the generation of task manuals deeply intertwines with the task-solving process, and the language agents’ capabilities influence the quality of these manuals in solving specific tasks, such as exploration, planning, reasoning, and induction. This deep coupling may lead to negative feedback loops, causing the quality of the manuals and the solving strategies to converge prematurely to suboptimal, or even trivial, solutions.

Secondly, for state/action translation, we can use pre-trained multimodal large models to process image frames (Wang et al., 2023a) or directly compare differences between adjacent frames (Du et al., 2023b, 2024). This approach has also received preliminary validation through previous work. Nevertheless, it performs poorly on non-image tasks, making it less versatile than SAMA. Additionally, most pre-trained multimodal models use predominantly natural images for training, which limits their generalizability to tasks involving non-natural images, such as those found in gaming or simulation environments.

K Societal Impact

Though PLMs priors have demonstrated remarkable common-sense aptitudes, it is widely recognized that these models exhibit significant susceptibility to detrimental social prejudices and stereotypes (Bender et al., 2021; Abid et al., 2021; Nadeem et al., 2021). When employing such models as goal generators, decomposers, and allocators within the MARL context, as SAMA exemplifies, it is imperative to thoroughly comprehend and counteract any potential adverse manifestations engendered by these biases. While our investigation is centered upon simulated environments and tasks, we assert the necessity for more rigorous examination if such systems were to be implemented in pursuit of open-ended learning within real-world settings, such as autonomous driving, multi-agent pathfinding, cloud computing, etc. Potential ameliorative approaches specific to SAMA could encompass (Du et al., 2023): the proactive screening of PLM generations to eradicate defamatory content prior to their utilization as generated goals and subgoals, prompting the PLM with directives concerning the nature of goals or subgoals to generate and/or the exclusive employment of the closed-form SAMA variant accompanied by meticulously confined goal and subgoal spaces.

References for Supplementary Material

- Abid, A., Farooqi, M., and Zou, J. (2021). Persistent anti-muslim bias in large language models. In *AIES*.
- Alayrac, J.-B., Donahue, J., Luc, P., Miech, A., Barr, I., Hasson, Y., Lenc, K., Mensch, A., Millican, K., Reynolds, M., et al. (2022). Flamingo: a visual language model for few-shot learning. In *NeurIPS*.
- Anonymous (2023). Can language agents approach the performance of RL? an empirical study on openAI gym. In *Submitted to The Twelfth International Conference on Learning Representations*. under review.
- Anonymous (2024a). Learning planning abstractions from language. In *ICLR*.
- Anonymous (2024b). PAE: Reinforcement learning from external knowledge for efficient exploration. In *ICLR*.
- Becht, M., Gurzki, T., Klarmann, J., and Muscholl, M. (1999). Rope: Role oriented programming environment for multiagent systems. In *IFCIS*.
- Bellemare, M., Srinivasan, S., Ostrovski, G., Schaul, T., Saxton, D., and Munos, R. (2016). Unifying count-based exploration and intrinsic motivation. In *NeurIPS*.
- Bender, E. M., Gebru, T., McMillan-Major, A., and Shmitchell, S. (2021). On the dangers of stochastic parrots: Can language models be too big? In *ACM FAccT*.
- Blukis, V., Terme, Y., Niklasson, E., Knepper, R. A., and Artzi, Y. (2020). Learning to map natural language instructions to physical quadcopter control using simulated flight. In *CoRL*.
- Böhmer, W., Kurin, V., and Whiteson, S. (2020). Deep coordination graphs. In *ICML*. PMLR.

- Branavan, S., Silver, D., and Barzilay, R. (2012). Learning to win by reading manuals in a monte-carlo framework. *Journal of Artificial Intelligence Research*, 43:661–704.
- Brohan, A., Chebotar, Y., Finn, C., Hausman, K., Herzog, A., Ho, D., Ibarz, J., Irpan, A., Jang, E., Julian, R., et al. (2023). Do as i can, not as i say: Grounding language in robotic affordances. In *CoRL*.
- 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. In *NeurIPS*.
- Burda, Y., Edwards, H., Storkey, A., and Klimov, O. (2019). Exploration by random network distillation. In *ICLR*.
- Cao, B., Lin, H., Han, X., and Sun, L. (2023). The life cycle of knowledge in big language models: A survey. *arXiv preprint arXiv:2303.07616*.
- 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. In *NeurIPS*.
- Carta, T., Romac, C., Wolf, T., Lamprier, S., Sigaud, O., and Oudeyer, P.-Y. (2023). Grounding large language models in interactive environments with online reinforcement learning. In *ICML*.
- Chan, C.-M., Chen, W., Su, Y., Yu, J., Xue, W., Zhang, S., Fu, J., and Liu, Z. (2024). Chateval: Towards better llm-based evaluators through multi-agent debate. In *ICLR*.
- Charakorn, R., Manoonpong, P., and Dilokthanakul, N. (2020). Investigating partner diversification methods in cooperative multi-agent deep reinforcement learning. In *ICONIP*.
- Chase, H. (2022). Langchain. <https://github.com/hwchase17/langchain>. Released on 2022-10-17.
- Chen, L., Wang, L., Dong, H., Du, Y., Yan, J., Yang, F., Li, S., Zhao, P., Qin, S., Rajmohan, S., et al. (2023a). Introspective tips: Large language model for in-context decision making. *arXiv preprint arXiv:2305.11598*.
- Chen, L., Zaharia, M., and Zou, J. (2023b). How is chatgpt’s behavior changing over time? *arXiv preprint arXiv:2307.09009*.
- Chen, W., Su, Y., Zuo, J., Yang, C., Yuan, C., Qian, C., Chan, C.-M., Qin, Y., Lu, Y., Xie, R., et al. (2024). Agentverse: Facilitating multi-agent collaboration and exploring emergent behaviors in agents. In *ICLR*.
- Cheng, Z., Kasai, J., and Yu, T. (2023). Batch prompting: Efficient inference with large language model apis. In *EMNLP Industry Track*.
- Choi, J., Sharma, A., Lee, H., Levine, S., and Gu, S. S. (2021). Variational empowerment as representation learning for goal-conditioned reinforcement learning. In *ICML*.
- Co-Reyes, J. D., Gupta, A., Sanjeev, S., Altieri, N., DeNero, J., Abbeel, P., and Levine, S. (2019). Meta-learning language-guided policy learning. In *ICLR*.
- Colas, C., Karch, T., Sigaud, O., and Oudeyer, P.-Y. (2022). Autotelic agents with intrinsically motivated goal-conditioned reinforcement learning: a short survey. *Journal of Artificial Intelligence Research*, 74:1159–1199.
- Dalal, M., Chiruvolu, T., Chaplot, D. S., and Salakhutdinov, R. (2024). Plan-seq-learn: Language model guided RL for solving long horizon robotics tasks. In *ICLR*.
- Dasgupta, I., Kaeser-Chen, C., Marino, K., Ahuja, A., Babayan, S., Hill, F., and Fergus, R. (2023). Collaborating with language models for embodied reasoning. *arXiv preprint arXiv:2302.00763*.
- De Cao, N., Aziz, W., and Titov, I. (2021). Editing factual knowledge in language models. In *EMNLP*.
- Devlin, S., Yliniemi, L., Kudenko, D., and Tumer, K. (2014). Potential-based difference rewards for multiagent reinforcement learning. In *AAMAS*.
- 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 *ICML*.

- Driess, D., Xia, F., Sajjadi, M. S., Lynch, C., Chowdhery, A., Ichter, B., Wahid, A., Tompson, J., Vuong, Q., Yu, T., et al. (2023). Palm-e: An embodied multimodal language model. In *ICML*.
- Du, Y., Watkins, O., Wang, Z., Colas, C., Darrell, T., Abbeel, P., Gupta, A., and Andreas, J. (2023a). Guiding pretraining in reinforcement learning with large language models. In *ICML*.
- Du, Y., Yang, M., Florence, P., Xia, F., Wahid, A., Ichter, B., Sermanet, P., Yu, T., Abbeel, P., Tenenbaum, J. B., et al. (2024). Video language planning. In *ICLR*.
- Du, Y., Yang, S., Dai, B., Dai, H., Nachum, O., Tenenbaum, J. B., Schuurmans, D., and Abbeel, P. (2023b). Learning universal policies via text-guided video generation. In *NeurIPS*.
- Ecoffet, A., Huizinga, J., Lehman, J., Stanley, K. O., and Clune, J. (2021). First return, then explore. *Nature*, 590(7847):580–586.
- 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 *NeurIPS Datasets and Benchmarks Track*.
- Foerster, J., Farquhar, G., Afouras, T., Nardelli, N., and Whiteson, S. (2018). Counterfactual multi-agent policy gradients. In *AAAI*.
- Grote, E.-M., Pfeifer, S. A., Röltgen, D., Kühn, A., and Dumitrescu, R. (2020). Towards defining role models in advanced systems engineering. In *ISSE*.
- Guo, Z., Thakoor, S., Pislari, M., Avila Pires, B., Altché, F., Tallec, C., Saade, A., Calandriello, D., Grill, J.-B., Tang, Y., et al. (2022). Byol-explore: Exploration by bootstrapped prediction. In *NeurIPS*.
- Gupta, T., Karkus, P., Che, T., Xu, D., and Pavone, M. (2022). Foundation models for semantic novelty in reinforcement learning. In *NeurIPS Foundation Models for Decision Making Workshop*.
- Gupta, T., Mahajan, A., Peng, B., Böhmer, W., and Whiteson, S. (2021). Uneven: Universal value exploration for multi-agent reinforcement learning. In *ICML*.
- Gur, I., Furuta, H., Huang, A., Safdari, M., Matsuo, Y., Eck, D., and Faust, A. (2024). A real-world webagent with planning, long context understanding, and program synthesis. In *ICLR*.
- Hanjie, A. W., Zhong, V. Y., and Narasimhan, K. (2021). Grounding language to entities and dynamics for generalization in reinforcement learning. In *ICML*.
- Hill, F., Lampinen, A., Schneider, R., Clark, S., Botvinick, M., McClelland, J. L., and Santoro, A. (2020a). Environmental drivers of systematicity and generalization in a situated agent. In *ICLR*.
- Hill, F., Mokra, S., Wong, N., and Harley, T. (2020b). Human instruction-following with deep reinforcement learning via transfer-learning from text. *arXiv preprint arXiv:2005.09382*.
- Hill, F., Tielemans, O., von Glehn, T., Wong, N., Merzic, H., and Clark, S. (2021). Grounded language learning fast and slow. In *ICLR*.
- Hong, S., Zheng, X., Chen, J., Cheng, Y., Wang, J., Zhang, C., Wang, Z., Yau, S. K. S., Lin, Z., Zhou, L., et al. (2024). Metagpt: Meta programming for multi-agent collaborative framework. In *ICLR*.
- Hu, H., Yarats, D., Gong, Q., Tian, Y., and Lewis, M. (2019). Hierarchical decision making by generating and following natural language instructions. In *NeurIPS*.
- Huang, W., Abbeel, P., Pathak, D., and Mordatch, I. (2022a). Language models as zero-shot planners: Extracting actionable knowledge for embodied agents. In *ICML*.
- Huang, W., Xia, F., Xiao, T., Chan, H., Liang, J., Florence, P., Zeng, A., Tompson, J., Mordatch, I., Chebotar, Y., Sermanet, P., Jackson, T., Brown, N., Luu, L., Levine, S., Hausman, K., and brian ichter (2022b). Inner monologue: Embodied reasoning through planning with language models. In *CoRL*.

- Janner, M., Narasimhan, K., and Barzilay, R. (2018). Representation learning for grounded spatial reasoning. In *ACL*.
- Jeon, J., Kim, W., Jung, W., and Sung, Y. (2022). Maser: Multi-agent reinforcement learning with subgoals generated from experience replay buffer. In *ICML*.
- Jiang, Y., Gupta, A., Zhang, Z., Wang, G., Dou, Y., Chen, Y., Fei-Fei, L., Anandkumar, A., Zhu, Y., and Fan, L. (2022). Vima: General robot manipulation with multimodal prompts. In *NeurIPS 2022 Foundation Models for Decision Making Workshop*.
- Kang, Y., Wang, T., Yang, Q., Wu, X., and Zhang, C. (2022). Non-linear coordination graphs. In *NeurIPS*.
- Kenton, J. D. M.-W. C. and Toutanova, L. K. (2019). Bert: Pre-training of deep bidirectional transformers for language understanding. In *NAACL*.
- Knott, P., Carroll, M., Devlin, S., Ciosek, K., Hofmann, K., Dragan, A., and Shah, R. (2021). Evaluating the robustness of collaborative agents. In *AAMAS*.
- Kulkarni, T. D., Narasimhan, K., Saeedi, A., and Tenenbaum, J. (2016). Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation. In *NeurIPS*.
- Küttler, H., Nardelli, N., Miller, A., Raileanu, R., Selvatici, M., Grefenstette, E., and Rocktäschel, T. (2020). The nethack learning environment. In *NeurIPS*.
- Kwon, M., Xie, S. M., Bullard, K., and Sadigh, D. (2023). Reward design with language models. In *ICLR*.
- Lhaksmana, K. M., Murakami, Y., and Ishida, T. (2018). Role-based modeling for designing agent behavior in self-organizing multi-agent systems. *International Journal of Software Engineering and Knowledge Engineering*, 28(01):79–96.
- Li, C., Wang, T., Wu, C., Zhao, Q., Yang, J., and Zhang, C. (2021a). Celebrating diversity in shared multi-agent reinforcement learning. In *NeurIPS*.
- Li, S., Puig, X., Paxton, C., Du, Y., Wang, C., Fan, L., Chen, T., Huang, D.-A., Akyürek, E., Anandkumar, A., et al. (2022). Pre-trained language models for interactive decision-making. In *NeurIPS*.
- Li, W., Wang, X., Jin, B., Lu, J., and Zha, H. (2023). Learning roles with emergent social value orientations. *arXiv preprint arXiv:2301.13812*.
- Li, W., Wang, X., Jin, B., Sheng, J., Hua, Y., and Zha, H. (2021b). Structured diversification emergence via reinforced organization control and hierarchical consensus learning. In *AAMAS*.
- Li, Y., Zhang, S., Sun, J., Du, Y., Wen, Y., Wang, X., and Pan, W. (2021c). Cooperative open-ended learning framework for zero-shot coordination. In *ICML*.
- Lin, L., Bai, Y., and Mei, S. (2024). Transformers as decision makers: Provable in-context reinforcement learning via supervised pretraining. In *ICLR*.
- Ma, Y. J., Liang, W., Wang, G., Huang, D.-A., Bastani, O., Jayaraman, D., Zhu, Y., Fan, L., and Anandkumar, A. (2024). Eureka: Human-level reward design via coding large language models. In *ICLR*.
- Mahajan, A., Rashid, T., Samvelyan, M., and Whiteson, S. (2019). Maven: Multi-agent variational exploration. In *NeurIPS*.
- Min, H., Jung, J., Kim, S., Kim, B., and Heo, J. (2018). Role-based automatic programming framework for interworking a drone and wireless sensor networks. In *SAC*.
- Min, S., Lyu, X., Holtzman, A., Artetxe, M., Lewis, M., Hajishirzi, H., and Zettlemoyer, L. (2022). Rethinking the role of demonstrations: What makes in-context learning work? In *EMNLP*.
- Nadeem, M., Bethke, A., and Reddy, S. (2021). Stereoset: Measuring stereotypical bias in pretrained language models. In *ACL*.

- Narasimhan, K., Barzilay, R., and Jaakkola, T. (2018). Grounding language for transfer in deep reinforcement learning. *Journal of Artificial Intelligence Research*, 63:849–874.
- Nguyen, D., Nguyen, P., Venkatesh, S., and Tran, T. (2022). Learning to transfer role assignment across team sizes. In *AAMAS*.
- Nguyen, D. T., Kumar, A., and Lau, H. C. (2018). Credit assignment for collective multiagent rl with global rewards. In *NeurIPS*.
- Pathak, D., Agrawal, P., Efros, A. A., and Darrell, T. (2017). Curiosity-driven exploration by self-supervised prediction. In *ICML*.
- Phan, T., Ritz, F., Belzner, L., Altmann, P., Gabor, T., and Linnhoff-Popien, C. (2021). Vast: Value function factorization with variable agent sub-teams. In *NeurIPS*.
- Ramesh, A., Pavlov, M., Goh, G., Gray, S., Voss, C., Radford, A., Chen, M., and Sutskever, I. (2021). Zero-shot text-to-image generation. In *ICML*.
- Rashid, T., Farquhar, G., Peng, B., and Whiteson, S. (2020a). Weighted qmix: Expanding monotonic value function factorisation for deep multi-agent reinforcement learning. In *NeurIPS*.
- Rashid, T., Samvelyan, M., De Witt, C. S., Farquhar, G., Foerster, J., and Whiteson, S. (2020b). Monotonic value function factorisation for deep multi-agent reinforcement learning. *The Journal of Machine Learning Research*, 21(1):7234–7284.
- Reid, M., Yamada, Y., and Gu, S. S. (2022). Can wikipedia help offline reinforcement learning? *arXiv preprint arXiv:2201.12122*.
- Rocamonde, J., Montesinos, V., Nava, E., Perez, E., and Lindner, D. (2024). Vision-language models are zero-shot reward models for reinforcement learning. In *ICLR*.
- Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. (2017). Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*.
- Shao, J., Lou, Z., Zhang, H., Jiang, Y., He, S., and Ji, X. (2022). Self-organized group for cooperative multi-agent reinforcement learning. In *NeurIPS*.
- Shi, R., Liu, Y., Ze, Y., Du, S. S., and Xu, H. (2024). Unleashing the power of pre-trained language models for offline reinforcement learning. In *ICLR*.
- Son, K., Kim, D., Kang, W. J., Hostallero, D. E., and Yi, Y. (2019). Qtran: Learning to factorize with transformation for cooperative multi-agent reinforcement learning. In *ICML*.
- Strouse, D., McKee, K., Botvinick, M., Hughes, E., and Everett, R. (2021). Collaborating with humans without human data. In *NeurIPS*.
- Sunehag, P., Lever, G., Gruslys, A., Czarnecki, W. M., Zambaldi, V., Jaderberg, M., Lanctot, M., Sonnerat, N., Leibo, J. Z., Tuyls, K., et al. (2018). Value-decomposition networks for cooperative multi-agent learning based on team reward. In *AAMAS*.
- Tam, A., Rabinowitz, N., Lampinen, A., Roy, N. A., Chan, S., Strouse, D., Wang, J., Banino, A., and Hill, F. (2022). Semantic exploration from language abstractions and pretrained representations. In *NeurIPS*.
- Tan, W., Zhang, W., Liu, S., Zheng, L., Wang, X., and An, B. (2024). True knowledge comes from practice: Aligning large language models with embodied environments via reinforcement learning. In *ICLR*.
- Tang, H., Hao, J., Lv, T., Chen, Y., Zhang, Z., Jia, H., Ren, C., Zheng, Y., Meng, Z., Fan, C., et al. (2018). Hierarchical deep multiagent reinforcement learning with temporal abstraction. *arXiv preprint arXiv:1809.09332*.
- Tellex, S., Gopalan, N., Kress-Gazit, H., and Matuszek, C. (2020). Robots that use language. *Annual Review of Control, Robotics, and Autonomous Systems*, 3:25–55.

- Wang, J., Ren, Z., Han, B., Ye, J., and Zhang, C. (2021a). Towards understanding cooperative multi-agent q-learning with value factorization. In *NeurIPS*.
- Wang, T., Dong, H., Lesser, V., and Zhang, C. (2020). Roma: multi-agent reinforcement learning with emergent roles. In *ICML*.
- Wang, T., Gupta, T., Mahajan, A., Peng, B., Whiteson, S., and Zhang, C. (2021b). RODE: Learning roles to decompose multi-agent tasks. In *ICLR*.
- Wang, X., Huang, Q., Celikyilmaz, A., Gao, J., Shen, D., Wang, Y.-F., Wang, W. Y., and Zhang, L. (2019). Reinforced cross-modal matching and self-supervised imitation learning for vision-language navigation. In *CVPR*.
- Wang, Z., Cai, S., Liu, A., Jin, Y., Hou, J., Zhang, B., Lin, H., He, Z., Zheng, Z., Yang, Y., et al. (2023a). Jarvis-1: Open-world multi-task agents with memory-augmented multimodal language models. *arXiv preprint arXiv:2311.05997*.
- Wang, Z., Cai, S., Liu, A., Ma, X., and Liang, Y. (2023b). Describe, explain, plan and select: Interactive planning with large language models enables open-world multi-task agents. In *NeurIPS*.
- Xiao, Y., Hoffman, J., and Amato, C. (2020). Macro-action-based deep multi-agent reinforcement learning. In *CoRL*.
- Xiao, Y., Tan, W., and Amato, C. (2022). Asynchronous actor-critic for multi-agent reinforcement learning. In *NeurIPS*.
- Xie, T., Zhao, S., Wu, C. H., Liu, Y., Luo, Q., Zhong, V., Yang, Y., and Yu, T. (2024). Text2reward: Dense reward generation with language models for reinforcement learning. In *ICLR*.
- Xu, S., Wang, H., and Wu, Y. (2022). Grounded reinforcement learning: Learning to win the game under human commands. In *NeurIPS*.
- Yang, C., Wang, X., Lu, Y., Liu, H., Le, Q. V., Zhou, D., and Chen, X. (2024). Large language models as optimizers. In *ICLR*.
- Yang, J., Borovikov, I., and Zha, H. (2020a). Hierarchical cooperative multi-agent reinforcement learning with skill discovery. In *AAMAS*.
- Yang, M., Zhao, J., Hu, X., Zhou, W., Zhu, J., and Li, H. (2022a). Ldsa: Learning dynamic subtask assignment in cooperative multi-agent reinforcement learning. In *NeurIPS*.
- Yang, Q., Dong, W., Ren, Z., Wang, J., Wang, T., and Zhang, C. (2022b). Self-organized polynomial-time coordination graphs. In *ICML*.
- Yang, Y., Wen, Y., Wang, J., Chen, L., Shao, K., Mguni, D., and Zhang, W. (2020b). Multi-agent determinantal q-learning. In *ICML*.
- 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. In *NeurIPS*.
- Yuan, H., Zhang, C., Wang, H., Xie, F., Cai, P., Dong, H., and Lu, Z. (2023). Plan4mc: Skill reinforcement learning and planning for open-world minecraft tasks. *arXiv preprint arXiv:2303.16563*.
- Yuan, L., Chen, D., Chen, Y.-L., Codella, N., Dai, X., Gao, J., Hu, H., Huang, X., Li, B., Li, C., et al. (2021). Florence: A new foundation model for computer vision. *arXiv preprint arXiv:2111.11432*.
- Zhang, C., Yang, K., Hu, S., Wang, Z., Li, G., Sun, Y., Zhang, C., Zhang, Z., Liu, A., Zhu, S.-C., et al. (2024a). Proagent: Building proactive cooperative ai with large language models. In *AAAI*.
- Zhang, H., Du, W., Shan, J., Zhou, Q., Du, Y., Tenenbaum, J. B., Shu, T., and Gan, C. (2024b). Building cooperative embodied agents modularly with large language models. In *ICLR*.

- Zhang, T., Li, Y., Wang, C., Xie, G., and Lu, Z. (2021). Fop: Factorizing optimal joint policy of maximum-entropy multi-agent reinforcement learning. In *ICML*.
- Zheng, L., Chen, J., Wang, J., He, J., Hu, Y., Chen, Y., Fan, C., Gao, Y., and Zhang, C. (2021). Episodic multi-agent reinforcement learning with curiosity-driven exploration. In *NeurIPS*.
- Zheng, S., Liu, J., Feng, Y., and Lu, Z. (2024). Steve-eye: Equipping llm-based embodied agents with visual perception in open worlds. In *ICLR*.
- Zhong, V., Hanjie, A. W., Wang, S., Narasimhan, K., and Zettlemoyer, L. (2021). Silg: The multi-domain symbolic interactive language grounding benchmark. In *NeurIPS*.
- Zhong, V., Rocktäschel, T., and Grefenstette, E. (2020). Rtfm: Generalising to new environment dynamics via reading. In *ICLR*.
- Zhu, X., Chen, Y., Tian, H., Tao, C., Su, W., Yang, C., Huang, G., Li, B., Lu, L., Wang, X., et al. (2023). Ghost in the minecraft: Generally capable agents for open-world environments via large language models with text-based knowledge and memory. *arXiv preprint arXiv:2305.17144*.