

UCLA

UCLA Electronic Theses and Dissertations

Title

Multi-Agent Systems: Enhancing Scalability, Task-Agent Adaptiveness and Benchmarking

Permalink

<https://escholarship.org/uc/item/0r9772h3>

Author

Long, Qian

Publication Date

2024

Peer reviewed|Thesis/dissertation

UNIVERSITY OF CALIFORNIA
Los Angeles

Multi-Agent Systems: Enhancing Scalability, Task-Agent Adaptiveness and Benchmarking

A dissertation submitted in partial satisfaction
of the requirements for the degree
Doctor of Philosophy in Computer Science

by

Qian Long

2024

© Copyright by

Qian Long

2024

ABSTRACT OF THE DISSERTATION

Multi-Agent Systems: Enhancing Scalability, Task-Agent Adaptiveness and Benchmarking

by

Qian Long

Doctor of Philosophy in Computer Science

University of California, Los Angeles, 2024

Professor Demetri Terzopoulos, Co-chair

Professor Song-Chun Zhu, Co-chair

Applying deep reinforcement learning to multi-agent environments has become a popular trend. However, creating adaptive agents that perform well in dynamic, complex settings remains challenging. Key difficulties include (1) scaling with the number of agents, as complexity grows exponentially with each additional agent, (2) adapting to new environments, where agents need to leverage past experiences, (3) cooperating with unfamiliar agents, since agents trained in fixed groups must interact effectively with unseen peers, and (4) operating within multi-modal input scenarios, beyond simple vector inputs. We identify the limitations of current approaches in addressing these challenges and propose novel methods to overcome them. The contributions of this thesis include the following:

1. *Evolutionary Population Curriculum (EPC)*: A training approach that enables agents to gradually adapt from small to large groups through a mix-and-match strategy, enhancing scalability.
2. *Social Gradient Fields (SocialGFs)*: A novel gradient-based state representation for multi-agent reinforcement learning, leveraging denoising score matching to learn social dynamics from offline samples. This adaptive representation allows agents to exhibit diverse behaviors.
3. *Inverse Attention Network*: A mechanism that models agents' Theory of Mind (ToM)

by inferring attentional states based on observations and prior actions, refining attention weights to improve decision-making.

4. *Multi-Modal Multi-Agent Benchmark in Minecraft (TeamCraft)*: A comprehensive benchmark designed to highlight the limitations of current Vision-Language Models (VLMs) in handling complex, dynamic multi-agent environments, setting a new standard for future research in multi-agent systems.

These contributions advance the field by providing scalable and adaptive solutions to fundamental challenges in multi-agent systems.

The dissertation of Qian Long is approved.

Quanquan Gu

Adnan Darwiche

Ying Nian Wu

Song-Chun Zhu, Committee Co-chair

Demetri Terzopoulos, Committee Co-chair

University of California, Los Angeles

2024

TABLE OF CONTENTS

1	Introduction	1
1.1	Scalability in Multi-Agent Systems	3
1.2	Task Adaptiveness in Multi-Agent Systems	5
1.3	Agent Adaptiveness in Multi-Agent Systems	8
1.4	A Benchmark for Multi-Modal Multi-Agent Systems	10
2	Related Work	13
2.1	Multi-Agent Reinforcement Learning	13
2.2	Attention-Based Policy Architecture	14
2.3	Curriculum Learning	14
2.4	Evolutionary Learning	15
2.5	Learning Adaptive Multi-Agent Systems	15
2.6	Social Forces in Multi-Agent Systems	16
2.7	Gradient Fields for Decision Making	16
2.8	Theory of Mind and Attention	17
2.9	Ad-Hoc Teaming	18
2.10	Platforms for Multi-Agent Systems	19
2.11	Embodied Language-Guided Benchmarks	20
2.12	Benchmarks Based on Minecraft	21
3	Evolutionary Population Curriculum for Scaling MARL	22
3.1	Evolutionary Population Curriculum	22
3.1.1	Population-Invariant Architecture	22
3.1.2	Population Curriculum	24

3.1.3	Evolutionary Selection	25
3.2	Experiments	28
3.2.1	Environments	28
3.2.2	Methods and Metric	29
3.2.3	Qualitative Results	30
3.2.4	Quantitative Results	31
4	Learning Social Gradient Fields for Adaptive Multi-Agent Systems	36
4.1	Preliminary	36
4.1.1	Theory of Social Force	36
4.1.2	Learning Gradient Fields via Score-Matching	36
4.2	Method	38
4.2.1	Problem Formulation	38
4.2.2	Offline Examples Collection	39
4.2.3	Learning Gradient Fields	40
4.2.4	Gradient-Enhanced Rewards	41
4.2.5	Learning Adaptive Policy	42
4.3	Experiments	42
4.3.1	Environments	42
4.3.2	Baselines and Evaluation Method	43
4.3.3	Quantitative Results in <i>Grassland</i>	45
4.3.4	Quantitative Results in Cooperative Navigation	45
4.3.5	Qualitative Results	47
5	Inverse Attention Agents for Multi-Agent Systems	49
5.1	Method	49

5.1.1	Self-Attention Structure	49
5.1.2	Attention Inference	51
5.1.3	Inverse Attention Agent	52
5.2	Experiments	53
5.2.1	Environment	54
5.2.2	Baseline and Evaluation Method	55
5.2.3	Quantitative Results	56
5.2.4	Impact of Different Population Scales	57
5.2.5	Human Experiments	57
5.2.6	Impact of Multiple Inverse-Att Agents	59
5.2.7	Inverse Attention Network Prediction Accuracy	59
6	A Benchmark for Multi-Modal Multi-Agent Systems in Minecraft	61
6.1	The TeamCraft Benchmark	61
6.1.1	Problem Formulation	61
6.1.2	Simulation Environment	61
6.1.3	Task Design	62
6.1.4	Centralized and Decentralized Agents	64
6.1.5	Diversity	65
6.1.6	Tasks and Expert Demonstrations Generation	67
6.1.7	Test Set and Generalization Set	68
6.2	Experiments	69
6.2.1	Baselines and Ablations	69
6.2.2	Evaluation Metrics	70
6.2.3	Evaluation Results	71

6.2.4	Qualitative Analysis	74
7	Conclusions	76
7.1	Limitations and Future Work	77
A	Evolutionary Population Curriculum Details	79
A.1	Environment Details	79
A.2	Training Details	80
A.3	Evaluation Details	80
B	SocialGF Details	82
B.1	Algorithm Details	82
B.2	Environment Details	82
B.3	Training Details	83
B.4	Network Structure	84
B.5	Evaluation Results Details	85
C	Inverse-Att Agent Details	87
C.1	Environmental Details	87
C.2	Training Details	88
C.3	Gradient Field Synthetic Data Generation	89
C.4	Gradient Field Representation of the Environment	89
D	TeamCraft Details	95
D.1	High Level Skills	95
D.2	Low Level Atomic Actions	96
D.3	Visual Diversity	96
D.4	Planner for Expert Demonstration	106

D.5 Grid-World Settings	114
D.6 TeamCraft-VLA Implementation Details	117
D.7 Additional Results of TeamCraft-VLA	119
References	122

LIST OF FIGURES

1.1	Population-invariant Q function	4
1.2	Learning gradient fields from examples for multi-agent systems	7
1.3	Inverse attention agent training pipeline	9
1.4	The main structure of the <i>TeamCraft</i> platform	12
3.1	Environment Visualizations	28
3.2	Example matches between EPC and MADDPG trained agents in <i>Grassland</i> . .	31
3.3	Quantitative result of adversarial battle	31
3.4	Quantitative result of food collection	31
3.5	Quantitive result in grassland game	32
3.6	Quantitative results in adversarial battle game and food collection game	33
3.7	Ablation analysis on the second curriculum stage in all the games	34
3.8	Environment generalization result	35
4.2	Four games in experiment	44
4.3	Quantitative results in grassland	46
4.4	Qualitative results	48
5.1	Inverse attention agent network architecture	50
5.2	Environment visualization of the spread, adversary and grassland game	54
5.3	Prediction accuracy of the inverse attention network	60
6.1	Multi-modal prompts examples	62
6.2	Task configurations example	63
6.3	The architecture of the TeamCraft-VLA model	66

6.4	Quantitative results across centralized, decentralized and grid-world settings . . .	68
6.5	Task success rates of centralized and decentralized VLA model and GPT-4o . . .	71
B.1	SocialGFs network structure	84
B.2	Grass eating ability of different methods in the grassland game	85
C.1	Qualitative results in pread, adversary and grassland games	90
D.1	Visualization of the shared visual diversity in the tasks	98
D.2	Visualization of the visual diversity in Clearing tasks	98
D.3	Visualization of the visual diversity in Building tasks	99
D.4	Visualization of crop appearances across various growing stages in Farming tasks	100
D.5	Visualization of the visual diversity in Smelting tasks	101
D.6	An example scene in the Seaside Village biome	102
D.7	An example scene in the Grass Village biome	102
D.8	An example scene in the Dessert Village biome	103
D.9	An example scene in the Half Mountain biome	103
D.10	An example scene in the Swamp biome	104
D.11	An example scene in the Iceberg biome	104
D.12	An example scene in the Snow Mountain biome	105
D.13	An example demonstration in the Building task, Part I	108
D.14	An example demonstration in the Building task, Part II	109
D.15	An example demonstration in the Clearing task, Part I	109
D.16	An example demonstration in the Clearing task, Part II	110
D.17	An example demonstration in the Farming task, Part I	110
D.18	An example demonstration in the Farming task, Part II	111
D.19	An example demonstration in the Smelting task, Part I	112

D.20 An example demonstration in the Smelting task, Part II	113
D.21 Prompt example for Building task under the grid-world setting	115
D.22 Prompt example for Clearing task under the grid-world setting	115
D.23 Prompt example for Farming task under the grid-world setting	116
D.24 Prompt example for Smelting task under the grid-world setting	116
D.25 Combining three orthogonal view images into a single composite image	117

LIST OF TABLES

2.1	Comparison with other benchmarks	20
4.1	Cross-validation results in the Grassland game with 4 sheep and 4 wolves	46
4.2	Success rate of three cooperative navigation games with different populations	46
5.1	Full result	56
5.2	Spread result	57
5.3	Adversary result	57
5.4	Grassland result	58
5.5	Results of the human experiments	58
5.6	Group reward in spread game with multi Inverse-Att agents	58
6.1	Task variants and dataset statistics	65
6.2	Comparison of TeamCraft-VLA model redundancy rates	73
B.1	The hyperparameters for learning GF	83
B.2	The hyperparameters for MAPPO	84
B.3	The occupation rate in cooperative navigation games	86
C.1	Hyperparameters for the gradient field	88
C.2	Hyperparameters for agent training	88
C.3	Hyperparameters for the inverse network	88
D.1	Comparison of TeamCraft-VLA redundancy rates	114
D.2	Comparison of TeamCraft-VLA action sequence length	114
D.3	Action space within <i>TeamCraft</i>	115
D.4	Hyperparameters for TeamCraft-VLA	117

D.5	Quantitative results of TeamCraft-VLA-7B-Cen and TeamCraft-VLA-7B-Dec . .	119
D.6	Quantitative results of TeamCraft-VLA-13B-Cen and TeamCraft-VLA-13B-Dec	120
D.7	Task success rates and subgoal success rates of various centralized models	120

ACKNOWLEDGMENTS

I am deeply grateful to Professor Demetri Terzopoulos, Professor Song-Chun Zhu, and Professor Ying-Nian Wu, who have been exceptional and thoughtful mentors throughout my PhD journey at UCLA. Their unwavering support has been invaluable, both academically and personally. I will always cherish my conversations with Professor Wu, who not only addressed my academic concerns but also provided much-needed guidance in my personal life. I fondly recall my in-depth discussions with Professor Terzopoulos about career goals and academic history, as well as Professor Zhu’s insightful research advice and his humor that lightened many moments. Their mentorship has been a cornerstone of my growth during these formative years.

For serving on my thesis committee, I would also like to express my gratitude to Professor Quanquan Gu and to Professor Adnan Darwiche who also gave me the opportunity to be his teaching assistant twice and fostered a wonderful friendship with his research group. His trust and guidance have been deeply motivating.

I thank each member of my committee for their insightful advice and valuable discussions. Their inputs have been pivotal in shaping my work and my growth as a researcher.

I am profoundly thankful to Dr. Xiaofeng Gao for his supervision during my internship at Amazon and his tremendous support on the TeamCraft project. His high standards and passion for research greatly accelerated my growth as an academic researcher. I am also deeply grateful to Dr. Skyler Zheng for her invaluable assistance and encouragement during my time at Amazon.

I am deeply grateful for the mentorship of Dr. Fangwei Zhong, whose invaluable guidance on SocialGFs and insightful discussions about multi-agent systems have been instrumental to my growth.

I extend my heartfelt thanks to my wonderful co-authors and labmates. Zhi Li and Dr. Ran Gong made significant contributions to the development of TeamCraft, and Ruoyan Li was instrumental in helping me create the Inverse Attention Agent. Their collaboration

and camaraderie have been truly inspiring.

To my family, especially my mother, I owe endless thanks for their unwavering support and love. My mother's guidance and comforting presence have carried me through this journey.

I would not be where I am today without the support of these incredible individuals. Their guidance, encouragement, and belief in me have made all the difference, and I am forever indebted to them.

VITA

2014–2018 BS (Electrical Engineering), Fudan University, Shanghai, China
2018–2020 MS (Computer Science), Carnegie Mellon University, Pittsburgh, PA
2020–2024 PhD Candidate (Computer Science), UCLA, Los Angeles, CA
2023 Intern, Amazon, Inc., San Jose, CA
2021–2024 Teaching Associate, Computer Science Department, UCLA, Los Angeles, CA

PUBLICATIONS

* Equal contribution and co-primary authorship.

Qian Long*, Ruoyan Li*, Minglu Zhao*, Tao Gao, Demetri Terzopoulos, “Inverse Attention Agent in Multi-Agent System,” *arXiv:2410.21794*, October 2024, pgs. 1–16

Qian Long*, Zhi Li, Ran Gong, Ying Nian Wu, Demetri Terzopoulos, Xiaofeng Gao, “Teamcraft: A Benchmark for Embodied Multi-Agent Systems in Minecraft,” in review by the *2025 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*.

Qian Long, Fangwei Zhong, Mingdong Wu, Yizhou Wang, Song-Chun Zhu, “SocialGFs: Learning Social Gradient Fields for Multi-Agent Reinforcement Learning,” *arXiv:2405.01839*, February 2024, pgs. 1–13.

Jiachen Li, **Qian Long**, Jian Zheng, Xiaofeng Gao, Robinson Piramuthu, Wenhua Chen, William Yang Wang, “T2V-Turbo-v2: Enhancing Video Quality with Data, Reward Model Selection and Motion Prior Distillation,” in review by the *2025 International Conference on Learning Representations (ICLR)*.

Yizhou Zhao, Yuanhong Zeng, **Qian Long**, Ying Nian Wu, Song-Chun Zhu, “Sim2Plan: RobotMotion Planning via Message Passing between Simulation and Reality,” *arXiv:2307.07862*, July 2023, pgs. 1–14.

Qian Long*, Zihan Zhou*, Abhinav Gupta, Fei Fang, Yi Wu, Xiaolong Wang, “Evolutionary Population Curriculum for Scaling Multi-Agent Reinforcement Learning,” *arXiv:2003.10423*, March 2020, pgs. 1–18.

CHAPTER 1

Introduction

Humans are inherently social beings, actively engaging in various social activities within a dynamic and interconnected world. A multi-agent system (MAS) serves as an effective simulator for this complexity, offering a framework to study relationships and interactions among agents. With MAS, we aim to replicate human capabilities and model real-world scenarios.

One of humanity’s greatest strengths is the ability to generalize and adapt to diverse environments. For example, humans can function effectively in both small groups and large populations, demonstrating remarkable adaptability to varying numbers of agents. This flexibility extends to tasks as well—when placed in new environments, humans draw on prior experiences to navigate unfamiliar situations. This task adaptability enables us to observe, learn, and adjust behavior as needed.

Furthermore, humans excel in adapting to interactions with different individuals, even those they’ve never met. Consider the game Dota, where five strangers from around the world collaborate to defeat an opposing team. Players quickly adapt to each other’s strategies and playstyles, achieving effective teamwork.

Lastly, humans thrive in a multi-modal world, seamlessly processing complex and ever-changing inputs from their surroundings. By integrating visual, auditory, and other sensory observations, we can navigate intricate environments. This capacity to manage and respond to diverse, complex stimuli highlights another dimension of our generalization prowess.

In this dissertation, we address four fundamental questions about agent design and adaptability in multi-agent systems:

1. How can we scale with the number of agents as complexity grows exponentially with each additional agent? We present the evolutionary population curriculum (Long et al., 2020), an approach that incrementally introduces new agents in successive rounds. This method effectively mitigates scalability challenges, enabling the system to handle larger populations.
2. How can agents adapt to new environments by leveraging past experiences? We introduce the SocialGFs representation (Long et al., 2024c), a high-level environmental abstraction that addresses the issue of sparse rewards. This representation allows agents to generalize and perform effectively in unseen environments by simply replacing the representations.
3. How can agents cooperate with unfamiliar peers, especially when trained in fixed groups? We tackle this challenge with the Inverse-Att agent (Long et al., 2024a), capable of inferring the attention of other agents. By dynamically updating its own attention and actions based on these inferences, the agent effectively adapts to interactions with unseen peers in dynamic settings.
4. How can agents operate within multi-modal input scenarios beyond simple vector-based representations? We present the TeamCraft Benchmark (Long et al., 2024b), a multi-modal, multi-agent benchmark inspired by Minecraft. This benchmark provides a rich, complex environment with diverse inputs, enabling the development and evaluation of advanced algorithms designed to handle multi-modal tasks.

Our work aims to push the boundaries of agent generalization and adaptability, drawing inspiration from human social and cognitive capabilities. The following sections will discuss each of the above four fundamental questions and list the contributions of this thesis in greater detail.

1.1 Scalability in Multi-Agent Systems

Many real-world problems involve interactions between multiple agents, and such problems become significantly more difficult in the presence of complex cooperation and competition among agents. Inspired by the tremendous success of deep Reinforcement Learning (RL) in single-agent applications, such as Atari games (Mnih et al., 2013), robotics manipulation (Levine et al., 2016), and navigation (Zhu et al., 2017; Wu et al., 2018; Yang et al., 2019), it has become a popular trend to apply deep RL techniques to multi-agent applications, including communication (Foerster et al., 2016; Sukhbaatar et al., 2016; Mordatch and Abbeel, 2018), traffic light control (Wu et al., 2017), physical combat (Bansal et al., 2018), and video games (Liu et al., 2019a; OpenAI, 2018).

A fundamental challenge for Multi-Agent Reinforcement Learning (MARL) is that, as the number of agents increases, the problem becomes significantly more complex and the variance of policy gradients can grow exponentially (Lowe et al., 2017). Despite advances in tackling this challenge via actor-critic methods (Lowe et al., 2017; Foerster et al., 2018), which utilize decentralized actors and centralized critics to stabilize training, recent works still scale poorly and are mostly restricted to less than a dozen agents. However, many real-world applications involve a moderately large population of agents, such as algorithmic trading (Wellman et al., 2005), sport team competition (Hausknecht and Stone, 2015), and humanitarian assistance and disaster response (Meier, 2015), where one agent should collaborate and/or compete with all other agents. When applying existing MARL algorithms directly to complex games with a large number of agents, the agents may fail to learn good strategies and end up with little interaction with other agents even when collaboration is significantly beneficial. Yang et al. (2018) proposed a provably converged mean-field formulation to scale up the actor-critic framework by feeding the state information and the *average value* of nearby agents' actions to the critic. However, this formulation strongly relies on the assumption that the value function for each agent can be *well approximated* by the mean of local pairwise interactions. This assumption often does not hold when the interactions between agents become complex, leading to a significant drop in performance.

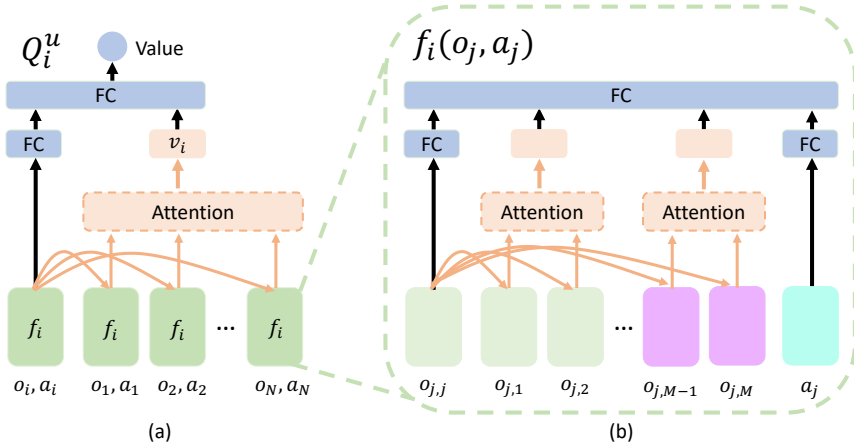


Figure 1.1: Our population-invariant Q function: (a) utilizes the attention mechanism to combine embeddings from different observation-action encoder f_i ; (b) is a detailed description for f_i , which also utilizes an attention module to combine M different entities in one observation.

We propose a general learning paradigm called *Evolutionary Population Curriculum* (EPC), which allows us to scale the number of agents exponentially. The core idea of EPC is to progressively increase the population of agents throughout the training process. In particular, we divide the learning procedure into multiple stages with increasing numbers of agents in the environment. The agents first learn to play in simpler scenarios with fewer agents and then leverage these experiences to gradually adapt to later stages with more agents and ultimately our desired population.

There are two key components in our curriculum learning paradigm: To process the varying number of agents during the curriculum procedure, the policy/critic must be *population-invariant*. So, we choose a self-attention-based architecture (Vaswani et al., 2017) that can generalize to an arbitrary number of agents with a fixed number of parameters. More importantly, we introduce an *evolutionary selection* process, which helps address the misalignment of learning goals across stages and improves agent performance in the target environment. Figure 1.1 illustrates the architecture of our paradigm.

Intuitively, our within-stage MARL training objective only incentivizes agents to overfit a particular population in the current stage. When moving towards a new stage with a larger population, the successfully trained agents may not adapt well to the scaled environment. To

mitigate this issue, we maintain multiple sets of agents in each stage, evolve them through cross-set mix-and-match and parallel MARL fine-tuning in the scaled environment, and select those with better adaptability to the next stage.

EPC is RL-algorithm agnostic and can be potentially integrated with most existing MARL algorithms. We illustrate the empirical benefits of EPC by implementing it on a popular MARL algorithm, MADDPG (Lowe et al., 2017), and experimenting on three challenging environments, including a predator-prey-style individual survival game, a mixed cooperative-and-competitive battle game, and a fully cooperative food collection game. We show that EPC outperforms baseline approaches by a large margin on all these environments as the number of agents grows even exponentially. We also demonstrate that our method can improve the stability of the training procedure.

We have published this work as (Long et al., 2020) and present it in [Chapter 3](#).

1.2 Task Adaptiveness in Multi-Agent Systems

A Multi-Agent System (MAS) is composed of multiple autonomous agents that interact with each other and the environment. It has many applications in domains such as games (Vinyals et al., 2019; OpenAI et al., 2019), social simulation (Vinitzky et al., 2022; Yaman et al., 2022), and distributed computing (Shalev-Shwartz et al., 2016). However, designing and learning MASs that can adapt to diverse and dynamic scenarios is a key challenge. For example, a MAS may need to cope with changes in the environment (such as obstacles or resources) and the number, goals, and roles of agents (such as teammates or adversaries). Therefore, it is desirable to build a MAS that can transfer agent knowledge across different settings.

Researchers into MARL have trained policies that can generalize across various aspects, such as multi-task performance (Wen et al., 2022; Omidshafiei et al., 2017), scalability (Agarwal et al., 2019; Long et al., 2020; Zhou et al., 2021; Hu et al., 2021), and communication (Omidshafiei et al., 2019; Jiang and Lu, 2018; Wang et al., 2022). Researchers have explored different techniques to enhance generalization, such as policy representation learning (Grover et al., 2018), meta learning (Chen et al., 2021), self-play (Zhong et al., 2021), curriculum

learning (Long et al., 2020), network architecture (Jiang and Lu, 2018; Xu et al., 2020). However, the agents learned by these methods are often tailored to specific tasks or environments and lack transferability and reusability.

The most important part of achieving such an adaptation ability in a MAS is to find a general and efficient way to represent complex environments. The concept of social force (Helbing and Molnár, 1995), borrowed from sociology, describes how individual behavior, interaction, and cognition are influenced by various factors in a social context. Examples of social force include attraction and repulsion among individuals, conformity and deviation from social norms, and cooperation and competition among groups. We argue that agents in a MAS are also subject to different types of forces that originate from the environment, other agents, and their intrinsic motivation. These forces can modulate the agents’ actions and strategies and ultimately determine their performance and adaptation.

The idea of using vector fields to represent forces has been explored in robotics, where artificial potential fields (Warren, 1989) or other gradient vector fields (Zhao et al., 2022) have been applied to various tasks such as environment simulations (Kolivand et al., 2021; Wan et al., 2014), robot navigation (Klančar et al., 2022; Konolige, 2000) and multi-agent path planning (Matoui et al., 2019; Zheng et al., 2015). However, these vector fields are often handcrafted and tailored to specific environments, limiting their generalizability and transferability. Furthermore, designing these vector fields explicitly is challenging in complex environments where multiple factors may influence the behavior of agents. Therefore, it is necessary to *learn these vector fields from data rather than specify them manually*.

We introduce Social Gradient Fields (SocialGFs), which are learned offline from examples of attractive or repulsive outcomes using denoising score matching (Song et al., 2021), a score-based generative modeling technique. SocialGFs represent these abstract forces as vector fields that guide the agents toward favorable states and away from unfavorable states. Tasks can be represented as compositions of gradients and often share common gradients, such as those for collision avoidance. Hence, these gradients can be reused across different tasks. Moreover, as the gradient fields are dense, they can provide informative guidance in sparse reward scenarios, addressing the credit assignment problem in multi-agent learning.

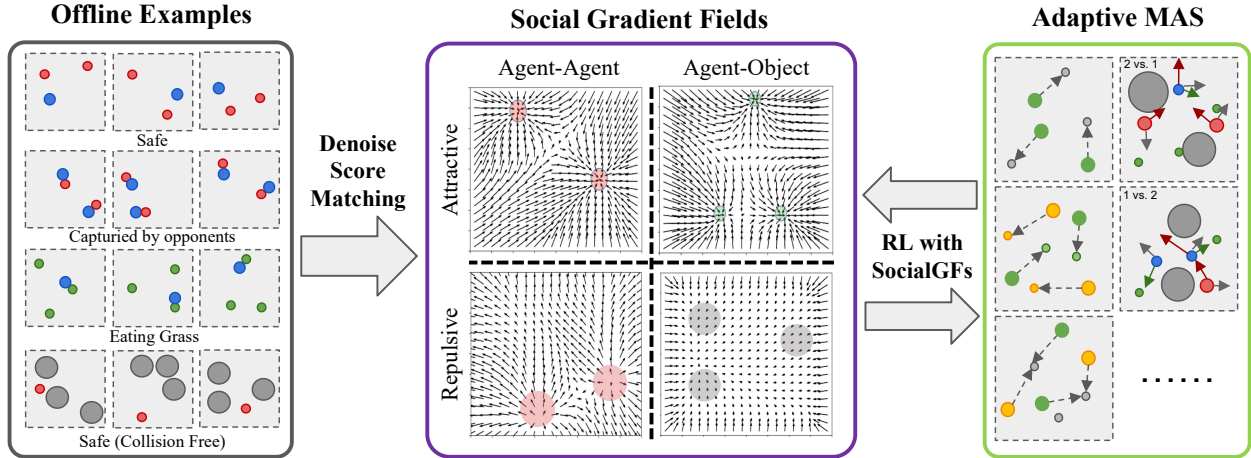


Figure 1.2: We use a score matching function to train each example, obtaining different gf functions. For various tasks, we select different sets of gf and apply them to the observation to generate a gf -based representation (SocialGFs). We then apply RL methods to train the adaptive agent based on that representation. By employing different gf functions for representation, the agent can adapt to various scenarios.

Furthermore, with graph neural networks, they can also scale easily with the number of entities. With these features, we can unify the representation in multi-agent environments across varying tasks and populations.

In the end, given these gradients to the agents, the agents need only learn the policy from the gradient-based state representation in an RL manner, rather than learning a state representation from scratch. Since the gradient fields can provide generalist representation among tasks and scenarios, agents can also adapt easily to a new environment by replacing the gradient-based representation. We show this pipeline in Figure 1.2.

To summarize, our main contributions are three-fold:

1. We propose learnable gradient fields, which are learned from offline examples by denoising score matching, for learning adaptive multi-agent policy.
2. We develop generalizable RL-based agents to learn to act based on the gradient fields and adapt to different scenarios.
3. We empirically demonstrate the effectiveness and generalization of our method in both the cooperative-competitive game and the cooperative game in a particle-world

environment.

We have published this work as (Long et al., 2024c) and present it in Chapter 4.

1.3 Agent Adaptiveness in Multi-Agent Systems

MARL has significantly advanced the study of complex, interactive behaviors in multi-agent systems, allowing intricate modeling of scenarios involving multiple autonomous agents. However, creating an ad-hoc agent that excels with various types of teammates and opponents poses a significant challenge. Current methods suffer a limitation: while agents trained together demonstrate proficient coordination, their performance deteriorates markedly when paired with unfamiliar agents.

To address this challenge, we delve deeper into multi-agent collaboration by incorporating a cognitive perspective, specifically through the modeling of attention and Theory of Mind (ToM) (Bratman, 1987) within the MARL framework, and we propose the *Inverse-Attention Agent*. Unlike classical ToM research, which focuses on attributing mental states such as beliefs and desires, our model shifts to the crucial yet less-emphasized component of *attention*. Our methodology adopts a mentalistic approach, explicitly modeling the internal attentional state of agents using an attention recognition neural network that can be trained end-to-end in combination with components of MARL. Contrary to traditional ToM modeling approaches that rely heavily on Bayesian inference to handle mental state transitions (Baker et al., 2009; Kleiman-Weiner et al., 2016; Shum et al., 2019; Gao et al., 2020; Tang et al., 2022), our method maintains the ontology of these states while focusing on the direct modeling of agents’ attentional mechanisms. This shift from Bayesian methods to more direct, attention-based modeling opens new pathways for understanding and enhancing agent interactions in complex environments.

To enhance task performance, we craft an agent that adeptly infers the attention of other agents — a crucial aspect of ToM. We employ gradient field functions to construct goals within the environment. Subsequently, we incorporate a self-attention architecture within the policy

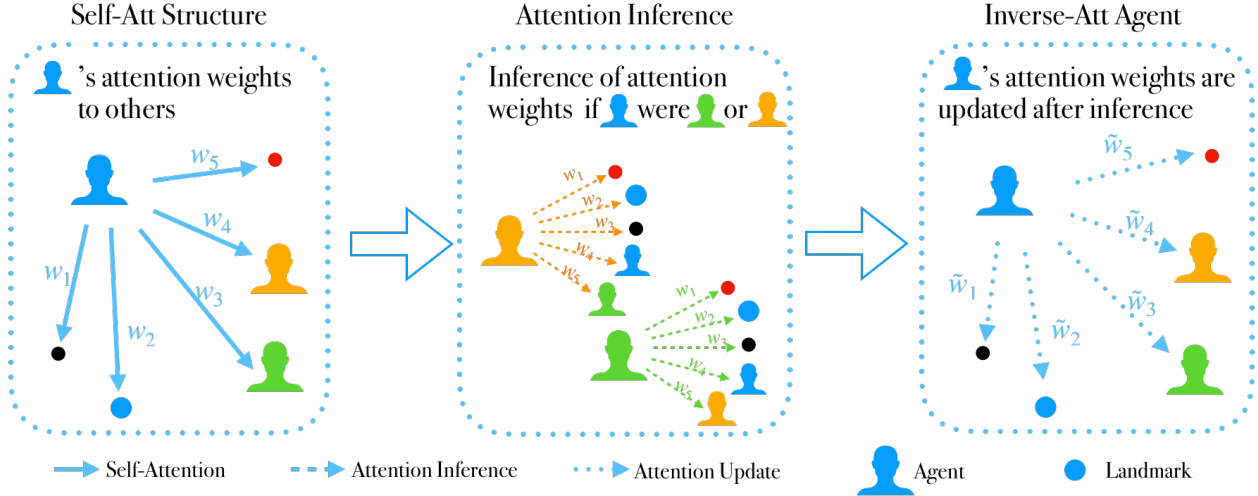


Figure 1.3: Pipeline for training the inverse attention agent: The first phase involves applying a self-attention mechanism, where the agent assigns attention weights to its observations and acts based on these weights. In the second phase, the agent performs attention inference on other agents of the same type using the inverse attention network. By placing itself in the position of these agents, it infers their attention weights, gaining insights into their goals and behaviors. In the final phase, the inverse attention agent uses the inferred information from the previous step to update its original attention weights, $\{w_1, w_2, \dots, w_n\}$ to $\{\tilde{w}_1, \tilde{w}_2, \dots, \tilde{w}_n\}$, consequently leading to changes in its final actions.

network to generate attention weights for these goals, guiding the agent’s actions accordingly. Throughout the training phase, we accumulate the weighted goals and corresponding actions into a training dataset. This dataset then serves to train an attention inference network that determines attention weights based on the observations and actions of other agents, effectively modeling their attention. In the final phase, we amalgamate the self-attention structure with the inverse attention to create our Inverse-Attention Agent, which integrates environmental observations with inferred attention weights from the inference network to fine-tune its final actions. The whole process is illustrated in Figure 1.3.

We demonstrate the effectiveness of our approach through a series of experiments adapted from the Multi-agent Particles Environment (MPE) (Lowe et al., 2017). Specifically, we employ a Mix-and-Match scheme to evaluate the performance of our models in ad-hoc collaboration, by pairing models trained using various algorithms. The results indicate that our model consistently outperforms baselines, particularly excelling in cooperative tasks. Moreover, we conduct a series of human experiments where humans join teams alongside

agents trained with different methods. Through both quantitative and qualitative analysis, our model demonstrates superior cooperation within ad-hoc teams comprising both humans and previously unseen agents.

To summarize, we introduce a MARL training scheme inspired by theories in cognitive science, where each agent explicitly reasons about the attentional states of group members. Our approach is specifically designed to enhance ad-hoc coordination among agents, addressing a long-standing challenge in MARL. Our framework also adopts a simplified state representation using social gradient fields, which further reduces the complexity of the environmental input that agents must process, thereby facilitating more flexible and efficient decision making. By explicitly modeling attention and incorporating cognitive principles, our approach paves the way for more effective multi-agent collaboration in complex, interactive scenarios.

We have published this work as (Long et al., 2024a) and present it in Chapter 5.

1.4 A Benchmark for Multi-Modal Multi-Agent Systems

Developing collaborative skills is essential for embodied agents, as collaboration is a fundamental aspect of human intelligence (Smith and Gasser, 2005). In the AI community, multi-agent collaboration is frequently studied using grid-world environments (Leibo et al., 2021; Suarez et al., 2021; Stone and Veloso, 2000; Gong et al., 2023c; Dong et al., 2024; Puig et al., 2021; Park et al., 2023; Zhang et al., 2024a; Wu et al., 2021; Long et al., 2024a). However, agents in these environments lack multi-modal understanding. By contrast, learning within visually-rich environments enables agents to develop useful representations of multi-agent dynamics (Chen et al., 2020; Jaderberg et al., 2019), as vision facilitates implicit communication, coordination, and collaborative execution (Jain et al., 2020, 2019).

Learning vision-based, multi-task, multi-agent systems is a challenging objective that presents several difficulties. These systems must develop detailed scene understanding to handle the diverse visual appearances of scenes. The complexity is further heightened by the numerous combinations of task configurations, such as object spatial arrangements, goal configurations, arbitrary numbers of agents, and heterogeneous agent capabilities. Consequently,

it is essential for multi-agent systems to acquire generalizable skills that can be effectively transferred across different settings.

An important step in addressing these challenges is to develop simulation systems that support multi-modal multi-agent learning. Recent advances in simulated environments have significantly facilitated progress in embodied vision-based systems (Yu et al., 2024; Jain et al., 2020; Chen et al., 2020; Perez-Liebana et al., 2019; Das et al., 2019). Despite notable progress, these systems have several limitations: (1) many of them target one or two-agent scenarios (Jain et al., 2019; Mandi et al., 2024; Wang et al., 2023a), (2) they are often limited to indoor settings with a narrow range of tasks (Puig et al., 2021; Zhang et al., 2024c), and (3) the task specifications are generally purely in text (Liu et al., 2022b; Mandi et al., 2024), making it hard to specify subtle task differences accurately and efficiently.

To drive progress in this area, we have developed a comprehensive benchmark, named *TeamCraft*, that features procedurally generated large-scale datasets specifically designed for multi-modal multi-agent systems. This benchmark utilizes the widely acclaimed open-world video game Minecraft as an experimental platform to engage with the complex dynamics of multi-modal multi-agent interactions. Inspired by the work of Jiang et al. (2022), we also leverage multi-modal prompts as task specifications to guide agent interactions, as language often fails to effectively convey spatial information (Cai et al., 2024). Our benchmark offers rich visual backgrounds, diverse object categories, complex crafting sequences, and varying task dynamics. These features enable systematic exploration of out-of-distribution generalization challenges for multi-modal, multi-task, multi-agent systems at scale. In particular, our benchmark evaluates a model’s ability to generalize to novel goal configurations, unseen number of agents, novel agent capabilities, and new types of visual backgrounds. To evaluate existing techniques using our benchmark, we design several baseline models to work within the framework and compare their performance. Our results highlight that current approaches to vision-conditioned collaboration and task planning encounter significant challenges when tested within *TeamCraft*’s complex and dynamic environment, especially when it comes to generalizations. The overall structure of *TeamCraft* is shown in Figure 1.4.

In summary, our main contributions are as follows:

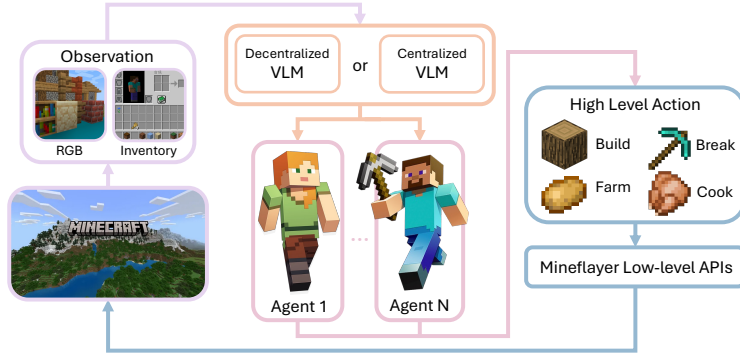


Figure 1.4: The *TeamCraft* platform consists of three main components: (1) a Minecraft server that hosts the game as an online platform, (2) Mineflayer, which serves as the interface for controlling agents in the server, and (3) a Gym-like environment that provides RGB and inventory observations to the models, allowing control of multiple agents through high-level actions.

1. *TeamCraft*, a new multi-modal multi-agent benchmark with its associated large-scale dataset encompassing complex tasks challenging multi-agent systems in a wide variety of generalization scenarios.
2. Extensive experiments and analyses on state-of-the-art multi-modal multi-agent models, uncovering their strengths and weaknesses to inform and inspire future research.
3. To ensure reproducibility and encourage future work from the community, we open source the entire platform, its training and evaluation code, and release the model checkpoints and training data.¹

We have published this work as (Long et al., 2024b) and present it in Chapter 6.

¹<https://github.com/teamcraft-bench/teamcraft>

CHAPTER 2

Related Work

In this chapter, we review prior work related to the main research topics relevant to this thesis; specifically, multi-agent reinforcement learning, attention-based policies, curriculum learning, evolutionary learning, learning adaptive multi-agent systems, social forces in multi-agent systems, gradient fields for decision making, theory of mind and attention, ad-hoc teaming, platforms for multi-agent systems, embodied language-guided benchmarks, and benchmarks based on Minecraft.

2.1 Multi-Agent Reinforcement Learning

There has been a long history in applying RL to multi-agent games (Littman, 1994; Shoham et al., 2003; Panait and Luke, 2005; Wright et al., 2019). Recently, deep RL techniques have been applied into the multi-agent scenarios to solve complex Markov games and great algorithmic advances have been achieved. Foerster et al. (2016) and He et al. (2016) explored a multi-agent variant of deep Q-learning; Peng et al. (2017) studied a fully centralized actor-critic variant; Foerster et al. (2018) developed a decentralized multi-agent policy gradient algorithm with a centralized baseline; Lowe et al. (2017) proposes the MADDPG algorithm which extended DDPG to the multi-agent setting with decentralized policies and centralized Q functions. Our population curriculum approach is a general framework for scaling MARL which can be potentially combined with any of these algorithms. Particularly, we implement our method on top of the MADDPG algorithm in this paper and take different MADDPG variants as baselines in experiments. There are also other works studying large-scale MARL recently (Lin et al., 2018; Jiang and Lu, 2018; Yang et al., 2018; Suarez et al., 2019), which

typically simplify the problem by weight sharing and taking only local observations. We consider a much more general setting with global observations and unshared-weight agents. Additionally, our approach is a general learning paradigm which is complementary to the specific techniques proposed in these works.

2.2 Attention-Based Policy Architecture

Attention mechanism is widely used in RL policy representation to capture object level information (Duan et al., 2017; Wang et al., 2018), represent relations (Zambaldi et al., 2018; Malysheva et al., 2018; Yang et al., 2019) and extract communication channels (Jiang and Lu, 2018). Iqbal and Sha (2019) use an attention-based critic. In our work, we utilize an attention module in both policy and critic, inspired by the transformer architecture (Vaswani et al., 2017), for the purpose of generalization to an arbitrary number of input entities.

2.3 Curriculum Learning

Curriculum learning can be tracked back to Elman (1993), and its core idea is to “start small”: learn the easier aspects of the task first and then gradually increase the task difficulty. It has been extended to deep neural networks on both vision and language tasks (Bengio et al., 2009) and much beyond: Karras et al. (2017) propose to progressively increase the network capacity for synthesizing high quality images; Murali et al. (2018) apply a curriculum over the control space for robotic manipulation tasks; several works (Wu and Tian, 2016; Florensa et al., 2017; Sukhbaatar et al., 2017; Wang et al., 2019) have proposed to first train RL agents on easier goals and switch to harder ones later. Baker et al. (2019) show that multi-agent self-play can also lead to autotricula in open-ended environments. In our paper, we propose to progressively increase the number of the agents as a curriculum for better scaling multi-agent reinforcement learning.

2.4 Evolutionary Learning

Evolutionary algorithms, originally inspired by Darwin’s natural selection, has a long history (Bäck and Schwefel, 1993), which trains a population of agents in parallel, and let them evolve via crossover, mutation and selection processes. Recently, evolutionary algorithms have been applied to learn deep RL policies to improve training scalability (Jaderberg et al., 2017; Salimans et al., 2017), promote diversity (Houthoofd et al., 2018) and encourage exploration (Conti et al., 2018; Khadka and Tumer, 2018). Leveraging this insight, we train several groups of agents in parallel and keep evolving them to larger population for the purpose of better generalization across stages and improved training stability. Czarnecki et al. (2018) proposed a similar evolutionary mix-and-match training paradigm to progressively increase agent capacity, i.e., larger action spaces and more parameters. Their work considers a fixed environment with an increasingly more complex agent and utilizes the traditional parameter crossover and mutation during evolution. By contrast, we focus on scaling MARL, namely an increasingly more complex environment with a growing number of agents. More importantly, we utilize MARL fine-tuning as an implicit mutation operator rather than the classical way of mutating parameters, which is more efficient, guided and applicable to even a very small number of evolution individuals. A similar idea of using learning for mutation is also considered by Gangwani and Peng (2018) in the single-agent setting.

2.5 Learning Adaptive Multi-Agent Systems

Adaptive multi-agent systems consist of multiple agents that can learn from their interactions and adapt to changing environments and goals. One of the challenges is how to coordinate the agents to achieve a new goal while transferring to a novel environment. Agarwal et al. (2019) used graph neural networks to handle the dynamic size of input. Long et al. (2020) used the attention mechanism and curriculum learning method to start from a small population environment and then adapt to large populations. Yang et al. (2020) applied mean-field theory to incorporate with large number of entities. Zhou et al. (2021) and Hu et al. (2021)

applied a transformer to deal with variant inputs. Liu et al. (2019b) used policy distillation (Rusu et al., 2016) for transfer learning among similar tasks. However, all of these methods rely on the high similarities between tasks. It would be hard for them to adapt to a completely new environment.

2.6 Social Forces in Multi-Agent Systems

Social force model (SFM) is a powerful tool for simulating the behavior of agents in a variety of environments (Kolivand et al., 2021; Wan et al., 2014; Chen et al., 2018; Huang et al., 2018). For example, Helbing and Molnár (1995) used SFM to simulate pedestrian behavior in crowds, where pedestrians adjust their movement to reach a stable state in response to repulsive and driving forces. In this way, SFM replicates observed behaviors like lane formation and collision avoidance. Gil et al. (2021) also applied SFM with machine learning techniques to build a social robot. However, SFM is sensitive to the parameters of the models and is difficult to calibrate and validate. Another disadvantage of SFM is that they are not always generalizable to new environments. This is because SFM is usually manually designed and defined based on empirical observations of real-world agents, which makes it expensive or impossible to transfer. To address these limitations, we propose a new data-driven approach that is based on the examples and use the gradient field to represent the social force. This approach is more generalizable to complicated environments because it does not rely on empirical observations of real-world agents.

2.7 Gradient Fields for Decision Making

The Gradient field is a vector field that describes the trend of change of a function or a distribution. Thus, it can be used to represent the social force in a multi-agent environment. It has been widely applied for decision-making like navigation tasks (Klančar et al., 2022; Konolige, 2000; Guldner and Utkin, 1995). Vail and Veloso (2003) applied gradient for role assignment based on different scenarios. Recent work by Zhao et al. (2022) introduced a

hybrid gradient vector field for path-following. All of the above methods are planning-based methods that require knowing the explicit expression of the target function or distribution or being able to estimate or approximate it effectively. Another fold of methods is Learning-based methods. They learn a gradient field from data to guide decision-making, without requiring knowing the explicit expression of the target function or distribution. These methods usually use deep neural networks to fit an energy function or a score function, thus implicitly defining a gradient field. For example, TarGF (Wu et al., 2022) uses denoising score matching (Song et al., 2021) to learn target gradient fields to rearrange objects without explicit goal specification. It learns a score function by minimizing the squared difference between the score of an energy function and the score of the true data distribution. Building on this, GraspGF (Wu et al., 2024), GFPose (Ci et al., 2022), and (Xue et al., 2023) expand this methodology to broader applications such as dexterous hand manipulation, human pose estimation, and the irregular shape packing. In this work, we also use denoising score matching to learn the gradient fields with a set of offline examples, but we use hybrid gradient fields to represent all the factors in the environments in multi-agent environments and further introduce an RL-based agent to act based on the gradient-based representation.

2.8 Theory of Mind and Attention

Theory of Mind (ToM) refers to the cognitive ability to attribute mental states to oneself and others (Bratman, 1987). It allows for tailored strategies to be generated in multi-agent scenarios, where one can reason about what other players are doing and determine one’s action accordingly. ToM has been an active area of research in multi-agent systems, with the goal of designing agents that coordinate more like humans. Previous work has primarily utilized Bayesian approaches to explicitly model beliefs, desires, and intentions (Shum et al., 2019; Kleiman-Weiner et al., 2016; Wu et al., 2021), providing a principled framework for inferring and updating beliefs about other agents’ mental states based on observed behavior and actions. To avoid the complexity of recurrently reasoning about each others’ mental state, Tang et al. (2020) and Stacy et al. (2021) proposed frameworks based on the idea of

shared agency, relying on coordinating group-level mental states and establishing a shared understanding of the task and goals among agents. Departing from Bayesian methods, [Rabinowitz et al. \(2018\)](#) proposed to integrate ToM reasoning directly into the neural network architecture, aiming to learn representations of other agents’ mental states in a data-driven manner.

Building upon previous frameworks, our work addresses three key aspects: First, we develop our model around the idea that ToM is not merely concerned with beliefs, desires, and intentions, but is a human-unique ability to be sensitive to what others are attending to. While attention as a critical mental state is often neglected in ToM research, our work aims to address this gap by explicitly modeling attention mechanisms. Second, we model cooperation from an individualistic perspective to encourage more flexible behavior generation. In this way, instead of shared agency, our agents maintain an individualistic perspective while coordinating with teammates. Third, we deviate from traditional Bayesian approaches to modeling ToM and instead develop an end-to-end neural network-based training, thus allowing for more flexibility and generalizability.

2.9 Ad-Hoc Teaming

Ad-hoc collaboration is defined as the challenge of enabling autonomous agents to effectively coordinate with previously unknown teammates, without any prior opportunities for coordination or agreement on strategies ([Stone et al., 2010](#)). Addressing this problem necessitates agents to model the behavior of their teammates and subsequently select actions that facilitate effective collaboration, while simultaneously adapting to changes or new information that emerges during the interaction. A prominent approach to modeling teammate behavior is type-based reasoning, which represents teammates as belonging to hypothesized behavior types ([Barrett et al., 2017](#)). Alternatively, neural network-based techniques have been proposed to infer teammate types from observations ([Rabinowitz et al., 2018](#); [Rahman et al., 2021](#); [Xie et al., 2021](#)). Once teammate models are obtained, agents perform downstream action planning with techniques such as Monte Carlo tree search ([Barrett et al., 2014](#); [Albrecht](#)

and Stone, 2019) and meta-learning action selection (Zintgraf et al., 2021). Adapting the agent’s behavior based on new information about teammates is also crucial in sustaining the collaboration. Addressing this, Macke et al. (2021) employed communication between agents and Lupu et al. (2021) proposed a method to generate diverse training trajectories to improve generalization to novel teammates.

In our work, we tackle the ad-hoc problem by leveraging the attention mechanism. We argue that the instability issues in ad-hoc settings typically arise because agents fail to comprehend unseen states, consequently making it difficult to generalize the trained policies. However, by implementing the attention mechanism, our agents are capable of selectively focusing on relevant aspects of the environment. This focused attention helps maintain consistency in decision-making across different scenarios (Cheng et al., 2023). Thus, the attention mechanism serves as a method for filtering information, which is crucial for achieving generalizability when interacting with previously unseen teammates.

2.10 Platforms for Multi-Agent Systems

The recent success of multi-agent reinforcement learning (MARL) methods (Lowe et al., 2020; Yu et al., 2021; Long et al., 2020, 2024c) has attracted attention, as these methods explore cooperation and competence behaviors among agents. However, many of the methods are evaluated in simplified 2D environments (Leibo et al., 2021; Suarez et al., 2021; Mordatch and Abbeel, 2017; Vinyals et al., 2019; Carroll et al., 2020). Recent work on embodied multi-agent benchmarks has considered more realistic tasks and environments (Liu et al., 2022a,b; Gong et al., 2023a; Park et al., 2023; Chang et al., 2024), but it often relies on certain privileged sensor information of the environment (Zhang et al., 2024b; Puig et al., 2021, 2023). Additionally, subject to environmental constraints, these works often have limited set of tasks (Jain et al., 2019; Tan et al., 2020) related to navigation and simple interactions such as object rearrangement (Szot et al., 2021). By comparison, TeamCraft is based on Minecraft, a three-dimensional, visually rich open-world realm characterized by procedurally generated landscapes and versatile game mechanics supporting an extensive spectrum of

Table 2.1: *TeamCraft* features visual observation for multi-agent control with widely-varied tasks specified by multi-modal prompts, targeting various types of generalization essential for multi-agent teaming. **MM Spec.:** multi-modal task specification. **Observation:** **V** for visual observation and **S** for state-based observation. **MA:** multi-agent control, **C** for centralized and **D** for decentralized. **Interaction:** object interaction. **Tool:** tool use. **Generalization:** types of generalization targeted, **E** for generalization on novel environments or scenes, **G** for novel goals, **A** for novel numbers of agents. **# Variants:** number of task variants involved.

Benchmark	MM Spec.	3D	Observation	MA	Interaction	Tool	Generalization	# Agents	# Variants	# Demonstrations
ALFRED (Shridhar et al., 2020a)	✗	✓	V	✗	✓	✓	E	1	2,600+	8,000+
FurnMove (Jain et al., 2020)	✗	✓	V	CD	✓	✗	E	2	30	✗
Marlo (Perez-Liebana et al., 2019)	✗	✓	V	D	✓	✗	✗	4+	14	✗
MineDojo (Fan et al., 2022)	✗	✓	V	✗	✓	✓	EG	1	3,000+	740,000+
MindAgent (Gong et al., 2023c)	✗	✓	VS	C	✓	✓	✗	4+	39	✗
Neural MMO 2.0 (Suárez et al., 2024)	✗	✗	S	CD	✓	✓	EGA	128+	25+	✗
Overcooked-AI (Carroll et al., 2020)	✗	✓	VS	C	✓	✓	✗	2	5	80
PARTNR (Chang et al., 2024)	✗	✓	VS	CD	✓	✓	E	2	100,000+	100,000+
RoCoBench (Mandi et al., 2024)	✗	✓	S	CD	✓	✓	G	2	6	✗
VIMA-Bench (Jiang et al., 2022)	✓	✓	V	✗	✓	✓	EG	1	1,000+	600,000+
Watch&Help (Puig et al., 2021)	✗	✓	S	CD	✓	✓	EG	2	1,200+	6,300+
TeamCraft	✓	✓	VS	CD	✓	✓	EGA	4+	55,000+	55,000+

object interactions, providing rich activities ripe for intricate collaborations.

2.11 Embodied Language-Guided Benchmarks

Several researchers have looked at the problem of using natural language as the interface between embodied agents, either in the form of task specifications (Shridhar et al., 2020b,a; Zheng et al., 2022; Gong et al., 2023b), question answering (Das et al., 2018; Gordon et al., 2018; Ma et al., 2023; Majumdar et al., 2024), instruction following (Anderson et al., 2018; Narayan-Chen et al., 2019; Jayannavar et al., 2020; Gao et al., 2022; Padmakumar et al., 2022; Wan et al., 2022; Gao et al., 2023), or as means of task coordination (Li et al., 2023; Mandi et al., 2024). VIMA-Bench (Jiang et al., 2022) builds on previous efforts in language-guided robotic manipulation (Zeng et al., 2020; Shridhar et al., 2021; Mees et al., 2022) and uses multi-modal prompts as uniform task specifications for object manipulation. *TeamCraft* extends multi-modal prompts to the multi-agent domain and uses them to specify a wide variety of collaborative tasks that require object interaction and navigation.

2.12 Benchmarks Based on Minecraft

Malmo (Johnson et al., 2016) marks the advent of a Gym-style platform tailored to Minecraft games. It paves the way for subsequent single-agent works such as MineRL (Guss et al., 2019), Voyager (Wang et al., 2023a), and MineDojo (Fan et al., 2022). Marlo (Perez-Liebana et al., 2019) extends Malmo to multi-agent scenarios, but the small number of task variations limit generalizations. Similar to our work, MindAgent (Gong et al., 2023c) and VillagerBench (Dong et al., 2024) focus on multi-agent collaboration in a multi-task setting. However, both of these use purely state-based observations, while *TeamCraft* tackles the more challenging problem of learning to collaborate from multi-modal perceptions. Table 2.1 compares *TeamCraft* with prior benchmarks.

CHAPTER 3

Evolutionary Population Curriculum for Scaling MARL

3.1 Evolutionary Population Curriculum

In this section, we will first describe the base network architecture with the self-attention mechanism (Vaswani et al., 2017) which allows us to incorporate a flexible number of agents during training. Then we will introduce the population curriculum paradigm and the evolutionary selection process.

3.1.1 Population-Invariant Architecture

We describe our choice of architecture based on the MADDPG algorithm (Lowe et al., 2017), which is population-invariant in the sense that both the Q function and the policy can take in an arbitrary number of input entities. We first introduce the Q function (Figure 1.1) and then the policy.

We adopt the decentralized learning framework, so each agent has its own Q function and policy network. Particularly for agent i , its centralized Q function is represented as follows:

$$Q_i^\mu(\mathbf{x}, a_1, \dots, a_N) = h_i([g_i(f_i(o_i, a_i)), v_i]), \text{ where } v_i = \text{attention}(f_i(o_j, a_j) \forall j \neq i) \quad (3.1)$$

Here $f_i(o_j, a_j)$ is an *observation-action encoder* (the green box in Figure 1.1(a)) which takes in the observation o_j and the action a_j from agent j , and outputs the agent embedding of agent j ; v_i denotes the *global attention embedding* (the orange box in Figure 1.1(a)) over all the agent embeddings. We will explain v_i and f_i later. g_i is a 1-layer fully connected network processing the embedding of the i th agent’s own observation and action. h_i is a 2-layer fully

connected network that takes the concatenation of the output of g_i and the global attention embedding v_i and outputs the final Q value.

Attention Embedding v_i : We define the attention embedding v_i by a weighted sum of each agent’s embedding $f_i(o_j, a_j)$ for $j \neq i$:

$$v_i = \sum_{j \neq i} \alpha_{i,j} f_i(o_j, a_j) \tag{3.2}$$

The coefficient $\alpha_{i,j}$ is computed by

$$\alpha_{i,j} = \frac{\exp(\beta_{i,j})}{\sum_{j \neq i} \exp(\beta_{i,j})}, \quad \beta_{i,j} = f_i^T(o_i, a_i) W_\psi^T W_\phi f_i(o_j, a_j) \tag{3.3}$$

where W_ψ and W_ϕ are parameters to learn. $\beta_{i,j}$ computes the correlation between the embeddings of agent i and every other agent j via an inner product. $\alpha_{i,j}$ is then obtained by normalizing $\beta_{i,j}$ by a softmax function. Since we represent the observations and actions of other agents with a weighted mean v_i from (3.2), we can model the interactions between agent i and an arbitrary number of other agents, which allows us to easily increase the number of agents in our curriculum training paradigm.

Observation-Action Encoder f_i : We now define the structure of $f_i(o_j, a_j)$ (Figure 1.1(b)). Note that the observation of agent j , o_j , also includes many entities, i.e., states of all visible agents and objects in the game. Suppose o_j contains M entities, i.e., $o_j = [o_{j,1}, \dots, o_{j,M}]$. M may also vary as the agent population scales over training procedure or simply during an episode when some agents die. Thus, we apply another attention module to combine these entity observations together in a similar way to how v_i is computed (Equations (3.2) and (3.3)).

In more details, we first apply an entity encoder for each entity type to obtain *entity embeddings* of all the entities within that type. For example, in o_j , we can have embeddings for agent entities (green boxes in Figure 1.1(b)) and landmark/object entities (purple boxes in Figure 1.1(b)). Then we apply an attention module over each entity type by attending the entity embedding of agent j to all the entities of this type to obtain an attended *type*

embedding (the orange box in Figure 1.1(b)). Next, we concatenate all the type embeddings together with the entity embedding of agent j as well as its action embedding. Finally, this concatenated vector is forwarded to a fully connected layer to generate the output of $f_i(o_j, a_j)$. Note that in the overall critic network of agent i , the same encoder f_i is applied to every observation-action pair so that the network can maintain a fixed size of parameters even when the number of agents increases significantly.

Policy Network: The policy network $\mu_i(o_i)$ has a similar structure as the observation-action encoder $f_i(o_i, a_i)$, which uses an attention module over the entities of each type in the observation o_i to adapt to the changing population during training. The only difference in this network is that the action a_i is not included in the input. Notably, we do not share parameters between the Q function and the policy.

3.1.2 Population Curriculum

We propose to progressively scale the number of agents in MARL with a curriculum. Before combining with the evolutionary selection process, we first introduce a simpler version, the vanilla population curriculum (PC), where we perform the following stage-wise procedure: (i) the initial stage starts with MARL training over a small number of agents using MADDPG and our population-invariant architecture; (ii) we start a new stage and double¹ the number of agents by cloning each of the existing agents; (iii) apply MADDPG training on this scaled population until convergence; (iv) if the desired number of agents is not reached, go back to step (ii).

Mathematically, given N trained agents with parameters $\theta = \{\theta_1, \dots, \theta_N\}$ from the previous stage, we want to increase the number of the agents to $2N$ with new parameters $\tilde{\theta} = \{\tilde{\theta}_1, \dots, \tilde{\theta}_N, \dots, \tilde{\theta}_{2N}\}$ for the next stage. In this vanilla version of population curriculum, we simply initialize $\tilde{\theta}$ by setting $\tilde{\theta}_i \leftarrow \theta_i$ and $\tilde{\theta}_{N+i} \leftarrow \theta_i$, and then continue MADDPG training on $\tilde{\theta}$ to get the final policies for the new stage. Although $\tilde{\theta}_i$ and $\tilde{\theta}_{N+i}$ are both initialized

¹Generally, we can scale up the population with any constant factor by introducing any amount of cloned agents. We use the factor of 2 as a concrete example here for easier understanding.

from θ_i , as training proceeds, they will converge to different policies since these policies are trained in a decentralized manner in MADDPG.

3.1.3 Evolutionary Selection

Introducing new agents by directly cloning existing ones from the previous stage has a clear limitation: the policy parameters suitable for the previous environment are not necessarily the best initialization for the current stage as the population is scaled up. In the purpose of better performance in the final game with our desired population, we need to promote agents with better adaptation abilities during early stages of training.

Therefore, we propose an evolutionary selection process to facilitate the agents’ scaling adaption ability during the curriculum procedure. Instead of training a single set of agents, we maintain K parallel sets of agents in each stage, and perform crossover, mutation and selection among them for the next stage. This is the last piece in our proposed *Evolutionary Population Curriculum* (EPC) paradigm, which is essentially population curriculum enhanced by the evolutionary selection process.

Specifically, we assume the agents in the multi-agent game have Ω different roles. Agents in the same role have the same action set and reward structure. For example, we have $\Omega = 2$ roles in a predator-prey game, namely predators and prey, and $\Omega = 1$ role of agents for a fully cooperative game with homogeneous agents. For notation conciseness, we assume there are N_1 agents for of role 1, namely $A_1 = \{\boldsymbol{\mu}_1, \dots, \boldsymbol{\mu}_{N_1}\}$; N_2 agents of role 2, namely $A_2 = \{\boldsymbol{\mu}_{N_1+1}, \dots, \boldsymbol{\mu}_{N_1+N_2}\}$, and so on. In each stage, we keep K parallel sets for each role of agents, denoted by $A_i^{(1)}, \dots, A_i^{(K)}$ for role i , and take a 3-step procedure, i.e., *mix-and-match* (*crossover*), *MARL fine-tuning* (*mutation*) and *selection*, as follows to evolved these K parallel sets of agents for the next stage.

Mix-and-Match (Crossover): In the beginning of a curriculum stage, we scale the population of agents from N to $2N$. Note that we have K parallel agent sets of size N_i for role i , namely $A_i^{(1)}, \dots, A_i^{(K)}$. We first perform a mix-and-match over these parallel sets *within*

every role i : for each set $A_i^{(j)}$, we pair it with all the K sets of the same role, which leads to $K(K + 1)/2$ new scaled agent sets of size $2N_i$. Given these scaled sets of agents, we then perform another mix-and-match *across* all the Ω roles: we pick one scaled set for each role and combine these Ω selected sets to produce a scaled game with $2N$ agents. For example, in the case of $\Omega = 2$, we can pick one agent set $A_1^{(k_1)}$ from the first role and another agent set $A_2^{(k_2)}$ from the second role to form a scaled game. Thus, there are $C_{\max} = (K(K + 1)/2)^\Omega$ different combinations in total through this mix-and-match process. We sample C games from these combinations for mutation in the next step. Since we are mixing parallel sets of agents, this process can be considered as the *crossover* operator in standard evolutionary algorithms.

MARL Fine-Tuning (Mutation): In standard evolutionary algorithms, mutations are directly performed on the parameters, which is inefficient in high-dimensional spaces and typically requires a large amount of mutants to achieve sufficient diversity for evolution. Instead, here we adopt MARL fine-tuning in each curriculum stage (step (iii) in vanilla PC) as our guided mutation operator, which naturally and efficiently explores effective directions in the parameter space. Meanwhile, due to the training variance, MARL also introduces randomness which benefits the overall diversity of the evolutionary process. Concretely, we apply parallel MADDPG training on each of the C scaled games generated from the mix-and-match step and obtain C mutated sets of agents for each role.

Selection: Among these C mutated sets of agents for each role, only the best K mutants can survive. In the case of $\Omega = 1$, the fitness score of a set of agents is computed as their average reward after MARL training. In other cases when $\Omega \geq 2$, given a particular mutated set of agents of a specific role, we randomly generate games for this set of agents and other mutated sets from different agent roles. We take its average reward from these randomly generated games as the fitness score for this mutated set. We pick the top- K scored sets of agents in each role to advance to the next curriculum stage.

Algorithm 1: Evolutionary Population Curriculum

Data: environment $E(N, \{A_i\}_{1 \leq i \leq \Omega})$ with N agents of Ω roles, desired population N_d , initial population N_0 , evolution size K , mix-and-match size C

Result: a set of N_d best policies

$N \leftarrow N_0$;

initialize K parallel agent sets $A_i^{(1)}, \dots, A_i^{(K)}$ for each role $1 \leq i \leq \Omega$;

initial parallel MARL training on K games, $E(N, \{A_i^{(j)}\}_{1 \leq i \leq \Omega})$ for $1 \leq j \leq K$;

while $N < N_d$ **do**

- $N \leftarrow 2 \times N$;
- for** $1 \leq j \leq C$ **do**
 - for each role $1 \leq i \leq \Omega$: $j_1, j_2 \leftarrow \text{unif}(1, K)$, $\tilde{A}_i^{(j)} \leftarrow A_i^{(j_1)} + A_i^{(j_2)}$
(*mix-and-match*);
 - MARL training in parallel on $E(N, \{\tilde{A}_i^{(j)}\}_{1 \leq i \leq \Omega})$ for $1 \leq j \leq C$ (*guided mutation*);
 - for** role $1 \leq i \leq \Omega$ **do**
 - for** $1 \leq j \leq C$ **do**
 - $S_i^{(j)} \leftarrow \mathbb{E}_{k_t \neq i \sim [1, C]} [\text{avg. rewards on } E(N, \{\tilde{A}_1^{(k_1)}, \dots, \tilde{A}_i^{(j)}, \dots, \tilde{A}_\Omega^{(k_\Omega)}\})]$
(*fitness*);
 - $A_i^{(1)}, \dots, A_i^{(K)} \leftarrow \text{top-}K \text{ w.r.t. } S_i \text{ from } \tilde{A}_i^{(1)}, \dots, \tilde{A}_i^{(C)}$ (*selection*);

return the best set of agents in each role, i.e., $\{A_i^{(k_i^*)} | k_i^* \in [1, K] \forall 1 \leq i \leq \Omega\}$;

Overall Algorithm: Finally, when the desired population is achieved, we take the best set of agents in each role based on their last fitness scores as the output. We conclude the detailed steps of EPC in [algorithm 1](#). Note that in the first curriculum stage, we just train K parallel games without mix-and-match or mutation. So, EPC simply selects the best from the K initial sets in the first stage while the evolutionary selection process only takes effect starting from the second stage. We emphasize that although we evolve multiple sets of agents in each stage, the three operators, mix-and-match, MARL fine-tuning and selection, are all *perfectly parallel*. Thus, the evolutionary selection process only introduces little influence on the overall training time. Lastly, EPC is an RL-algorithm-agnostic learning paradigm that can be potentially integrated with any MARL algorithm other than MADDPG.

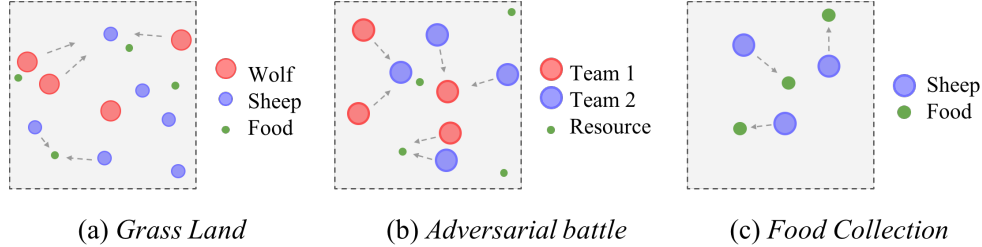


Figure 3.1: Environment Visualizations

3.2 Experiments

We experiment on three challenging environments, including a predatory-prey-style *Grassland* game, a mixed-cooperative-and-competitive *Adversarial Battle* game and a fully cooperative *Food Collection* game. We compare EPC with multiple baseline methods on these environments with different scales of agent populations and show consistently large gains over the baselines. In the following, we will first introduce the environments and the baselines, and then both qualitative and quantitative performances of different methods on all three environments.

3.2.1 Environments

All these environments are built on top of the particle-world environment (Mordatch and Abbeel, 2018) where agents take actions in discrete timesteps in a continuous 2D world.

Grassland: In this game, we have $\Omega = 2$ roles of agents, N_S sheep and N_W wolves, where sheep moves twice as fast as wolves. We also have a fixed amount of L grass pellets (food for sheep) as green landmarks Figure 3.1(a). A wolf will be rewarded when it collides with (eats) a sheep, and the (eaten) sheep will obtain a negative reward and becomes inactive (dead). A sheep will be rewarded when it comes across a grass pellet and the grass will be collected and respawned in another random position. Note that in this survival game, each individual agent has its own reward and does not share rewards with others.

Adversarial Battle: This scenario consists of L units of resources as green landmarks and two teams of agents (i.e., $\Omega = 2$ for each team) competing for the resources Figure 3.1(b).

Both teams have the same number of agents ($N_1 = N_2$). When an agent collects a unit of resource, the resource will be respawned and *all the agents in its team* will receive a positive reward. Furthermore, if there are more than *two* agents from team 1 collide with *one* agent from team 2, *the whole team 1* will be rewarded while the trapped agent from team 2 will be deactivated (dead) and the whole team 2 will be penalized, and vice versa.

Food Collection: This game has N food locations and N fully cooperative agents ($\Omega = 1$). The agents need to collaboratively occupy as many food locations as possible within the game horizon [Figure 3.1\(c\)](#). Whenever a food is occupied by any agent, the whole team will get a reward of $6/N$ in that timestep for that food. The more food occupied, the more rewards the team will collect.

In addition, we introduce collision penalties as well as auxiliary shaped rewards for each agent in each game for easier training. All the environments are fully observable so that each agent needs to handle a lot of entities and react w.r.t. the global state.

3.2.2 Methods and Metric

We evaluate the following approaches in our experiments: (1) the MADDPG algorithm (Lowe et al., 2017) with its original architecture (**MADDPG**); (2) the provably-converged mean-field algorithm (Yang et al., 2018) (**mean-field**); (3) the MADDPG algorithm with our population-invariant architecture (**Att-MADDPG**); (4) the vanilla population curriculum without evolutionary selection (**vanilla-PC**); and (5) our proposed EPC approach (**EPC**). For EPC parameters, we choose $K = 2$ for *Grassland* and *Adversarial Battle* and $K = 3$ for *Food Collection*; for the mix-and-match size C , we simply set it C_{\max} and enumerate all possible mix-and-match combinations instead of random sampling.

For *Grassland* and *Adversarial Battle* with $\Omega = 2$, we evaluate the performance of different methods by competing their trained agents against our EPC trained agents. Specifically, in *Grassland*, we let sheep trained by each approach compete with the wolves from EPC and collect the average sheep reward as the evaluation metric for sheep. Similarly, we take the

same measurement for wolves from each method. In *Adversarial Battle*, since two teams are symmetric, we just evaluate the shared reward of one team trained by each baseline against another team by EPC as the metric. For *Food Collection* with $\Omega = 1$, since it is fully cooperative, we take the team reward for each method as the evaluation metric. In addition, for better visualization, we plot the *normalized scores* by normalizing the rewards of different methods between 0 and 1 in each scale for each game.

3.2.3 Qualitative Results

We qualitatively illustrate the learned strategies by MADDPG and EPC in the three games here.

In *Grassland*, as the number of wolves goes up, it becomes increasingly more challenging for sheep to survive; meanwhile, as the sheep become more intelligent, the wolves will be incentivized to be more aggressive accordingly. In [Figure 3.2](#), we illustrate two representative matches for competition, including one using the MADDPG sheep against the EPC wolves ([Figure 3.2a](#)), and the other between the EPC sheep and the MADDPG wolves ([Figure 3.2b](#)). From [Figure 3.2a](#), we can observe that the MADDPG sheep can be easily eaten up by the EPC wolves (note that dark circle means the sheep is eaten). On the other hand, in [Figure 3.2b](#), we can see that the EPC sheep learns to eat the grass and avoid the wolves at the same time.

In *Adversarial Battle*, we visualize two matches in [Figure 3.3](#) with one over agents by EPC ([Figure 3.3a](#)) and the other over agents by MADDPG ([Figure 3.3b](#)). We can clearly see the collaborations between the EPC agents: although the agents are initially spread over the environment, they learn to quickly gather as a group to protect themselves from being killed. While for the MADDPG agents, their behavior shows little incentives to cooperate or compete — these agents stay in their local regions throughout the episode and only collect resources or kill enemies very infrequently.

In *Food Collection* ([Figure 3.4](#)), the EPC agents in [Figure 3.4a](#) learn to spread out and occupy as many food as possible to maximize the team rewards. While only one agent among the MADDPG agents in [Figure 3.4b](#) successfully occupies a food in the episode.

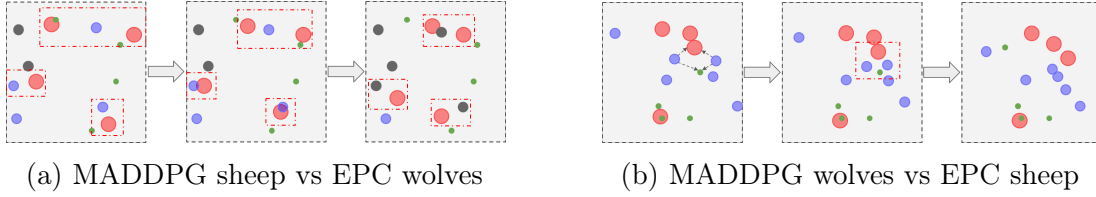


Figure 3.2: Example matches between EPC and MADDPG trained agents in *Grassland*

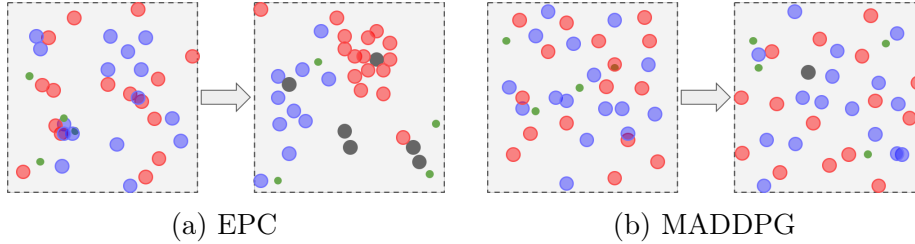


Figure 3.3: Quantitative result of adversarial battle

3.2.4 Quantitative Results

Quantitative Results in *Grassland*: In the *Grassland* game, we perform curriculum training by starting with 3 sheep and 2 wolves, and gradually increase the population of agents. We denote a game with N_S sheep and N_W wolves by “scale N_S - N_W ”. We start with scale 3-2 and gradually increase the game size to scales 6-4, 12-8 and finally 24-16. For the two curriculum learning approach, vanilla-PC and EPC, we train over 10^5 episodes in the first curriculum stage (scale 3-2) and fine-tune the agents with 5×10^4 episodes after mix-and-match in each of the following stage. For other methods that train the agents from scratch, we take the same accumulative training iterations as the curriculum methods for a fair comparison.

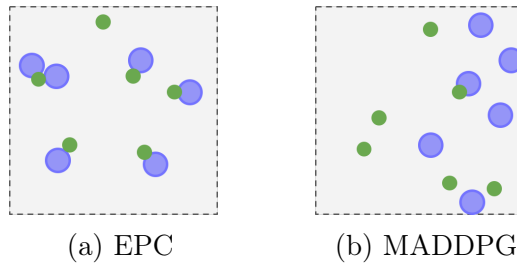


Figure 3.4: Quantitative result of food collection

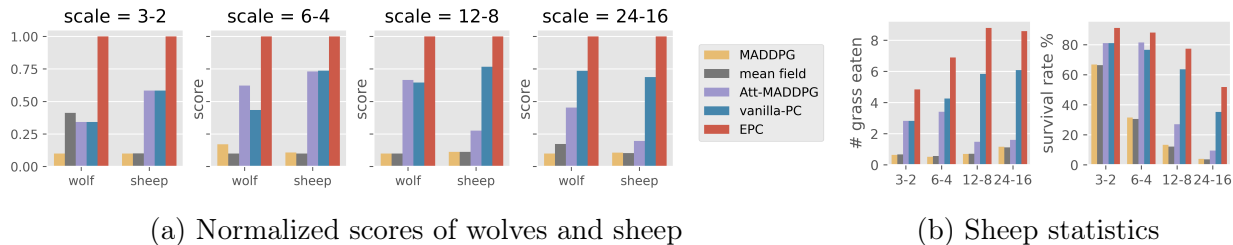


Figure 3.5: Quantitative result in grassland game. In part (a), we show the normalized scores of wolves and sheep trained by different methods when competing with EPC sheep and EPC wolves respectively. In part (b), we measure the sheep statistics over different scales (x-axis), including the average number of total grass pellets eaten per episode (left) and the average percentage of sheep that survive until the end of episode (right). EPC trained agents (yellow) are consistently better than any baseline method.

Main Results: We report the performance of different methods for each game scale in Figure 3.5a. Overall, there are little differences between the mean-field approach and the original MADDPG algorithm while the using the population-invariant architecture (i.e., Att-MADDPG) generally boosts the performance of MADDPG. For the method with population curriculum, vanilla-PC performs almost the same as training from scratch (Att-MADDPG) when the number of agents in the environment is small (i.e., 6-4) but the performance gap becomes much more significant when the population further grows (i.e., 12-18 and 24-16). For our proposed EPC method, it consistently outperforms all the baselines across all the scales. Particularly, in the largest scale 24-16, EPC sheep receive 10x more rewards than the best baseline sheep without curriculum training.

Detailed Statistics: Besides rewards, we also compute the statistics of sheep to understand how the trained sheep behavior in the game. We perform competitions between sheep trained by different methods against the EPC wolves and measure the average number of total grass pellets eaten per episode, i.e., *#grass eaten*, and the average percentage of sheep that survive until the end of an episode, i.e., *survival rate*, in Figure 3.5b. We can observe that as the population increases, it becomes increasingly harder for sheep to survive while EPC trained sheep remain a high survival rate even on the largest scale. Moreover, as more sheep in the game, EPC trained sheep consistently learn to eat more grass even under the strong

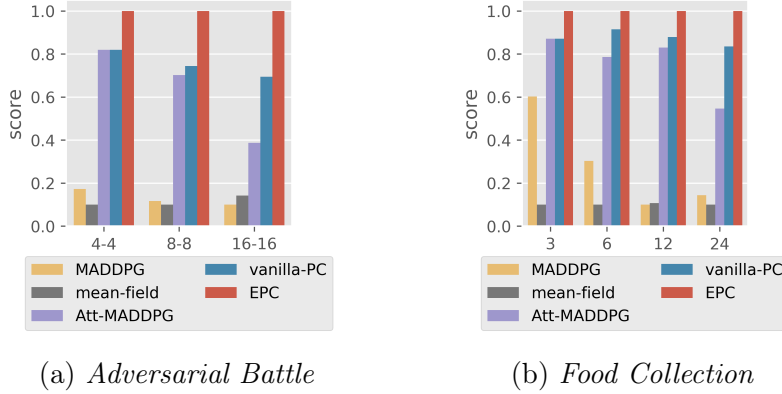


Figure 3.6: Quantitative results in adversarial battle game and food collection game

pressure from wolves. In contrast, the amount of eaten grass of MADDPG approach (i.e., Att-MADDPG) drastically decreases when the number of wolves becomes large.

Quantitative Results in *Adversarial Battle*: In this game, we evaluate on environments with different sizes of agent population N , denoted by scale N_1 - N_2 where $N_1 = N_2 = N/2$. We start the curriculum from scale 4-4 and then increase the population size to scale 8-8 ($N = 16$) and finally 16-16 ($N = 32$). Both vanilla-PC and EPC take 5×10^4 training episodes in the first stage and then 2×10^4 episodes in the following two curriculum stages. We report the normalized scores of different methods in Figure 3.6a, where agents trained by EPC outperform all the baseline methods increasingly more significantly as the agent population grows.

Quantitative Results in *Food Collection*: In this game, we begin curriculum training with $N = 3$, namely 3 agents and 3 food locations, and progressively increase the population size N to 6, 12 and finally 24. Both vanilla-PC and EPC perform training on 5×10^4 episodes on the first stage of $N = 3$ and then 2×10^4 episodes in each of the following curriculum stage. We report the normalized scores for all the methods in Figure 3.6b, where EPC is always the best among all the approaches with a clear margin. Note that the performance of the original MADDPG and the mean-field approach drops drastically as the population size N increases. Particularly, the mean-field approach performs even worse than the original MADDPG method. We believe this is because in this game, the agents must act according

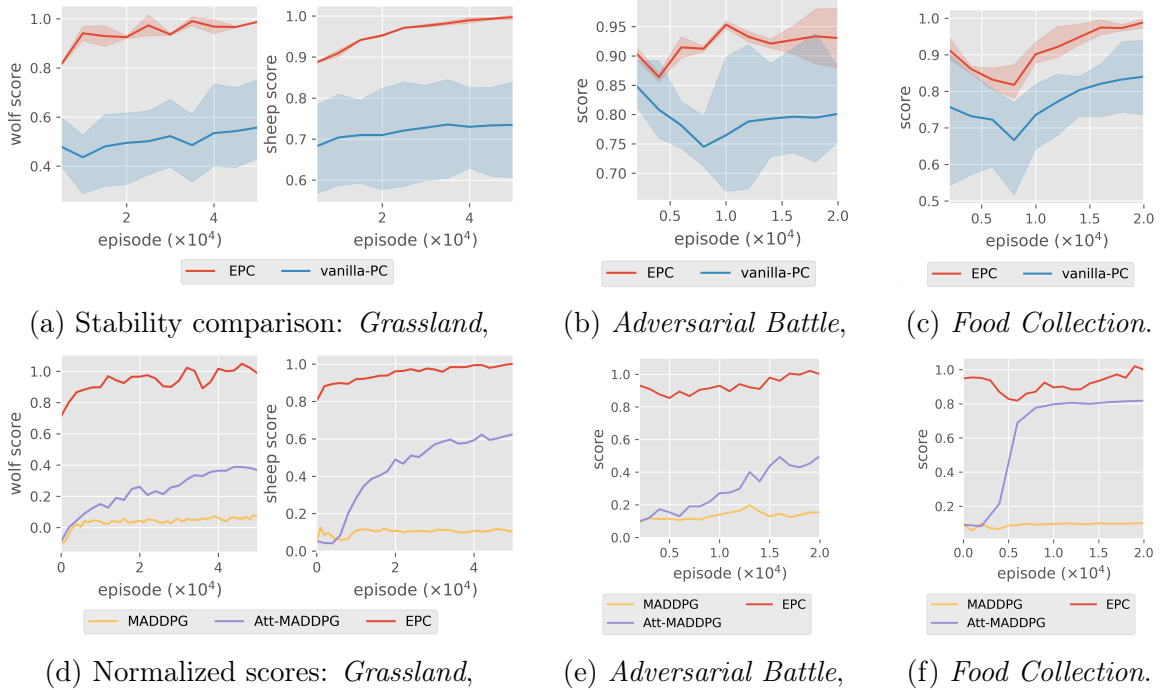


Figure 3.7: Ablation analysis on the second curriculum stage in all the games. Stability comparison (top) in (a), (b) and (c): We use 3 different seeds to perform training and observe EPC has much less variance comparing to vanilla-PC. Normalized scores during fine-tuning (bottom) in (d), (e) and (f): This illustrates that EPC can successfully transfer the agents trained with a smaller population to a larger population by fine-tuning.

to the global team state collaboratively, which means the local approximation assumption in the mean-field approach does not hold clearly.

Stability Analysis: The evolutionary selection process in EPC not only leads to better final performances but also stabilizes the training procedure. We validate the stability of EPC by computing the variance over 3 training seeds for the same experiment and comparing with the variance of vanilla-PC, which is also obtained from 3 training seeds. Specifically, we pick the second stage of curriculum learning and visualize the variance of agent scores throughout the stage of training. These scores are computed by competing against the final policy trained by EPC. We perform analysis on all the 3 environments: *Grassland* with scale 6-4 (Figure 3.7a), *Adversarial Battle* with scale 8-8 (Figure 3.7b) and *Food Collection* with scale 6 (Figure 3.7c). We can observe that the variance of EPC is much smaller than vanilla-PC in different games.

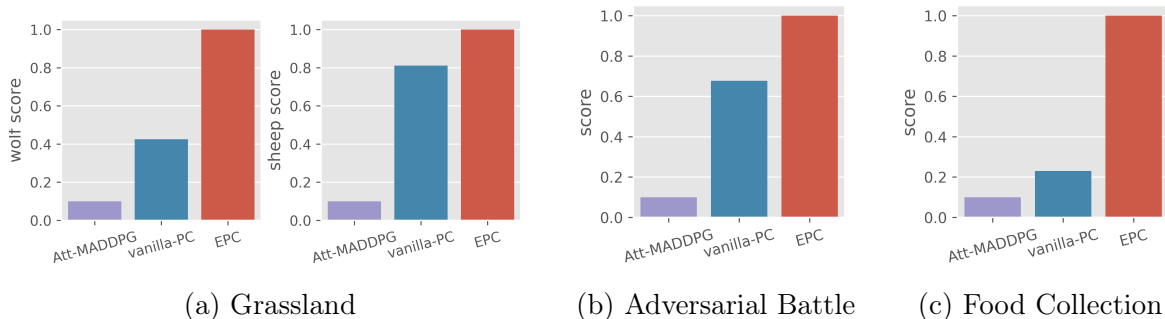


Figure 3.8: Environment Generalization: We take the agents trained on the largest scale and test on an environment with twice the population. We perform experiments on all the games and show that EPC also advances the agents’ generalizability.

Convergence Analysis: To illustrate that the self-attention based policies trained from a smaller scale is able to well adapt to a larger scale via fine-tuning, we pick a particular mutant by EPC in the second curriculum stage and visualize its learning curve throughout fine-tuning for all the environments, *Grassland* (Figure 3.7d), *Adversarial Battle* (Figure 3.7e) and *Food Collection* (Figure 3.7f). The scores are computed in the same way as the stability analysis. By comparing to MADDPG and Att-MADDPG, which train policies from scratch, we can see that EPC starts learning with a much higher score, continues to improve during fine-tuning and converges to a much better solution.

Generalization: We investigate whether the learned policies can generalize to a different test environment with even a larger scale than the training ones. To do so, we take the best polices trained by different methods on the largest population and directly apply these policies to a new environment with a doubled population by self-cloning. We evaluate in all the environments with EPC, vanilla-PC and Att-MADDPG and measure the normalized scores of different methods, which is computed in the same way as the fitness score. In all cases, we observe a large advantage of EPC over the other two methods, indicating the better generalization ability for policies trained by EPC.

CHAPTER 4

Learning Social Gradient Fields for Adaptive Multi-Agent Systems

4.1 Preliminary

4.1.1 Theory of Social Force

Theory of social force (Patten, 1896; Helbing and Molnár, 1995) assumes that agents are influenced by various social forces in their environment, such as attraction, repulsion, cohesion, or alignment. Social force theory can be formulated mathematically as a system of differential equations that describe the motion of each agent as a function of its position, velocity, and the forces exerted by other agents and external factors. The general form of the social force model is:

$$\frac{d\mathbf{v}_i}{dt} = f_i(\mathbf{x}_i, \mathbf{v}_i, \mathbf{x}_{-i}, \mathbf{v}_{-i}), \frac{d\mathbf{x}_i}{dt} = \mathbf{v}_i \quad (4.1)$$

where \mathbf{x}_i and \mathbf{v}_i are the position and velocity vectors of agent i , and f_i is the net force acting on agent i , which depends on its own state and the state of other agents ($\mathbf{x}_{-i}, \mathbf{v}_{-i}$). The specific form of f_i can vary depending on the domain and the assumptions made about the agent’s behavior and objectives. And the agents usually will be driven by multiple forces.

4.1.2 Learning Gradient Fields via Score-Matching

The score-based generative model aims to learn the *gradient field* of a log-data-density, *i.e.*, the *score function*. Given samples $\{\mathbf{x}_i\}_{i=1}^N$ from an unknown data distribution $\{\mathbf{x}_i \sim p_{\text{data}}(\mathbf{x})\}$, the goal is to learn a *score function* to approximate $\nabla_{\mathbf{x}} \log p_{\text{data}}(\mathbf{x})$ via a *score network*

$\mathbf{s}_\theta(\mathbf{x}) : \mathbb{R}^{|\mathcal{X}|} \rightarrow \mathbb{R}^{|\mathcal{X}|}$.

$$\mathcal{L}(\theta) = \frac{1}{2} \mathbb{E}_{p_{\text{data}}} [\|\mathbf{s}_\theta(\mathbf{x}) - \nabla_{\mathbf{x}} \log p_{\text{data}}(\mathbf{x})\|_2^2] \quad (4.2)$$

During the test phase, a new sample is generated by Markov Chain Monte Carlo (MCMC) sampling, *e.g.*, Langevin Dynamics (LD), which is out of our interest since we focus on gradient field estimation.

However, the vanilla objective of score-matching in (4.2) is intractable, since $p_{\text{data}}(\mathbf{x})$ is unknown. To this end, the Denoising Score-Matching (DSM) (Vincent, 2011) proposes a tractable objective by pre-specifying a noise distribution $q_\sigma(\tilde{\mathbf{x}}|\mathbf{x})$, and train a score network to denoise the perturbed data samples, where $q_\sigma(\tilde{\mathbf{x}}|\mathbf{x}) = \mathcal{N}(\tilde{\mathbf{x}}; \mathbf{x}, \sigma^2 I)$ is a Gaussian kernel with tractable gradient in our cases:

$$\begin{aligned} \mathcal{L}(\theta) &= \mathbb{E}_{\substack{\tilde{\mathbf{x}} \sim q_\sigma(\tilde{\mathbf{x}}|\mathbf{x}), \\ \mathbf{x} \sim p_{\text{data}}(\mathbf{x})}} [\|\mathbf{s}_\theta(\tilde{\mathbf{x}}) - \nabla_{\tilde{\mathbf{x}}} \log q_\sigma(\tilde{\mathbf{x}}|\mathbf{x})\|_2^2] \\ &= \mathbb{E}_{\substack{\tilde{\mathbf{x}} \sim q_\sigma(\tilde{\mathbf{x}}|\mathbf{x}), \\ \mathbf{x} \sim p_{\text{data}}(\mathbf{x})}} \left[\left\| \mathbf{s}_\theta(\tilde{\mathbf{x}}) - \frac{1}{\sigma^2} (\mathbf{x} - \tilde{\mathbf{x}}) \right\|_2^2 \right] \end{aligned} \quad (4.3)$$

DSM guarantees that the optimal score network holds $\mathbf{s}_\theta^*(\mathbf{x}) = \nabla_{\mathbf{x}} \log p_{\text{data}}(\mathbf{x})$ for almost all \mathbf{x} .

In practice, we adopt an extension of DSM (Song et al., 2020) that estimates a *time-dependent score network* $\mathbf{s}_\theta(\mathbf{x}, t) : \mathbb{R}^{|\mathcal{X}|} \times \mathbb{R}^1 \rightarrow \mathbb{R}^{|\mathcal{X}|}$ to denoise the perturbed data from different noise levels simultaneously:

$$\begin{aligned} \mathcal{L}(\theta) &= \mathbb{E}_{t \sim \mathcal{U}(\epsilon, T)} \\ &\left\{ \mathbb{E}_{\substack{\tilde{\mathbf{x}} \sim q_{\sigma(t)}(\tilde{\mathbf{x}}|\mathbf{x}), \\ \mathbf{x} \sim p_{\text{data}}(\mathbf{x})}} \lambda(t) \left[\left\| \mathbf{s}_\theta(\tilde{\mathbf{x}}, t) - \frac{1}{\sigma^2(t)} (\mathbf{x} - \tilde{\mathbf{x}}) \right\|_2^2 \right] \right\} \end{aligned} \quad (4.4)$$

where $T, \epsilon, \lambda(t) = \sigma^2(t), \sigma(t) = \sigma_0^t$ and σ_0 are hyper-parameters. The optimal time-dependent score network holds $\mathbf{s}_\theta^*(\mathbf{x}, t) = \nabla_{\mathbf{x}} \log q_{\sigma(t)}(\mathbf{x})$ where $q_{\sigma(t)}(\mathbf{x})$ is the perturbed data distribution:

$$q_{\sigma(t)}(\tilde{\mathbf{x}}) = \int q_{\sigma(t)}(\tilde{\mathbf{x}}|\mathbf{x}) p_{\text{data}}(\mathbf{x}) d\mathbf{x} \quad (4.5)$$

Algorithm 2: Learning SocialGFs from Offline Examples

```
Generate  $N_e$  examples from the environment  $E(O_E, R_E)$  with the observation
function  $O_e$  and reward function  $R_e$ ;
while  $N < N_e$  do
    Learn gradient fields  $gf$  from example  $E_N$ ;
    if  $E_N \subseteq S^+$  then
        add  $gf$  to attractive representation set  $gf^+$ ;
        if  $R_E$  is sparse then
             $R_E \leftarrow R_E - \lambda|gf^+|$ ; // credit assignment
        else
            add  $gf$  to repulsive representation set  $gf^-$ ;
     $O_{GF} \leftarrow \{gf^+, gf^-\}$ ; // combine GFs
return the new environment representation:  $E(O_{GF}, R_E)$ ;
```

4.2 Method

In this section, we will introduce how to learn the gradient fields in the multi-agent environment and how to train an adaptive agent by using the gradient-based state representation.

4.2.1 Problem Formulation

In the context of a multi-agent environment denoted as $E(O_E, R_E)$, which is characterized by an observation function O_E and a reward function R_E , we initially express O_E as the concatenation of gf vectors: O_E is assigned as gf_1, gf_2, \dots, gf_n . Additionally, we adapt the reward function R_E to suit sparse reward environments by deducting the magnitude of the positive gf^+ : R_E is updated to $R_E - \lambda|gf_E^+|$. For adaptation, we simply substitute the existing gf representation, yielding the agent to respond accordingly to the new output. Further elaboration on these details follows.

For representing the O_E with gf , the initial step involves acquiring the example set S . For each example within this set, a score network \mathbf{s}_θ^* is trained by minimizing the expression specified in (4.4). This score network essentially embodies the learned gradient fields. By applying each \mathbf{s}_θ^* to the observation, a Gradient Field (GF) representation of the environment, denoted as O_{GF_E} , is obtained through the concatenation of all the individual outputs. This

representation can be further subdivided into attractive GF gf_E^+ and repulsive GF gf_E^- as $O_{GF_E} \leftarrow \{gf_E^+, gf_E^-\}$. Thereby transforming the environment representation to $E(O_{GF_E}, R_E)$.

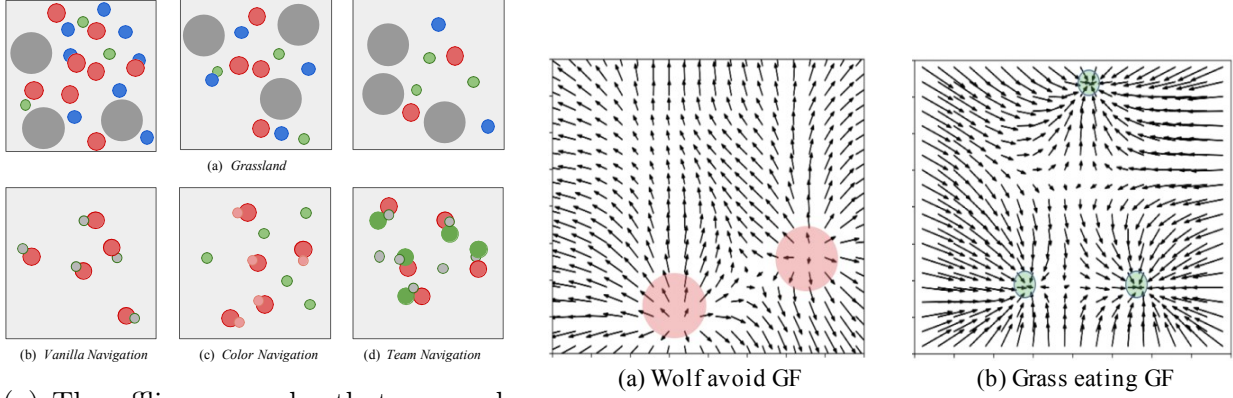
For scenarios with sparse rewards where exploration of the attractive example set is challenging, credit assignment is implemented. The magnitude of gf_E^+ is subtracted from these sparse attractive examples with a discount factor applied to the reward function R_E , resulting in $R_E \leftarrow R_E - \lambda|gf_E^+|$. Subsequently, Multi-Agent Reinforcement Learning (MARL) methods are applied to the new gf representation of the environment. An policy trained on this represented environment E is denoted as $\pi_\phi\{a_t|gf_E^+, gf_E^-\}$.

For adaptation, specifically transferring the policy $\pi_\phi\{a_t|gf_{E_1}^+, gf_{E_1}^-\}$ trained on environment E_1 to the environment E_2 , the GF representation $O_{GF_{E_2}}$ is obtained in the new environment, which is partitioned into attractive GF $gf_{E_2}^+$ and repulsive GF $gf_{E_2}^-$. Subsequently, the $gf_{E_1}^+, gf_{E_1}^-$ from E_1 are replaced with those from E_2 , resulting in the updated agent $\pi_\phi\{a_t|gf_{E_2}^+, gf_{E_2}^-\}$. gf could be shared among different tasks.

4.2.2 Offline Examples Collection

The creation of example set S involves the delineation of two fundamental categories: attractive examples $S^+ = \{s^+\} \subseteq S$ and repulsive examples $S^- = \{s^-\} \subseteq S$ tailored to each type of agent based on the task description, reward function, and expert knowledge. Subsequently, these example sets will serve as the foundation for training gradient fields in subsequent stages.

To elucidate, in the guidance of agents towards task completion, attractive examples pertain to goal states, whereas repulsive examples encompass scenarios where agents get punished. Examples can be obtained from triggered event. Let's consider a grassland game scenario, where the game dynamics stipulate that wolves prey on sheep, while sheep aim to evade wolves and consume grass. For the wolf, successful consumption of a sheep designates instances as attractive examples, whereas for the sheep, evading such situations constitutes repulsive examples. It is noteworthy that within each aspect, attractive S^+ or repulsive S^- , there may exist multiple types of example sets. Different example sets correspond to distinct



(a) The offline examples that we used for learning gradient fields. The different colors of the balls indicate that they belong to different classes.

(b) The visualization of learned Gradient fields for sheep in grassland game. Red and green circles indicate the wolves and the grass respectively.

goal states that agents aim to achieve or avoid. This process of generating example sets proves particularly efficacious in scenarios with sparse rewards, where achieving the designated goal state poses a formidable challenge and the diversity of examples is inherently limited.

In practice, we collect the examples based on the event triggered by the agents. Figure 4.1a shows several examples extracted from the environments. The different colors of the balls indicate that they belong to different classes. We show examples of the grassland game in Figure 4.1a(a). The red ball represents a wolf and the gray ball represents sheep. The left example is collected when the “sheep eaten” event is triggered. This is used as a repulsive example for sheep and a positive example for wolves. The middle example is collected when the “grass eaten” event is triggered. This is used as a positive example for sheep. The right side example is collected from frames in the game. It is used as an agent-object example of legal position since the agent is not colliding with the boundaries or obstacles. It is used as a positive example for both sheep and wolves. Figure 4.1a shows the three examples (b, c, d) of navigation games. They are collected when the “success” event is triggered.

4.2.3 Learning Gradient Fields

Utilizing the attractive example sets S^+ and repulsive example sets S^- as a foundation, we can proceed to train score networks to provide attractive gradient field gf^+ and repulsive gradient field gf^- , by adhering to the principles outlined in (4.4). The learned score networks

can subsequently be employed at each step to estimate the gradient from each type of example set. This gradient serves as an elevated representation of the environment, offering insights into the direction and distance of an agent to all kinds of example sets within S .

The gradient vectors obtained through this process encapsulate crucial information regarding an agent’s proximity to attractive or repulsive instances, thereby enabling an abstract understanding of the environment. This higher-level representation, derived from the attractive and repulsive gradient field networks, enhances the agent’s capacity to navigate and respond appropriately to the varying challenges posed by the attractive and repulsive examples within the environment. In Figure 4.1b, we show part of the gf applied in the grassland game. The arrows are generated by putting sheep in that location. The length of the arrows measures the magnitude of the gradients. Figure 4.1b(a) is the wolf avoid gf_{wolf_avoid} where the red circles are the positions of wolves. Figure 4.1b(b) is the grass-eaten gf_{grass_eaten} where the green circles indicate grass.

4.2.4 Gradient-Enhanced Rewards

In scenarios characterized by sparse rewards, where rewards are only allocated to a limited set of states, reward shaping becomes a common strategy. However, its applicability is problem-specific and demands meticulous engineering. In the context of gf generation from these sparse attractive examples, the magnitude $|gf^+|$ serves as an indicator of the current state’s distance from the distribution of attractive examples. Consequently, to mitigate sparsity, the absolute value of the GF, denoted as $|gf^+|$, is subtracted from the reward function with a discount factor, resulting in $R_e \leftarrow R_e - \lambda|gf^+|$.

The objective is to maximize the reward function, thereby minimizing the distance between the current state and the example state. This approach ensures that the agent, by seeking to maximize rewards, progressively narrows the gap between its current state and the target state, facilitating more effective learning and convergence toward the desired outcomes.

4.2.5 Learning Adaptive Policy

Adaptive ability refers to an agent’s capacity to alter its behavior in response to environmental shifts or novel tasks. The gradient field represents a form of representation adept at encoding the social rules and relationships within an environment. The introduction of adaptive ability in a Multi-Agent System can be accomplished through the replacement of gf representation. An agent $\pi_\phi\{a_t|gf_{E_1}^+, gf_{E_1}^-\}$ target for E_1 can transfer to E_2 by simply applying the gf_{E_2} representation: $\pi_\phi\{a_t|gf_{E_2}^+, gf_{E_2}^-\}$. A novel representation of the altered environment is introduced, compelling agents to transition to a new policy rooted in the updated gf . This transition prompts adjustments in the agents’ interactions with each other and the environment, fostering a dynamic and responsive behavior reflective of the modified environmental conditions. Also, gf can be reused among different tasks and agents.

4.3 Experiments

We tested our approach on four challenging environments, including a wolf-sheep game on grassland and three varieties of fully cooperative navigation games with sparse rewards. We found that our approach, SocialGFs, outperformed all other baselines in those four environments. We also tested the adaptive performance of SocialGFs across tasks and with different scales of agent populations and found that it performed well in those four environments.

4.3.1 Environments

All of the environments we used in our experiments were built on top of the particle-world environment (Mordatch and Abbeel, 2018), which is a continuous 2D world where agents can take actions in discrete timesteps.

Grassland: As is shown in Figure 4.2(a), there are two types of agents: sheep (blue) and wolves (red). The sheep need to collect grass pellets (green) and avoid wolves, while the

wolves need to eat sheep. The sheep move faster than the wolves. There are a fixed number of grass pellets (food for sheep) and large gray obstacles in the environment that can impede the movement of the agents. When a wolf collides with (eats) a sheep, the wolf is rewarded, and that (eaten) sheep will obtain a penalty. This will be marked as a "sheep eaten" event. When any sheep comes across a grass pellet, that sheep is rewarded and the grass will be collected and respawned in another random position, and this will be marked as a "grass eaten" event. The goal of the sheep is to collect as many grass pellets as possible and avoid being eaten by the wolves. The goal of the wolves is to eat as many sheep as possible.

Cooperative Navigation: In this environment, agents must cooperate through physical actions to reach a set of landmarks L . The agents will receive a reward and the event will be marked as "success" only when all the landmarks are occupied correctly. We designed three varieties of cooperative navigation for different difficulties:

- **Vanilla Navigation:** N cooperative agents aim to simultaneously occupy N landmarks without any conflicts, as shown in [Figure 4.2\(b\)](#).
- **Color Navigation:** There is an equal number of red and green agents, as well as an equal number of corresponding red and green landmarks. The objective of the game is for each agent to navigate towards and reach the landmark that matches their own color, as shown in [Figure 4.2\(c\)](#).
- **Team Navigation:** In the game shown in [Figure 4.2\(d\)](#), there are two teams of agents, one red and the other green. Both teams are composed of an equal number of agents. However, the number of agents exceeds the number of available landmarks. For a landmark to be deemed successfully occupied, it must be touched by at least one agent from each team.

4.3.2 Baselines and Evaluation Method

In the experiments, we compared the following approaches:

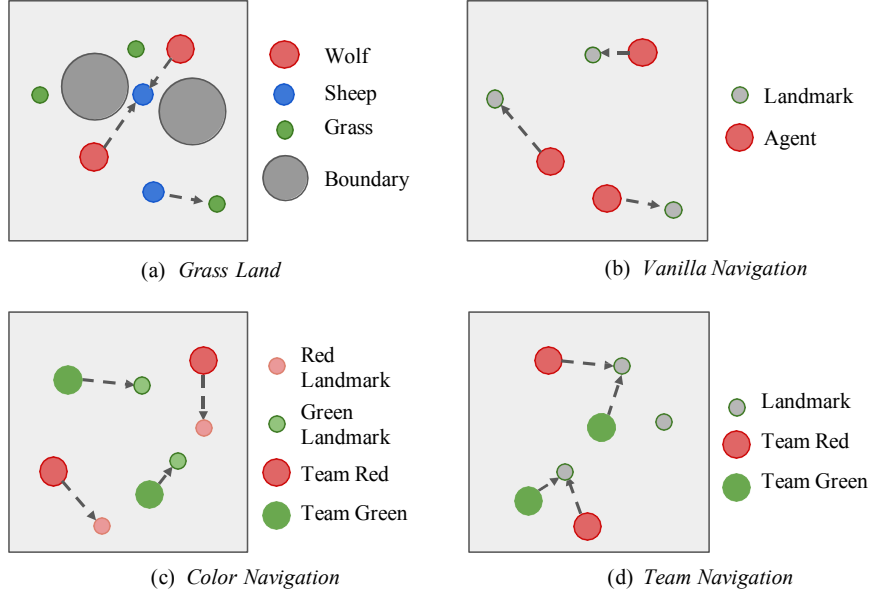


Figure 4.2: Four games used in the experiments. Figure (a) shows the grassland game where sheep work together to collect grass and avoid wolves, while the wolves work cooperatively to eat sheep. Figure (b)(c)(d) are three varieties of cooperative navigation games where agents need to cooperatively reach different landmarks.

- **Original (sparse) Reward:** this method trains the agents using event-based sparse reward and MAPPO (Yu et al., 2021) algorithm;
- **Reward Engineering:** this method trains the agents with human-shaped dense reward and MAPPO;
- **SocialGFs*:** Agents are trained with the original reward using the gradient fields learned from previously collected offline examples. The adaptive model, denoted by an asterisk (*), is trained from the SocialGFs sheep model at a scale of 4 – 4 within the Grassland game environment.
- **SocialGFs+ (for Cooperative Navigation):** Specifically designed for sparse reward scenarios, this method further incorporates gradient field representation with credit assignment.

For the grassland, we follow the evaluation method of Long et al. (2020) by competing agents trained from different methods. Specifically, we let sheep trained by different approaches

compete with wolves trained from SocialGFs to get the average reward as the evaluation of sheep. Similarly, we compete for wolves with SocialGFs sheep to evaluate different wolves. For better visualization, we use normalized reward from 0.1-1 to show the difference between methods. We also do a cross-match of every wolf and sheep trained under a scale of 4-4. In cooperative navigation, we calculate the final step success rate as evaluation metrics.

4.3.3 Quantitative Results in *Grassland*

We denote a game with N_W wolves and N_S sheep by “scale N_W - N_S ”. Both N_W and N_S can choose from $\{2, 4, 8\}$ so totally there are 9 scales. The SocialGFs sheep’s GF-representation is: $\{gf_{grass_eaten}^+, gf_{boundary_avoid}^+, gf_{wolf_avoid}^-\}$, and the wolf’s GF-representation is: $\{gf_{sheep_chasing}^+, gf_{boundary_avoid}^+\}$. During MARL training, every method is trained with 2×10^6 episodes on all scales. The SocialGFs* we choose are sheep models trained from scale 4 – 4. To adapt the sheep, there is nothing to change since the gf representation is the same. For wolves, we replace $gf_{grass_eaten}^+$ with the wolves $gf_{sheep_chasing}^+$ and remove the $gf_{grass_eaten}^+$.

Main Results: The results are shown in [Figure 4.3](#). From the graph, we can see that SocialGFs* and SocialGFs dominate most of the scales. Overall, there is little difference between the original reward and Reward Engineering results. For adaptive ability, SocialGFs* adapt perfectly as sheep for all the scales, surprisingly, they are even the best wolves for a few scales. We also do cross matches between all the wolves and sheep trained from different methods on scale of 4-4. The result is shown in [Table 4.1](#). The red numbers are the reward of wolves and the green numbers are the result of sheep. When we fix the same group of sheep or wolves, the SocialGFs and SocialGFs* method always perform better compared to the other methods.

4.3.4 Quantitative Results in Cooperative Navigation

In the vanilla navigation game, we have 2-5 agents and the number of landmarks is the same as the agent. For the color navigation game, there are two teams of the same equal agents,

Table 4.1: Cross-validation results in the Grassland game with 4 sheep and 4 wolves

Wolf \ Sheep	Original Reward	Reward Engineering	SocialGFs	<i>SocialGFs*</i>
Original Reward	0.904	0.905	2.351	1.07
Reward Engineering	-0.62	-0.638	-1.819	-0.734
SocialGFs	1.345	0.662	4.804	2.577
<i>SocialGFs*</i>	-1	-0.316	-3.86	-1.894
Original Reward	0.169	0.266	0.695	0.855
Reward Engineering	0.323	0.209	0	-0.178
SocialGFs	0.169	0.273	0.707	0.877
<i>SocialGFs*</i>	0.315	0.206	-0.019	-0.192



Figure 4.3: The quantitative results in Grassland. In the left two figures, we scale the wolf and sheep reward to 0.1-1. In the rightmost figure, we list the grass eaten amount for sheep in 100 time steps to measure the grass collection ability.

Table 4.2: Success rate of three cooperative navigation games with different populations

# of agents	Vanilla Navigation				Color Navigation				Team Navigation			
	2	3	4	5	2	3	4	5	2	3	4	5
Original Reward	0.01	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Reward Engineering	0.19	0.0	0.0	0.0	0.05	0.0	0.0	0.0	0.004	0.001	0.0	0.0
SocialGFs	0.998	0.948	0.88	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
<i>SocialGFs</i> ⁺	0.992	0.971	0.883	0.751	0.484	0.428	0.348	0.240	0.359	0.265	0.203	0.106
<i>SocialGFs</i> [*]	0.771	0.639	0.571	0.400	0.417	0.302	0.243	0.142	0.121	0.098	0.051	0.038

each with 2-5 agents and the landmark number is equal to the number of agents. For the team navigation game, there are two teams of equal agents. The number of agents in each team and landmarks are (2,3), (3,5), (4,6), and (5,7). The SocialGFs agent’s GF-representation is $gf_{navigation}^+$. The SocialGFs* we choose are sheep models trained in a grassland game of scale 4-4. We replaced the representation from $gf_{grass_eaten}^+$, $gf_{boundary_avoid}^+$, $gf_{wolf_avoid}^-$ with $gf_{navigation}^+$.

We list the success rate of different navigation games in [Table 4.2](#). Both the original reward method and the Reward Engineering method failed in all scenarios except the smallest scale of the vanilla navigation game. SocialGFs are only able to work in simpler scenarios in vanilla navigation games that have less than 5 agents. SocialGFs+ outperforms all the other methods with a large gap in all the environments. SocialGF* agents are also managed to succeed in all the cooperative navigation games which shows the strong adaptability of SocialGFs.

4.3.5 Qualitative Results

In the grassland game, as the number of wolves increases, it becomes more difficult for sheep to develop grass-collecting abilities. This is because the wolves are able to more easily catch and eat the sheep, which reduces the amount of time that the sheep have to collect grass.

We take a few example screenshots of two representative grassland matches between SocialGFs agents and original reward agents from the experiment rendering result in [Figure 4.4a](#). The small gray balls represent grass. The red balls represent wolf and the blue dot represents sheep. The upper graph of [Figure 4.4a](#) shows the results of a simulation with 4 original reward wolves and 2 SocialGFs sheep. The green box shows that one of the wolves is not chasing any sheep and the other wolves are not able to catch the sheep. The red box shows a sheep can collect grass and avoid the wolves at the same time. The lower graph shows the results of a simulation with 4 SocialGFs* wolves and 2 original reward sheep. The red box shows that the wolves can successfully corner and eat the sheep.

We also visualize two representative team navigation games in [Figure 4.4b](#). Among the

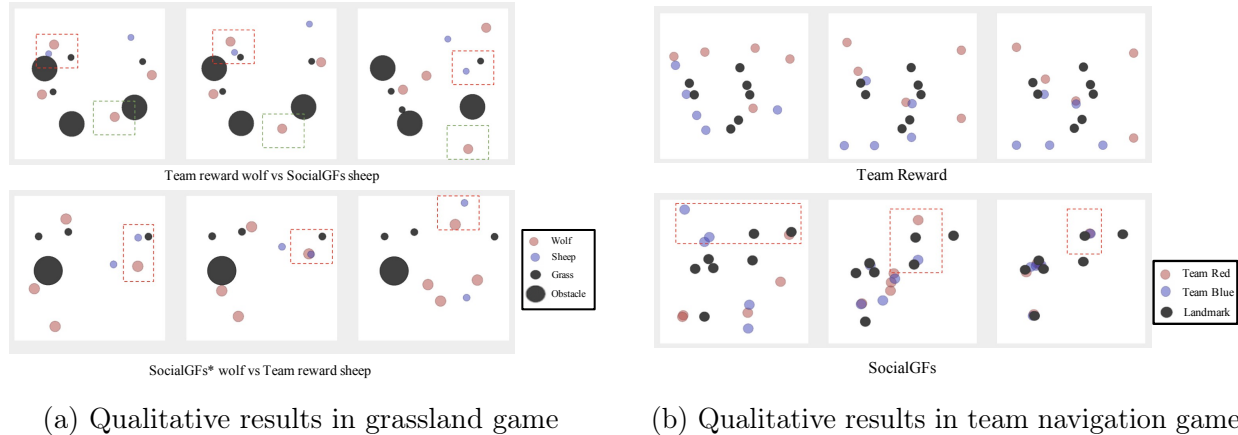


Figure 4.4: Figure (a) is the qualitative results in grassland game. We visualize two representative grassland matches between SocialGFs agents and original reward agents. The upper graph shows the results of a simulation with 4 original reward wolves and 2 SocialGF sheep. The green box shows that one of the wolves is not chasing any sheep and the other wolves are not able to catch the sheep. The red box shows a sheep can collect grass and avoid the wolves at the same time. The lower graph shows the results of a simulation with 4 SocialGFs* wolves and 2 original reward sheep. The red box shows that the wolves can successfully corner and eat the sheep. Figure (b) is the qualitative results in team navigation game. We visualize two representative team navigation games from the original reward method and SocialGFs method. The upper graph shows that the original reward method is unable to learn to play this game. The bottom figure shows the SocialGFs agents successfully cooperate together to occupy 5 landmarks

cooperative navigation games, the team navigation game is the most difficult. The upper graph of Figure 4.4b shows that the original reward method is unable to learn to play this game. The bottom figure shows the SocialGFs agents successfully cooperating to occupy 5 landmarks. In the red box, we show two agents from different teams coming all the way together to occupy the landmark in between.

These results suggest that agents trained with SocialGFs have adapted successfully in new environments and SocialGFs is a more effective method for training agents to play the grassland game and cooperative navigation games than the original reward method.

CHAPTER 5

Inverse Attention Agents for Multi-Agent Systems

5.1 Method

In this section, we introduce the inverse-attention agent, which acts based on its attention weights of goals and updates the weights based on the inferred goal weights of other agents. The agent is trained in three phases. The first phase involves applying the self-attention mechanism to the policy function (Vaswani et al., 2017) so that an agent acts based on the weights of the attentions. The second phase is to infer the attention of other agents of the same type using the inverse attention network. By placing itself in the position of the other agents, the agent infers their attention weights, gaining insights into their attention and behavior. The final phase involves training the inverse attention agent. The agent uses the inferred attention weights from the previous phase to update its own attention weights, consequently leading to changes in its final actions. The following subsections present the details of our algorithm. The overall pipeline is shown in Figure 1.3. The architecture of the inverse attention agent policy network is shown in Figure 5.1.

5.1.1 Self-Attention Structure

We incorporate a self-attention mechanism into the policy network in order to explicitly model the agent’s mental states through attention weights assigned to different goals. These mental states are crucial, as they influence the agent’s final action, and altering the attention weights will lead to changes in the action. Additionally, during this phase, we prepare the data required for training the inverse attention network. The agent trained with this structure is referred to as the Self-Att agent.

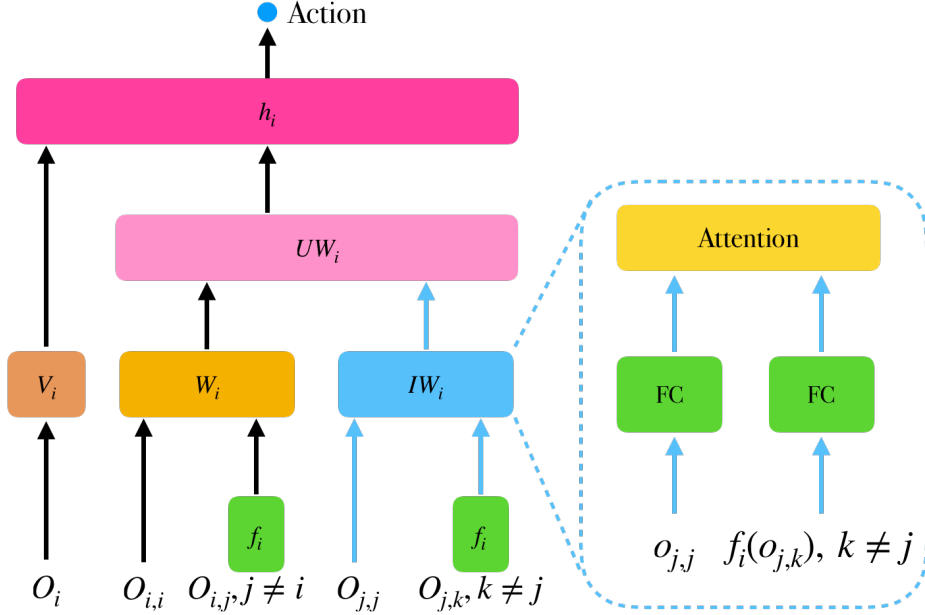


Figure 5.1: Network architecture of the inverse attention agent. For agent i , W_i is the observation embedding function which takes in the observation and outputs initial attention weights. IW_i is the inverse attention network which takes in the action and observation of the other agents and outputs the inferred attention weights. The UW_i takes consideration of self initial weights and inferred weights from others and update a_i 's attention weights. The h_i function outputs the final action based on the updated weights.

Self-Attention Network: Instead of using an MLP for the policy network, we adopt a structure similar to that proposed by Long et al. (2020), which utilizes attention embeddings of the goal for the decentralized policy network. This can be represented as

$$\pi_i(\mathbf{o}_i) = h_i(W_i(f_i(o_i)), V_i(f_i(o_i))), \quad o_i = o_{i,1}, o_{i,2}, \dots, o_{i,N}, \quad (5.1)$$

where o_i is the observation of agent i , which can be decomposed into a combination of N goals within the environment, f_i is a two-layer fully connected network that embeds the N goals, $W_i(o_i)$ is the weight encoder that outputs the attention weights of the goals, whose details we will discuss later, and V_i is the value function implemented as a two layer fully connected layer. The function h_i multiplies the attention weights with all the goal information to get the weighted goals $goal_{\text{weighted}} = \sum w_{i,j} V_i(f_i(o_{i,j}))$ and then passed into a two-layer fully connected neural network to output the final action.

Attention Weight Function: We define the attention weight embedding function as

$$W_i(f_i(o_i)) = \text{attention}(f_i(o_{i,i}), f_i(o_{i,j})), \quad \forall j \neq i. \quad (5.2)$$

Self-attention is applied to the agent’s own information along with all the goals and generates the output attention weights.

Attention Inference Dataset Construction: We collect the data pairs $(\{w_{i,1}, w_{i,2}, \dots, w_{i,N}\}, o_i)$ during the training of the self-attention network. Once the policy has converged, we retain only the most recent 1/10 of these data pairs in the dataset D for training our inverse attention network.

5.1.2 Attention Inference

We train the inverse attention network using the attention inference dataset D . This network operates inversely to the self-attention network: It outputs the predicted attention weights based on the given goals and past actions. An agent then applies the inverse attention network to other agents of the same type to infer their attention weights. This allows the current agent to infer the attentions of other agents without knowing the ground truth values of their attention weights.

Inverse Attention Network: We propose the inverse attention network, designed to infer the attention of agents of the same type from their perspective. This function has the following structure:

$$\{w_{i,1}, w_{i,2}, \dots, w_{i,N}\} = IW_i(o_i) = \text{attention}(o_{i,i}, f_i(o_{i,j})), \quad \forall j \neq i. \quad (5.3)$$

Similar to the attention weight function, the IW_i function predicts the attention weight by applying self-attention between the embedding of self-information $o_{i,i}$ and the embedding of other goals $f_i(O_{i,j})$. Using the dataset D , which is agent’s own data collected during the training process of Phase 1, we train the IW_i network by minimizing the following loss

function:

$$\text{Loss} = L2(\{w_{i,1}, w_{i,2}, \dots, w_{i,N}\}, IW_i(o_i)). \quad (5.4)$$

Inference of Others: To estimate the attention weights of other agents, we position the current agent as if it were one of the other agents. In a fully observable environment, we gather the observations and previous actions of other agents of the same type. Then, we utilize the inverse attention network to infer their attention towards different goals. The inference attention weight for a_j is

$$\{\tilde{w}_{j,1}, \tilde{w}_{j,2}, \dots, \tilde{w}_{j,N}\} = IW_i(o_j). \quad (5.5)$$

5.1.3 Inverse Attention Agent

We construct the inverse attention agent (Inverse-Att) by concatenating its original attention weights with the inferred weights. This concatenated vector is then passed through a fully connected layer, UW_i , to update the attention weights. This mechanism enables an inverse attention agent to adapt its attention weights and consequently adjust its actions based on the attentions of other agents. The process is represented by the following equation:

$$\{\tilde{w}_{i,1}, \tilde{w}_{i,2}, \dots, \tilde{w}_{i,N}\} = UW_i(W_i(f_i(o_i)), IW_i(o_j)), \quad \forall j \neq i \text{ and } a_j \text{ is the same type as } a_i, \quad (5.6)$$

where o_j is the observation from the other agent in the previous step, $IW_i(o_j)$ outputs the inverse attention estimation of agent j from agent i , and UW_i is the weight updating model, which concatenates the estimated attention weights from other agents of the same type with the original weights $W_i(f_i(x))$. The concatenated weights are then passed through a one-layer fully connected network to update the attention weights to \tilde{w} . Given that the attention weight embedding is a high-level representation, only a shallow network is required.

For fast convergence, the UW_i are initialized to 1 when connected to the original attention weight and 0 when connected to the other agents' attention weights. This initialization guarantees that the initial output aligns with that of the Self-attention agent, maintaining

Algorithm 3: Inverse attention algorithm

Data: environment $E(N, \{A_i\})$ with N agents

Result: an inverse attention agent

Use (5.1) as the policy function for the target training agent a_i ; // self-att framework

while a_i 's policy network is not converged **do**

 MARL training on a_i ;
 Collect $\{w, o_i\}$ pair and store it to the dataset D ; // training dataset for IW

Keep the 1/10 latest $\{w, o_i\}$ in D and discard the rest;

Training inverse attention network IW using $\{w, o_i\}$ pair from the trimmed D by following (5.4);

Replace the policy network of a_i with (5.7) using trained IW ; // inverse-att framework

while a_i 's policy network is not converged **do**

 MARL training on a_i ;

Return: a_i is the inverse attention agent;

unchanged attention weights. Commencing the optimization process from this point enhances effectiveness.

The complete policy network module for the inverse attention agent is outlined as follows:

$$\pi_i(\mathbf{o}_i) = h_i(UW_i(W_i(f_i(o_i)), IW_i(o_j)), V_i(f_i(x))), \quad \forall j \neq i \text{ and } a_j \text{ is the same type as } a_i. \quad (5.7)$$

algorithm 3 is the inverse attention algorithm.

5.2 Experiments

We adopted MAPPO (Yu et al., 2022) as our MARL training scheme. We evaluated the adaptive performance of the policies in collaboration with human participants across all these tasks and found that the inverse attention agent performed well in these environments. Additionally, we compared our agents' behavior with human behavior. Our results indicate that the inverse attention agent is more adaptable to a wider variety of agents.

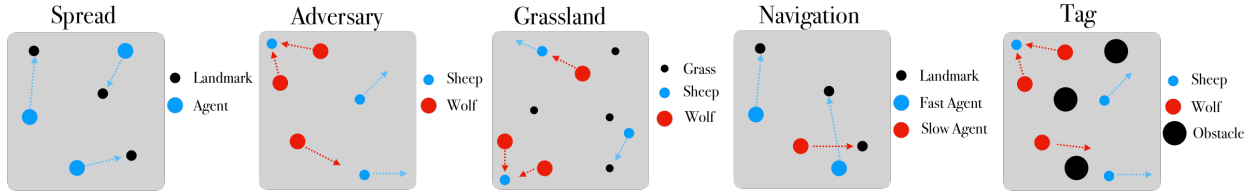


Figure 5.2: Environment visualization of the spread, adversary and grassland game

5.2.1 Environment

All of the environments used in our experiment were built on top of MPE (Mordatch and Abbeel, 2017; Lowe et al., 2017). Agents are depicted as particles inhabiting a continuous two-dimensional space, where they navigate, interact with one another, and with static landmarks, all within discrete time intervals. The key distinction lies in our application of the GF function atop raw observations, resulting in a gf representation structured as $\{gf_1, gf_2, \dots, gf_N, gf_{\text{wall}}\}$. This representation comprises the gradient fields of N entities and the gradient field of the boundaries, serving as the goals for the agents. We assessed the efficacy of our algorithm across five challenging environments: Spread, Adversary, Grassland, Navigation, and Tag on MPE. Visualizations of these environments are depicted in Figure 5.2.

Spread: This is a fully cooperative game. In this scenario, N agents must spread to cover N landmarks. At each discrete time step, an agent earns 5 points if it occupies a landmark, and 0 points otherwise. The optimal state is each agent occupies a distinct landmark. The default setting is $N = 3$.

Adversary: This is a fully competitive game that involves two types of agents: N sheep and N wolves. Wolves catch sheep, while sheep avoid capture. Sheep are faster than wolves. Each wolf earns 5 points for catching a sheep, while each caught sheep loses 5 points. The scale of this game is denoted as $N - N$. The default setting is $N = 3$.

Grassland: This is a mixed game. Entities include N sheep, N wolves, and 4 grass. Sheep aim to collect as much grass as possible while evading wolves. Wolves seek to catch as many sheep as they can. Each sheep earns 3 points for each collected grass but loses 5 points

for each capture. Wolves earn 5 points for each caught sheep. Grass re-spawns at random positions after being consumed by sheep. The scale of this game is denoted as $N - N$. The default setting is $N = 3$.

Navigation: This is a fully cooperative game with a group reward system. There are two fast agents and one slow agent, and the goal is for the agents to navigate to three different landmarks as quickly as possible. The team earns 5 points for each landmark occupied, shared equally among all agents. The fast agents must accommodate the slow agent to help reach the more distant landmarks.

Tag: This is a mixed game involving 3 wolves and 3 sheep, with group rewards. The objective for the wolf team is to catch the sheep while avoiding three obstacles during the chase. When any wolf catches a sheep, all wolves received 5 points and all sheep lose 5 points.

5.2.2 Baseline and Evaluation Method

In the experiments, we compared agents trained with the following methods: (1) **MAPPO**: We follow the same method as (Yu et al., 2022). (2) **IPPO**: We follow the same method as (Schulman et al., 2017) (3) **MAA2C**: It extends the existing on-policy actor-critic algorithm A2C (Mnih et al., 2016) by applying centralised critics conditioned on the state of the environment. (4) **ToM2C***: A modified ToM method adapted from (Wang et al., 2022), where agents directly predict other agents goals without communication. (5) **Self-Att**: We adopt the self-attention structure proposed in Section 5.1.1. (6) **Inverse-Att**: Our proposed attention inference agent described in Section 5.1.3. All the baseline methods are trained for the same amount of accumulative episodes as an Inverse-Att agent required. All agents trained in these methods do not see agents of other methods until evaluation time.

We assessed the performance of our algorithm through *mix-and-match*. For each method, we select three different seeds to form three distinct groups. We formed an agent pool by putting agents trained using all the evaluated methods. During the cross-competition period, agents were randomly sampled from this pool to form a unique composition. The cross

Table 5.1: Full result

	MAPPO	IPPO	MAA2C	ToM2C*	Self-Att	Inverse-Att
Spread	31.82±1.21	0.55±0.01	48.46±0.47	49.04±3.77	283.89±6.23	404.14±5.40
Adversary Sheep	-113.97±1.74	-121.03±4.17	-96.54±3.26	-86.15±2.80	-23.15±0.83	-16.91±1.84
Adversary Wolf	48.76±1.86	25.18±1.40	109.14±0.69	61.58±1.31	107.93±2.26	110.15±3.74
Grassland Sheep	-84.50±4.20	-86.54±0.32	-70.79±2.41	-61.60±1.98	-10.02±5.19	28.59±0.93
Grassland Wolf	27.86±0.79	22.60±1.38	74.28±1.27	41.11±1.59	93.68±1.61	101.21±2.84
Navigation	251.14±4.93	230.10±0.89	279.87±7.05	249.41±5.29	328.24±7.76	497.96±10.75
Tag Sheep	-93.51±2.13	-111.79±3.03	-77.81±2.03	-56.69±1.02	-11.74±0.65	-5.99±0.44
Tag Wolf	42.17±3.46	26.77±0.82	59.41±1.10	52.99±1.17	67.60±1.00	109.81±1.36

competitions were conducted over 1000 episodes with each episode consisting of 200 steps. During each episode, we recorded the agents’ rewards according to their respective methods and calculated the average reward for performance comparison.

5.2.3 Quantitative Results

In this section, we delve into the quantitative results across all five games. We conducted training for MAPPO, IPPO, MAA2C, ToM2C* and Self-Att agents over 40 million steps across all scales. For training the Inverse-Att agent, Phase 1 encompassed 20 million steps, followed by another 20 million steps for Phase 3. Phase 2 training involved offline learning, obviating the need for interaction with the environments. Subsequently, evaluations were carried out over 2×10^6 steps of mix-and-match for all five games.

We tested all the methods across all environments, with the full results presented in Table 5.1. The Inverse-Att method demonstrates significant improvement over all the other methods across the five games in cooperative, competitive, and mixed settings, as well as with both individual and team rewards. This highlights the superior adaptability and generality of Inverse-Att when dealing with unseen agents. Self-Att ranks second, while the remaining baselines perform similarly and rank below both Inverse-Att and Self-Att. Due to the similar performance of MAPPO, IPPO, MAA2C, and ToM2C*, and computational limitations, in the following sections we will focus on MAPPO, Self-Att, and Inverse-Att in the Spread, Adversary, and Grassland games.

Table 5.2: Spread result

	MAPPO	Self-Att	Inverse-Att
2	51.14 \pm 1.15	288.75 \pm 2.80	363.02 \pm 1.99
3	31.10 \pm 0.93	258.81 \pm 5.35	379.95 \pm 1.99
4	19.59 \pm 0.22	253.48 \pm 1.13	269.12 \pm 0.86

Table 5.3: Adversary result

	MAPPO WOLF	Self-Att WOLF	Inverse-Att WOLF	MAPPO SHEEP	Self-Att SHEEP	Inverse-Att SHEEP
2-2	14.30 \pm 0.61	39.75 \pm 1.88	42.45 \pm 1.50	-64.23 \pm 1.70	-18.49 \pm 0.48	-13.82 \pm 1.10
3-3	29.30 \pm 1.42	71.95 \pm 2.38	75.44 \pm 1.02	-132.75 \pm 2.27	-30.79 \pm 1.82	-12.61 \pm 0.24
4-4	32.93 \pm 1.21	66.75 \pm 1.62	67.15 \pm 0.97	-134.31 \pm 3.01	-16.82 \pm 0.28	-15.59 \pm 0.65

5.2.4 Impact of Different Population Scales

We tested the scalability of Inverse-Att by comparing with MAPPO and Self-Att in the Spread, Adversary, and Grassland games in the MPE environment.

In the Spread game, evaluations were conducted across three scales: 2, 3, and 4 agents. The results are detailed in Table 5.2. Meanwhile, in the Adversary and Grassland games, evaluations were conducted across scales of 2-2, 3-3, and 4-4 for both sheep and wolves. The outcomes are presented in Table 5.3 and Table 5.4, respectively.

Across all three games, notable enhancements were evident when comparing the outcomes of MAPPO and Self-Att, underscoring the efficacy of the self-attention network. The Inverse-Att method exhibited superior performance across the tested environments, particularly in cooperative-related games such as Spread and Grassland. This superiority likely stems from the attention inference being exclusively applied to agents of the same type (teammates), a factor more critical in cooperative settings than in fully competitive games.

5.2.5 Human Experiments

To further assess the adaptive capabilities of the Inverse-Att agents, we conducted human experiments to evaluate their performance in collaboration with human players. Five partici-

Table 5.4: Grassland result

	MAPPO WOLF	Self-Att WOLF	Inverse-Att WOLF	MAPPO SHEEP	Self-Att SHEEP	Inverse-Att SHEEP
2-2	20.80 \pm 0.23	36.85 \pm 2.18	56.69 \pm 1.78	-70.17 \pm 1.71	10.25 \pm 2.33	33.66 \pm 1.23
3-3	22.06 \pm 0.78	65.50 \pm 1.96	70.28 \pm 0.78	-99.73 \pm 1.19	-21.52 \pm 1.32	25.45 \pm 0.13
4-4	35.38 \pm 0.54	46.35 \pm 0.79	81.67 \pm 1.51	-138.71 \pm 2.68	14.14 \pm 0.71	36.16 \pm 0.14

Table 5.5: Results of the human experiments

	MAPPO	Self-Att	Inverse-Att	Human
Spread	16.8 \pm 13.81	272.0 \pm 30.41	332.3 \pm 17.13	115.86 \pm 55.89
Adversary (Wolf)	81.5 \pm 38.42	197.4 \pm 25.85	286.9 \pm 22.21	188.4 \pm 74.87
Adversary (Sheep)	-48.8 \pm 24.06	-8.0 \pm 3.46	-0.8 \pm 1.16	-15.33 \pm 10.47
Grassland (Wolf)	49.3 \pm 20.38	197.9 \pm 12.76	185.7 \pm 30.45	132.86 \pm 51.84
Grassland (Sheep)	-20.75 \pm 13.95	9.8 \pm 8.66	20.34 \pm 2.33	-30.52 \pm 32.15

pants engaged in Spread, Grassland and Adversary games across the following scales: Spread (3 agents), Adversary (3-3), and Grassland (3-3), assuming five distinct roles (agent for Spread; sheep and wolf for Adversary and Grassland). In each role, participants cooperated with teammate agents of the same type trained using each of the three methods (MAPPO, Self-Att, Inverse-Att) over five episodes, while opponent agents remained consistently MAPPO agents. Agents were sampled by type in each environment and then fixed to ensure uniformity across participant groups. Each episode comprised 100 steps, with rewards recorded based on the respective training methods. The results are summarized in Table 5.5.

It is noteworthy that both Self-Att and Inverse-Att agents outperformed human participants in the tested environments. Across most roles within the environments, Inverse-Att agents exhibited greater adaptability to human players and demonstrated greater stability

Table 5.6: Group reward in spread game with multi Inverse-Att agents

#N Inv-Att Scale	0	1	2	3	4
2	86.44 \pm 0.36	472.92 \pm 4.93	593.57 \pm 7.02	-	-
3	148.53 \pm 1.64	635.40 \pm 7.01	812.77 \pm 5.12	1070.28 \pm 8.75	-
4	109.26 \pm 1.60	494.59 \pm 11.30	701.60 \pm 13.95	818.24 \pm 8.20	1140.23 \pm 4.67

compared to the other methods, with exceptions observed in the role of wolf in the Grassland environment.

5.2.6 Impact of Multiple Inverse-Att Agents

In multi-agent systems, incorporating Theory of Mind (ToM) can introduce intricate dynamic interactions. This complexity is amplified when the inference is reciprocal, potentially creating a cognitive loop that exacerbates uncertainty and leads to unpredictable or suboptimal decision-making outcomes. Our objective is to explore the impact of increasing the number of Inverse-Att agents on emergent behavior patterns.

We initiated the investigation by replacing randomly sampled MAPPO agents, trained with different seeds, with Inverse-Att agents gradually. Evaluations were conducted over 2×10^6 steps per group in the Spread game, across three different scales: 2, 3, and 4 agents. The total rewards earned by the teams are summarized in [Table 5.6](#). We observe a nonlinear marginal return pattern as more Inverse-Att agents are introduced into the game at all three scales. This observation underscores the effectiveness of our Inverse-Att agent in cooperating effectively with other attention-aware agents.

5.2.7 Inverse Attention Network Prediction Accuracy

We collected 2×10^6 steps of weights-observation pair from one Self-Att agent per environment, considered as the attention ground truth of those agents. We then inputted the observation into our inverse attention network and compared the predicted weights with the ground truth across the spread, adversarial, and grassland environments at scales {spread: 3, adversarial: 3-3 and grassland: 3-3}. We evaluated the effectiveness of our inverse network in accurately predicting weights based on their rank. For example, if the inverse network identifies gf_{wall} as having the highest weight, this prediction is considered accurate if the self-attention network also determines that the weight of gf_{wall} is the highest. The results are summarized in [Figure 5.3](#). The prediction of the most significant attention reaches nearly 100% accuracy, demonstrating that the inverse network can accurately predict the attentions of other agents,

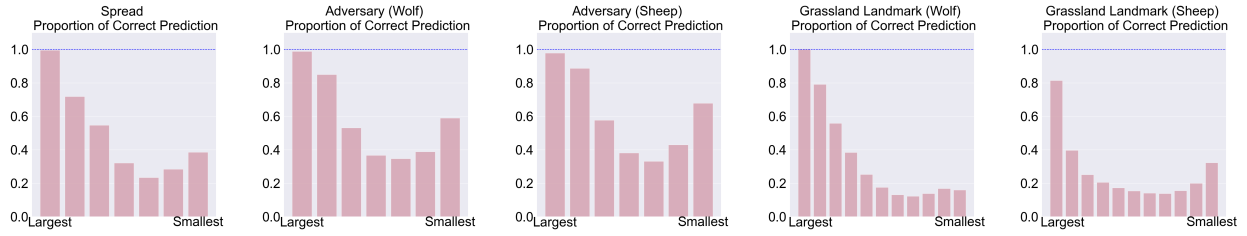


Figure 5.3: We evaluated the prediction accuracy of the inverse attention network across five roles in the spread, adversary, and grassland environments under the scale of {spread: 3, adversarial: 3 – 3 and grassland: 3 – 3}. In each bar graph, from left to right, we display the prediction accuracy from the most attended goal to the least attended goal. The results demonstrate that the inverse network can accurately predict the attentions of other agents, particularly for the top two attentions of interest.

especially for the top two attentions of interest.

CHAPTER 6

A Benchmark for Multi-Modal Multi-Agent Systems in Minecraft

6.1 The TeamCraft Benchmark

6.1.1 Problem Formulation

Assume that an embodied multi-agent system comprised of N agents needs to complete a complex task involving navigation and object manipulation. The task is specified in a multi-modal prompt $x_L = \{x_l\}_{l=1}^L$, which is a sequence of interleaved language and image tokens with length L . At time step t , each agent receives partial observation $o_n^t \in O$ from the full observation space O . To complete the task, each agent can choose to perform a high level action $a_t \in A$ from the full set of action A . The action can be further decomposed into a sequence of low level control signals.

6.1.2 Simulation Environment

TeamCraft utilizes Minecraft as its foundational simulation environment, offering a complex, open-world setting for multi-agent interactions. With a Gym-like environment, it facilitates the execution of intricate multi-agent commands via self-explanatory skills. [Figure 1.4](#) illustrates the platform architecture. High level skills from the model can be translated into low level control signals via nested API calls through Mineflayer¹. After execution, visual observation of each agent are rendered and provided as input to the model.

¹<https://github.com/PrismarineJS/mineflayer>









	Building	Clearing	Farming	Smelting
System Prompt	 <p>Three bots need to build a building on the platform. Write actions for <i>bot1, bot2, bot3</i> based on given observation.</p>	 <p>Three bots need to break everything on the platform. Write actions for <i>bot1, bot2, bot3</i> based on given observation.</p>	 <p>Two bots need to grow on the platform. The goal is to get 4 carrot. Write actions for <i>bot1, bot2</i> based on given observation.</p>	 <p>Three bots need to craft 3 smooth sandstone. To craft, I need obtain 'sandstone' with a pickaxe. Write actions for <i>bot1, bot2, bot3</i> ...</p>
Observation	 <p>bot1 has 5 bricks, 3 iron_ore, bot2 has 2 sea_lantern, bot3 has 1 brick...</p>	 <p>bot1 has a stone_axe, bot2 has a stone_pickaxe, bot3 has a stone_sword...</p>	 <p>bot1 has 3 carrot, 1 potato, bot2 has 3 carrot, 2 beetroot...</p>	 <p>bot1 has 1 beef, bot2 has 1 iron_sword, bot3 has 1 iron_shovel...</p>
Action	<pre>placeItem(bot1, 'bricks', (-1,0,-1)) placeItem(bot2, 'oak_planks', (0,0,0)) placeItem(bot3, 'iron_ore', (0,0,-1))</pre>	<pre>mineBlock(bot1, (-1,0,1)) mineBlock(bot2, (-2,0,0)) mineBlock(bot3, (-1,1,1))</pre>	<pre>farmWork(bot1, (1,-1,1), sow, 'carrot') farmWork(bot2, (-1,-1,-2), sow, 'carrot')</pre>	<pre>putItem(bot1, 'sandstone', (0,0,-1)), obtainBlock(bot2, (2,0,0)), obtainBlock(bot3, (1,0,-3))</pre>

Figure 6.1: Multi-modal prompts are provided for all tasks. The system prompt includes both the three orthographic views and specific language instructions. Observations consist of first-person views from different agents, along with agent-specific information.

Multi-Modal Prompts: In our work, the multi-modal prompt x_L consisting of a language instruction interleaved with a set of orthographic projection images (i.e. top, left, front views) for task specification. Depending on the specific task, the images can specify either the initial states, intermediate states or the goal states.

Observation and Actions: To mimic real world settings of embodied visual agent teaming, we use first-person view RGB image and inventory information as the observation o_n . The action space A involves high-level self-explanatory skills such as *obtainBlock* to obtain a block and *farmWork* to farm a crop. Most actions take three input parameters, including (1) agent name such as *bot1*, as the action-executing entity, (2) item name such as *dirt*, and (3) a 3D vector indicating the position of the target. There are 8 types of actions in total. A complete list of actions are described in the supplementary.

6.1.3 Task Design

TeamCraft introduces a variety of complex and interactive tasks that challenge the agents' capabilities in planning, coordination, and execution within a collaborative and dynamic environment. Each task is designed to test different facets of MA interaction, including role distribution, real-time decision-making, and adaptability to changing environments. Task examples are shown in Figure 6.2 and the corresponding prompt examples are shown in

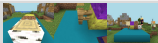


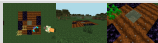
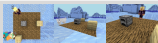
	Building	Clearing	Farming	Farming	Smelting	Smelting
Scenes	village	snow_mountain	village	swamp	ice_on_water	desert_village
Base	cyan_concrete	gold_block	hay_block	obsidian	oak_wood	glass
Goal	Build 1x2x4 building	Clean 3D building	Potato *3	wheat *4	cooked_mutton *1	smooth_quartz *2
Object	[dirt, wool, fence, sandstone, sponge]	[grass_block, dirt, birch_log, bookshelf,]	-	-	[birch_planks, sheep]	[oak_planks, quartz_block]
Agent	3	3	2	2	3	2
Inventory	[dirt, wool, fence, sponge, log, stone]	[stone_axe, stone_sword]	[carrot, beetroot]	[wheat_seeds, carrot, potato]	[iron_pickaxe, iron_axe, iron_sword]	[iron_pickaxe, iron_axe]
Demonstration						

Figure 6.2: We present example task configurations, as a combination of distinct biomes, playground base blocks, task goals, target blocks materials and agent counts. Agents are initialized with unique inventories, which provide them with different capabilities to complete various activities. A detailed distribution is provided in the supplementary.

Figure 6.1.

Building: Agents erect a structure based on a provided orthographic view blueprint. Each agent possesses a unique inventory of building blocks necessary for the construction. Successful completion requires visual cognition to associate blueprint components with inventory items, spatial reasoning to reconstruct a 3D structure from 2D images and map it to 3D coordinates for action targets, and collaborative coordination with other agents to resolve action dependencies. For example, an agent cannot place a floating block and should wait for another agent to build the supporting block first.

Clearing: Agents are required to remove all blocks from a specified area. Besides spatial understanding and awareness of action dependencies, agents must employ appropriate tools to break blocks, which vary in durability, thereby requiring multiple interactions for complete removal. The use of correct tools can dramatically reduce the time required to remove blocks. Thus agents must coordinate task assignments to optimize block-breaking efficiency. Strategic coordination is essential in this task as agents need to dynamically decide which blocks to target based on their current tools, and assist each other even without the optimal tools when necessary.

Farming: Agents sow and harvest crops on designated farmland plots. They must monitor crop growth stages, from newly planted to fully grown, and harvest only when crops reach maturity. Efficient task completion requires spatial reasoning to select appropriate farmland, visual cognition to assess crop maturity, and continuous updating of farmland states based on other agents' actions. As the available farmland exceeds what is needed, understanding other agents' actions to avoid redundancy, and dynamically allocating sub-tasks based on positions, available seeds, and crop maturity are essential. For example, some agents can sow while others are harvesting, stop when the total yield meets the goal.

Smelting: Agents obtain processed items using furnaces by gathering materials and coordinating actions. They collect resources from the environment, by harvesting blocks or killing mobs, or use existing inventory items to produce goal items like cooked food or refined materials. Agents also need to gather fuel before they can make use of furnaces. Efficient task completion requires spatial understanding to locate furnaces and resources, and coordinating actions with inter-agent dependencies. For instance, if one agent is collecting beef, others should focus on gathering fuel rather than duplicating efforts. Working as a team to use limited furnaces efficiently is crucial, rather than each agent independently smelting their own goal item.

6.1.4 Centralized and Decentralized Agents

TeamCraft supports centralized and decentralized control.

Centralized Agents: The centralized model is given the observational data of all agents, including the first-person view, action history, and inventory information. Based on these comprehensive data, the model generates the actions for all agents simultaneously. This approach leverages the full scope of information available in the environment to coordinate and optimize the actions of all agents collectively.

Table 6.1: Task variants and dataset statistics

	Building	Clearing	Farming	Smelting
# Action Sequences	2 – 6	2 – 9	2 – 7	2 – 8
# Agents	2 – 3	2 – 3	2 – 3	2 – 3
# Tools	–	1 – 4	–	1 – 4
# Scenes	6	5	4	5
# Base Types	10	11	9	11
# Furnaces	–	–	–	1 – 2
# Target Block Types	19	16	3	13
# Target Block Counts	5 – 12	4 – 9	2 – 14	1 – 4
# Fuel Types	–	–	–	12
# Resource Types	–	–	–	20
# Dimensional Shapes	2	2	2	1
# Placement Shapes	7715	12724	13188	8885
# Total Demonstrations	14998	14641	14815	10803
# Test Set	50	50	50	50
# Generalization Set	200	200	150	200
# Generalization Conditions	4	4	3	4

Decentralized Agents: The decentralized models do not receive information about other agents except for the initial inventory of the team. Each model generates actions solely for the individual agent based on its limited view. This setting simulates a more realistic scenario where agents operate independently with restricted information, focusing on their actions absent of any centralized coordination.

6.1.5 Diversity

The tasks are complex and challenging, testing multi-agent systems in diverse settings. Table 6.1 provides task statistics and variants, with visual diversity detailed in supplementary.

Object Diversity: More than 30 target object or resource are used in tasks. Objects, such as a fence, an anvil, or a stone block, have different shapes and textures. Farm crops have different visual appearances during growth stage. The smelting task has resources with different appearances, such as chickens or rabbits.

Inventory Diversity: Agent’s inventory might include essential items mixed with non-essential ones (i.e., distractors), realistically simulating scenarios where agents must choose the right materials for specific tasks while managing inventory constraints. Agents are also

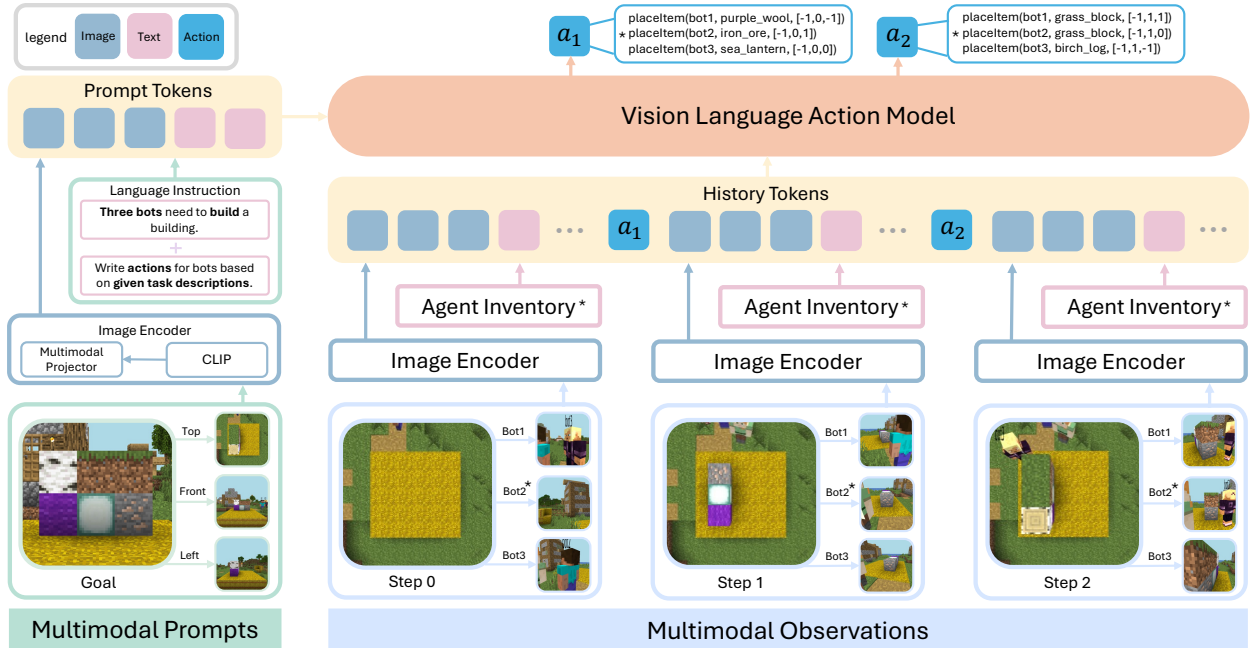


Figure 6.3: The architecture of the TeamCraft-VLA model. Multi-modal task specifications combining three orthographic views images of the task goal states and corresponding language instructions are encoded as initial input to the model. Agents inventories and visual observations are further encoded in each step to generate actions for agents. For decentralized setting, the model only has access to one agent’s information, exemplified by Bot2: items associated with a * represent the fact that only the data associated with agent 2 are available.

provided with random tools for each task. Having the appropriate tools significantly enhances efficiency in tasks like clearing. For smelting, some resources must be collect by agent with specific tools.

Scene Diversity: More than 10 scenes are included in the tasks, covering biomes such as village, mountain, forest, swamp, desert, etc. Tasks take place on grounds with diverse textured bases such as glass, concrete, and quartz. Certain tasks may involve additional complexity, including farmland which are intermixed with non-plantable blocks.

Goal Diversity: Each task requires achieving a varying number of goal targets. Building requires different blocks placed into various shapes, categorized based on dimensionalities, e.g., 2D (all blocks are at the same level) or 3D (some blocks are on top of others). Farming requires various target crops and yields. For the smelting task, the target object is sampled

from various food or processed items.

6.1.6 Tasks and Expert Demonstrations Generation

To create a rich learning environment and effective imitation learning dataset, systematic scenario design and data collection methods are employed, as follows:

Task Generation: Variables from a diversity pool, such as agent counts, scenes, and goals, are sampled to establish task configurations. Specifically, a solvable task is formulated by rejection sampling of the essential task variables. "Solvable" implies that the task can be completed within the Minecraft world rules and is within the agents' capabilities. For example, in smelting tasks, fuel must either be available to collect in the scene or directly accessible in the inventory.

Planner-Based Demonstrations Generation: Given the task specifications, a planner assigns actions to agents at every time step, utilizing privileged information of the environment. Assume agent i performing action j , the planner optimizes a cost function designed to minimize total task completion time T , idle actions E_i , action dependencies D , redundant actions U , and the cost c_{ij} for agent i performing action j :

$$C = w_1T + w_2 \sum_{i=1}^N E_i + w_3D + w_4 \sum_{i=1}^N \sum_{j \in A_i} c_{ij} + w_5U \quad (6.1)$$

where w_1, w_2, w_3, w_4, w_5 are weighting coefficients. Details of the weights are available in the supplementary.

As shown in Table 6.1, we generated 55,000 unique task variants, each with one demonstration. A demonstration consists of a multi-modal prompt as task specification, including three orthographic view images representing task initial states or goal states and the corresponding language instructions. At each time step, agent inventories, first-person RGB observations and actions are recorded.

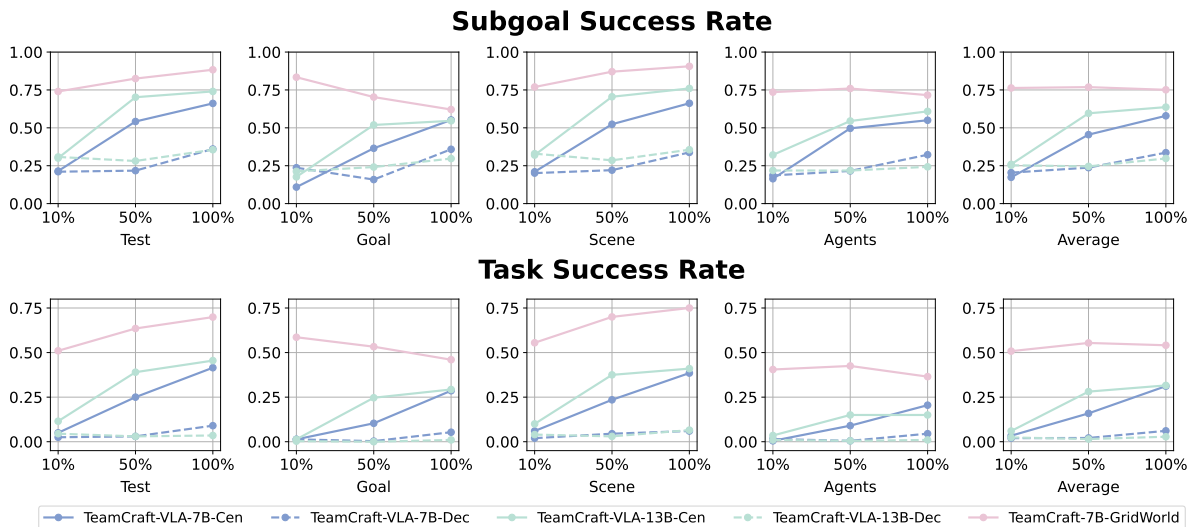


Figure 6.4: Subgoal success rate and task success rate across centralized, decentralized and grid-world settings. The leftmost column displays the *Test* category, which shares similar data distribution as training. The *Goal*, *Scene* and *Agents* categories represent generalization tasks involving unseen goals, scenes, and tasks involving four agents, respectively. Average performance is presented in the rightmost column.

6.1.7 Test Set and Generalization Set

TeamCraft features a test set, where agents are initialized with random position, orientation, and inventory. Other variables follow the same distribution as training. To evaluate the model generalization, we further designed a generalization set with hold-out elements excluded from training data. In general, we withheld test cases involving *four agents*, whereas the training data include only two or three agents. We also introduced unseen *scenes* not present during training. In addition to these general hold-outs, we implemented task-specific exclusions as following: 1) Building: novel *shapes* and *materials* to build. We exclude 8 block placement *shapes*, defining how target blocks are arranged on the ground. These shapes varied in complexity, containing 5 to 12 blocks in both 2D and 3D configurations. Additionally, we omitted 3 block *materials* appeared in clearing but not in building. 2) Clearing: novel *shapes* and *materials* to clear. We held out 6 block placement *shapes* with block counts ranging from 4 to 9. We also excluded 3 block *materials* present in building but absent in clearing. 3) Farming: novel *crops* to farm and collect. 4) Smelting: novel number of *furnaces* and *goal* objects. We excluded 4 unseen *goal* objects and introduced scenarios with novel number of

furnaces in the scene. As shown in Table 6.1, with 50 samples per task for the test set and each generalization condition, our benchmark contains a total of 950 test cases.

6.2 Experiments

6.2.1 Baselines and Ablations

TeamCraft-VLA: We introduce TeamCraft-VLA (Vision-Language-Action), a multi-modal vision-language action model designed for multi-agent collaborations. As shown in Figure 6.3, the model first encodes multi-modal prompts specifying the task, then encodes the visual observations and inventory information from agents during each time step to generate actions. Following Liu et al. (2024), the VLA model architecture consisting of a CLIP encoder for images, a projector to align the image features with language model. We use CLIP ViT-L/14 as the visual encoder and a linear projector for modality alignment. The model is trained on the demonstration data for three epochs before convergence.

Proprietary VLA: We use GPT-4o as the proprietary VLA. Specifically, we use similar prompt structures as the centralized finetuned TeamCraft-VLA model, with additional task information in the initial system prompt to provide background knowledge of the task. The system prompt contains recipes, input, output formats, all available blocks, items, workspace limitations, and one successful rollout of a similar task in the same task family. At the first step, we additionally provide the first user prompt, where the model is given a specific multi-modal task specification accompanied by initial visual observations and inventory details of the agents. Based on the system prompts and user prompts, the model predicts the actions. As the interaction progresses with subsequent prompts, the context is maintained and expanded with the addition of prior responses and updated visual data. Detailed prompts are available in the supplementary.

Grid-World Settings: In order to understand the impact of learning in multi-modal environment as opposed to purely text-based or state-based environment, we perform an

ablation study by translating the *TeamCraft* environments into a 3D grid-world. We retain the same prompt structure of the training data used in the TeamCraft-VLA models, with the main difference being that environmental information (i.e. visual observations and three orthographic view images) is now represented in text, describing the voxel coordinate of each block, e.g. "brick is at (2,3,0), stone is at (2,3,1)...". We fine-tuned a LLM in centralized setting with variance in the dataset size (10%, 50%, and 100% of the total data) for three epochs before convergence.

Ablations: We performed a total of 15 ablation studies, varying in dataset sizes (10%, 50%, and 100% of the total data), control settings (centralized and decentralized), experiment settings (Multi-Modal and Grid-World) and sizes of the VLA model (7B and 13B).

6.2.2 Evaluation Metrics

We evaluated the performance of the methods based on three key metrics: task success rate, subgoal success rate and redundancy rate.

Subgoal Success Rate: This metric evaluates the effectiveness of agents in completing tasks. Given M test cases, each test case m has s_m^g subgoals, and agents complete s_m^d subgoals. The subgoal success rate SGS is defined as:

$$SGS = \frac{1}{M} \sum_{m=1}^M \frac{s_m^d}{s_m^g} \tag{6.2}$$

Specifically, subgoals are designed based on the task requirements, i.e. the number of blocks to be built for building and the number of target objects to be created for smelting.

Task Success Rate: This metric indicates the proportion of test cases that the model can successfully complete from start to finish. Specifically, the task success rate TS is defined as:

$$TS = \frac{1}{M} \sum_{m=1}^M \mathbb{1} [s_m^d = s_m^g] \tag{6.3}$$

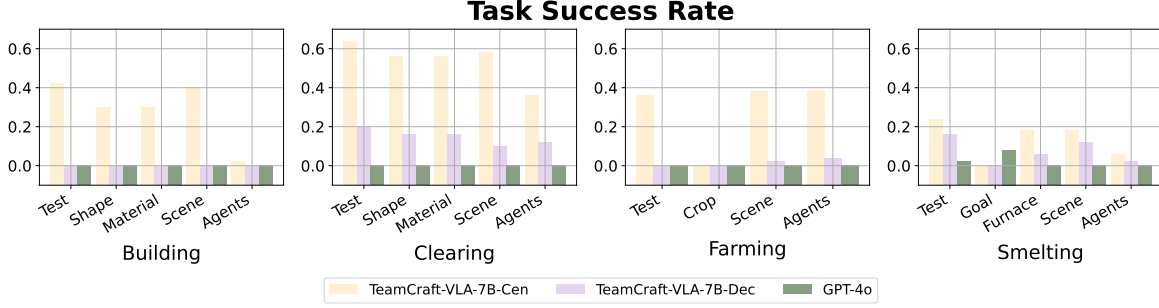


Figure 6.5: Task success rates of centralized and decentralized VLA model and GPT-4o. Models are trained with the full data except GPT-4o. TeamCraft-VLA-7B-Cen outperforms the other two methods by a significant margin across nearly all variants.

A higher success rate reflects the model’s ability to consistently achieve the desired outcomes in various scenarios.

Redundancy Rate: This metric assesses whether multiple agents are performing the same action at the same time, which would lead to conflicts. Assume p_m is the total number of actions for test case m and q_m the number of conflicts between agents, the redundancy rate RR is defined as:

$$RR = \frac{1}{M} \sum_{m=1}^M \frac{q_m}{p_m} \quad (6.4)$$

A lower redundancy rate indicates better task allocation among agents and a higher level of cooperative efficiency.

6.2.3 Evaluation Results

We evaluated the subgoal success rate and task success rate of the models. As illustrated in Figure 6.4, our analysis and findings are discussed below:

Success Rate: For both the 7B and 13B models, the subgoal success rate and task success rate fall short of optimal performance. This is particularly evident in challenging tasks such as smelting, with both subgoal and task success rates below 40%. This highlights the inherent difficulty of the designed tasks and underscores the current limitations of VLA models in handling multi-step, sequentially dependent processes.

Across Model Size: In Figure 6.4, we observe that as training data increased, the performance of the 7B model approaches that of the 13B model, especially when generalizing to novel goals and number of agents. This suggests that scaling up model sizes blindly do not guarantee success.

Multi-Modal Environment vs. Grid-World: The performance of the language model in the text-based Grid-World significantly surpasses VLA models in multi-modal settings. This suggests that state descriptions provided purely in text format are less challenging for models than multi-modal inputs, underscoring a notable gap in current VLA models’ ability to effectively interpret visual information. For the language model, we observe a surprising trend in the *Goal* and *Agents* splits: training with more data lower the success rate. This decline suggests that the generalization capacity for certain task categories actually diminishes as training goes on. One possible cause is when exposed to more data, the model relies more heavily on patterns specific to the training examples, limiting its ability to adapt to unseen scenarios.

On Generalization Splits: For VLA models, performance generally drops when models transfer to novel generalization splits, especially in the *Goal* and *Agents* categories. The *Scene* split primarily tests image understanding, while the *Goal* and *Agents* splits emphasize task planning and allocation, critical factors in multi-agent systems. This indicates that VLA models still struggle with planning for unseen goals and adapting to variable numbers of agents.

Scaling Law: As training data increases, we observe significant improvements in both subgoal and task success rates across centralized and decentralized settings, underscoring the importance of dataset size for achieving better performance. The improvement is particularly pronounced when the training data increases from 10% to 50% in centralized settings. This suggests that while more data generally leads to better performance, gains diminish beyond a certain point, especially in the decentralized setting.

Table 6.2: Comparison of TeamCraft-VLA model redundancy rates

	Test	Goal	Scene	Agents	Average
TeamCraft-VLA-7B-Cen	0.01	0.02	0.01	0.01	0.01
TeamCraft-VLA-7B-Dec	0.13	0.12	0.13	0.24	0.15

Centralized vs. Decentralized: Figure 6.5 compares centralized and decentralized settings in terms of subgoal and task success rates across all task variants. Centralized tasks exhibit significantly better performance across nearly all variants, highlighting the challenge of effective planning with partial information. This finding also demonstrates that multi-agent systems cannot be simplistically modeled as single agents interacting with environments containing other agents. In decentralized settings, the absence of comprehensive agent modeling is particularly impactful, especially for cooperation-intensive tasks like "Farming" or "Building".

Redundancy Rate: Table 6.2 compares redundancy rates between centralized and decentralized settings. Our results show that decentralized agents exhibit significantly higher redundancy rates than centralized agents, indicating reduced efficiency in task planning and allocation. This inefficiency becomes even more pronounced as the number of agents increases, creating greater challenges for effective task allocation. In decentralized settings, the absence of centralized control complicates the avoidance of redundant work, as each agent must independently infer the intentions of others to prevent duplication. By contrast, a centralized controller can efficiently assign distinct tasks to each agent, minimizing overlap and enhancing overall efficiency. These findings suggest that VLA models lacking explicit mechanisms to understand or infer the actions of other agents, highlighting a critical need for improved inter-agent communication and awareness within decentralized systems.

GPT-4o Result: We evaluate GPT-4o on in a one-shot prompt learning setup and it failed on almost all test cases. A detailed analysis reveals that GPT-4o struggles with mapping block coordinates based on visual inputs, demonstrate a lack of 3D spatial reasoning needed for accurate task execution. This shortcoming severely impacts performance, since most of

our tasks require precise spatial orientation and alignment. For example, in building task, a brick should be placed at (8,0,8), while the output of model is "placeItem(bot1, 'bricks', (7,0,9))" which leads to wrong execution.

6.2.4 Qualitative Analysis

We performed a qualitative analysis across three generalization splits, examining how models handle novel goals, new scenes, and novel number of agent:

Goals: When faced with novel goals, the models struggle to generalize beyond familiar items and often fail to adapt to specific, unseen objectives. For example, in the "farming" task, if instructed to farm beetroot—a crop not encountered in training—the model might generate a command like "farm_work(bot1, (9,3,3), 'sow', 'beef')," causing Bot1 to sow "beef", which appears in the training data for "smelting". This behavior reflects the model's reliance on similar, previously seen items in the training data and reveals its limited ability to infer new tasks based solely on partial similarity.

Object State Recognition: VLA models show strong generalization to new scenes, performing comparably to the *Test* set. However, errors often arise in recognizing object states. For example, in "farming" tasks, agents may harvest crops before they are fully grown due to challenges in identifying crop states, especially when encountering new scenes. This highlights limitations in precise object state recognition when operating within unseen environments.

Agents: For generalization to four agents, models frequently ignoring the fourth agent and assigning tasks inefficiently only to two or three agents. For example, for the building task, the model predicts the action sequences {"placeItem(bot1, 'birch_log', (4,4,7))", "placeItem(bot2, 'sandstone', (4,4,6))", "placeItem(bot3, 'dirt', (3,4,6))"} with the fourth agent overlooked, reducing productivity and sometimes preventing timely task completion. This limitation becomes especially evident in tasks requiring full coordination, such as "Building." In these

tasks, each of the four agents holds unique blocks in their inventory, and all agents must contribute their specific block to a shared platform to complete the structure. The model's inability to distribute tasks effectively across all agents often leads to incomplete structures or outright task failure. This highlights a significant limitation in precise coordination and workload distribution necessary for successful multi-agent collaboration.

CHAPTER 7

Conclusions

In this dissertation, we have addressed four key challenges in the field of adaptive multi-agent systems.

To tackle the issue of scalability, we proposed Scale Multi-Agent Reinforcement Learning (Scale-MARL), which leverages curriculum learning over agent populations combined with evolutionary selection. Our approach demonstrates significant improvements over baseline methods, not only in performance but also in training stability. These promising results across various environments suggest that our method is generalizable and could potentially enhance the scalability of other MARL algorithms. Furthermore, we envision that training with large agent populations may foster the emergence of swarm intelligence in environments governed by simple rules.

To enable adaptation to novel tasks, we introduced a learnable gradient-based state representation called Social Gradient Fields (SocialGFs). SocialGFs are trained using a set of exemplar states via denoising score matching. By incorporating these fields into the state and reward representations, agents can efficiently learn policies through MARL. Our method significantly outperforms baseline approaches and demonstrates remarkable adaptability across tasks. These results provide a fresh perspective on enhancing the adaptability of multi-agent systems, showcasing the potential of SocialGFs to facilitate task generalization.

To adapt to novel agents, we presented the Inverse Attention Agent, which utilizes inferred attention weights of other agents to adjust its own focus and actions. This mechanism enables the agent to adapt its behavior dynamically, showing superior performance when interacting with unseen agents trained by various methods as well as with human participants. The adaptability of the Inverse Attention Agent highlights its potential for robust interactions in

diverse settings.

Finally, to create a complex and dynamic environment for testing multi-agent systems, we introduced TeamCraft, a benchmark for multi-modal, multi-agent collaborative task planning in Minecraft. TeamCraft features challenging tasks and evaluation splits designed to assess agent performance across novel goal configurations, unseen team sizes, and unfamiliar environments. Through extensive experiments, we identified limitations of current models and outlined promising research directions for developing collaborative, multi-modal agents.

7.1 Limitations and Future Work

For task adaptiveness, one modification is to rank the importance of each SocialGF at the start of generalization to a new environment. This prioritization can enhance adaptability across tasks. Extending SocialGFs to more photo-realistic 3D environments, such as UnrealCV (Qiu et al., 2017), is also a key direction for future work.

For agent adaptiveness, the current attention inference mechanism is limited to agents of the same type. In future work, we aim to model the theory of mind for agents of different types. Additionally, we plan to develop an attention model for the UW network that can dynamically accommodate an arbitrary number of inferred attention weights.

For the TeamCraft benchmark, given the limited capabilities of existing multi-agent vision-language-action (VLA) models, *TeamCraft* currently relies on MineFlayer as an oracle controller to execute skills predicted by the models. A critical avenue for future research is enabling VLA models to directly control multiple agents via low-level actions (Wang et al., 2023c,b). Our models have been trained using procedurally generated multi-agent demonstration data. Incorporating noisy yet diverse real-world demonstrations from human players could further improve model generalization (Baker et al., 2022; Fan et al., 2022). Currently, decentralized *TeamCraft* agents rely solely on implicit communication (Jain et al., 2019), passively perceiving other agents and the environment to gather information and collaborate. Empowering agents to communicate explicitly through natural language (Narayan-Chen et al., 2019; Jayannavar et al., 2020; Mandi et al., 2024) holds great potential

for reducing redundant actions and enhancing efficiency. Multi-player video games have long been utilized as testbeds for human-AI collaboration (Carroll et al., 2020; Gao et al., 2020; Amresh et al., 2023). Extending *TeamCraft* to include human players represents an exciting and promising research direction.

APPENDIX A

Evolutionary Population Curriculum Details

A.1 Environment Details

In the grassland game, sheep gets +2 reward when he eats the grass, -5 reward when eaten by wolf. The wolf get +5 reward when eats a sheep. We also shape the reward by distance, sheep will get less negative reward when it is more closer to grass and wolf will less negative reward when it is more closer to sheep.

In the Adversarial battle game, agent will get +1 reward when he eats the food, -6 reward when killed by other agents. If N agents kill an enemy, they will be rewarded $+6/N$. We shape the reward by distance. Agent will receive less negative rewards when it is more closer to other agents and grass. We want to encourage collision within agents and also will be easier for them to learn to eat.

In the food collection game, there are N agents and N food locations. Each agent will get a shared $+6/N$ reward per timestep when one food is occupied by any agent. If one agent gets collision with another, all of the agents will get a punish of $-6/N$. We shape the reward by distance. Agents will receive less negative rewards when it gets closer to the food. Since the number of agents and food are equal, we want to avoid the collision within agents and let the agents to learn to occupy as many food as possible.

We use the *normalized reward* as the score during evaluation. For a particular game with a particular scale, we first collect the reward for each type of agents, namely the average reward of each individual of that type *without* the shaped rewards. Then we re-scale the collected rewards by considering the lowest reward among all methods as score 0 and highest reward as score 1.

A.2 Training Details

We use the Adam optimizer with learning rate 0.01, $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\varepsilon = 10^{-8}$ across all experiments. $\tau = 0.01$ is set for target network update and $\gamma = 0.95$ is used as discount factor. We also use a replay buffer of size 10^6 and we update the network parameters after every 100 samples. The batch size is 1024.

We set $K = 2$ in all the games during training except that $K = 3$ in the food collection game. During EPC training, in the grassland game, we train the scale of 3 sheep 2 wolf for 100000 episodes. We train another 50000 episodes every time the agents number doubles. In the adversarial battle game and food collection game, we train the first scale for 50000 episodes. We train another 20000 episodes every time the agents number doubles.

In the grassland game, the entity types are the agent itself, other sheep, other wolf and food. We thus have four types of entity encoders for each of those entity types. In the adversarial battle game, Similar to grassland game, the entity types are agent itself, other teammates, enemies and food. We also have four types of entity encoders for each of those entity types. Since there is only one group in the food collection game, the entity types are agent itself, other teammates and food. We thus have three entity encoders in our network.

A.3 Evaluation Details

To evaluate the agents trained in the environment with $\Omega = 2$, we make two roles of agents trained with different approaches compete against each other. Each competition is simulated for 10000 episodes. The average normalized reward over the 10000 episodes will be used as the competition score for each side. Note that in our experiments, we let all the methods compete against our EPC approach for evaluation. For adversarial battle game, we take the average score of two teams as the model’s final evaluation score, since the two teams in this game are completely symmetric.

In the food collection game, since there is only one role, we simply simulate the model for 10000 episodes. The average normalized reward over the 10000 episodes will be used as the

score of the model.

APPENDIX B

SocialGF Details

B.1 Algorithm Details

algorithm 4 conducts the entire training process. It begins by categorizing each example from the given set S into specific example sets E_i based on triggered events e_i . Subsequently, it trains various categories of target score network Φ_i using Denoise Score Matching (algorithm 1) in all the example sets E_i . Next, it obtains a representation of the environment, denoted as O_{GF} , which consists of positive (gf^+) and negative (gf^-) gradients. Finally, it carries out MARL training on the environment represented by O_{GF} and the reward function R_E .

B.2 Environment Details

B.2.1 Reward

In the grassland game, the sheep gets +2 reward when the sheep eat the grass, -5 reward when eaten by the wolf. The wolf gets +5 reward when eats a sheep. For the *Reward Engineering*, the wolf’s reward will be minus the minimum distance to the sheep, $R_{wolf-} = Min(Distance(self, sheep_{all}))$ and the sheep’s reward will be minus the minimum distance to the grass: $R_{sheep-} = Min(Distance(self, grass_{all}))$.

In the cooperation navigation games, every agent will get a reward of 10 after every landmark is correctly occupied. For the reward engineering method: the agent reward will be minus the minimum distance from each landmark to that agent: $Reward- = Min(Distance(self, landmark_{all}))$. The agent will also get a bonus +1 reward when it successfully occupies a landmark.

Algorithm 4: Whole Training Process

Given a set of example S ;
for *example* e **in** S **do**
 Categorize e to example set E_i based on triggered events e_i ; // **Categorize each example**
Train different categories of target score network Φ_i via Denoise Score Matching (algorithm 1) on all the example set E_i ;
Get gf representation of the environment: $O_{GF} \leftarrow \{gf^+, gf^-\}$;
Conduct the MARL training on the gf represented environment: $E(O_{GF}, R_E)$;

Table B.1: The hyperparameters for learning GF

lr	activation	sigma	t_0	hidden size	optimizer	optimizer betas	network
$2e - 4$	Relu	25	1	64	Adam	[0.5, 0.999]	GNN

B.2.2 Observation

In the grassland game, agents receive comprehensive information including the relative positions of other agents, grass, and obstacles, alongside the velocities of the other agents. In cooperative navigation games, in addition to the relative locations and velocities of other agents, they also gain access to color markings and relative locations of all other landmarks.

B.2.3 Action

The agent’s action is represented by a two-dimensional continuous vector, which describes the force applied to an entity, considering both magnitude and angular direction.

B.3 Training Details

We follow all the hyperparameters used by Wu et al. (2022) for GF learning based on Table B.1 and PPO (Yu et al., 2021) for MARL (Table B.2). We select 1000 examples for learning every gf and set $t = 0.01$ for computing the gf score.

Table B.2: The hyperparameters for MAPPO

lr	activation	gain	share policy	hidden size	optimizer	optimizer epsilon	network
$7e-4$	Tanh	0.01	False	64	Adam	$1e-5$	MLP

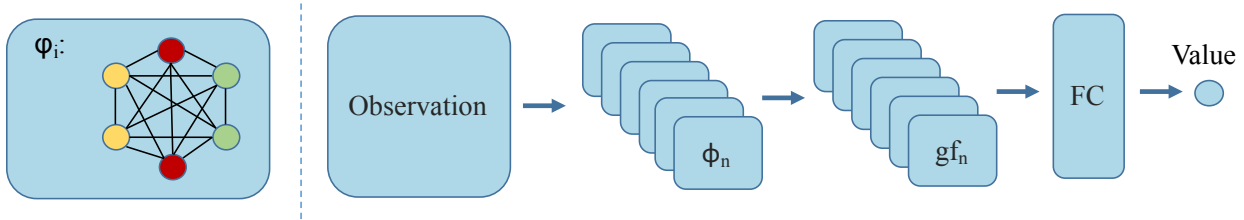


Figure B.1: The structure of the Neural Network. We delve into the network architecture devised for crafting gf representations and driving the generation of the final action. The target score network functions ϕ_i harness the capabilities of graph neural networks to adeptly capture intricate relationships among the various entities, where nodes symbolize both agents and landmarks. We construct the gradient field representation O_{GF} by applying each GF_i function ϕ_i to its corresponding category i and consolidating the resulting outputs gf_i through concatenation. This concatenated representation O_{GF} undergoes further refinement through a fully connected layer FC, culminating in the production of the final action.

B.4 Network Structure

In this section, we delve into the network architecture devised for crafting gf representations and driving the generation of the final action Figure B.1. Specifically, the target score network functions ϕ_i harness the capabilities of graph neural networks to adeptly capture intricate relationships among the various entities, where nodes symbolize both agents and landmarks. The target score network seamlessly embeds observations into distinct gradient fields gf_i . The structure of ϕ is based on the codes from the TarGF (Wu et al., 2022). It contains two hidden dimensions with a hidden dimension size of 64 and then connects to the EdgeConv layer to get the final output.

Throughout the process of MARL training, we construct the gradient field representation O_{GF} by applying each GF_i function ϕ_i to its corresponding category i and consolidating the resulting outputs gf_i through concatenation. Subsequently, this concatenated representation O_{GF} undergoes further refinement through a fully connected layer (FC), culminating in the production of the final action. FC is identical for all the tasks based on the codes from the

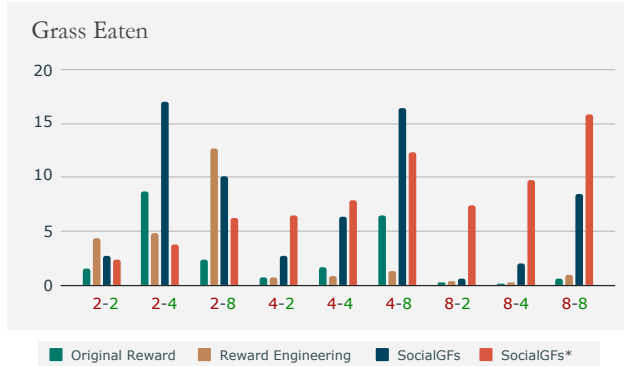


Figure B.2: Grass eating ability of different methods in the grassland game. Original Reward and Reward Engineering sheep both fail to learn to eat grass, especially when there are more wolves. SocialGFs sheep learn to eat grass successfully, and the amount of grass they eat is inversely proportional to the number of wolves

MAPPO (Yu et al., 2021). It has two hidden layers and the hidden dimension is 64.

B.5 Evaluation Results Details

In the grassland game, all the reward and grass-eaten statistics are generated by competing against the *SocialGFs* wolf and sheep in 1000 episodes.

The sheep have a more challenging role to train in this environment than wolves because they must not only avoid wolves but also collect as much grass as possible. In addition to rewards, we also show the grass-eating ability of sheep in the rightmost figure in Figure B.2. This shows how much grass is collected by sheep in 100 time steps. We can see that Original Reward and Reward Engineering sheep both fail to learn to eat grass, especially when there are more wolves. This is because being eaten by wolves is a huge punishment for sheep, and the rewards for eating grass are difficult to explore. SocialGFs sheep learn to eat grass successfully, and the amount of grass they eat is inversely proportional to the number of wolves.

In cooperation navigation games, we also show occupation rate in Table B.3. It shows how much time a landmark is occupied during the game. We can see from the table that *SocialGF+* outperforms all the other methods in almost all the tested scenarios. The reward engineering method helps the occupation performance but is not good enough to lead the

Table B.3: The occupation rate in cooperative navigation games

	Vanilla Navigation				Color Navigation				Team Navigation			
	2	3	4	5	2	3	4	5	2	3	4	5
<i>originalreward</i>	0.005	0.008	0.021	0.031	0.02	0.025	0.028	0.049	0.00	0.002	0.005	0.005
<i>RewardEngineering</i>	0.189	0.094	0.082	0.089	0.32	0.13	0.079	0.1	0.044	0.025	0.025	0.026
<i>SocialGFs</i>	0.560	0.880	0.744	0.300	0.08	0.106	0.103	0.101	0.0	0.002	0.003	0.004
<i>SocialGFs⁺</i>	0.570	0.876	0.744	0.734	0.719	0.719	0.734	0.719	0.532	0.563	0.583	0.517
<i>SocialGFs[*]</i>	0.496	0.770	0.67	0.668	0.651	0.655	0.685	0.69	0.376	0.440	0.463	0.472

agents to the success state shown in Table 4.2. We run 1000 episodes to calculate the occupation rate and final success rate.

APPENDIX C

Inverse-Att Agent Details

C.1 Environmental Details

Reward: In the Spread scenario, during training, agents are awarded +100 reward for occupying a landmark at every timestep. For reward engineering, the agent’s reward will be deducted proportional to the minimum distance to the landmarks $R_{\text{distance}} = 0.2 \min(\text{Distance}(\text{self}, \text{Landmark}_{\text{all}}))$. In the Adversary scenario, wolves are awarded with +100 reward for every sheep it catches, while sheep are awarded with -100 reward for every time it gets caught. The wolves reward will be deducted proportional to the minimum distance to the sheep $R_{\text{distance}} = 0.2 \min(\text{Distance}(\text{self}, \text{Sheep}_{\text{all}}))$. In the Grassland scenario, wolves are awarded a +5 reward for every sheep they catch, while sheep are awarded a -5 reward for every time they get caught and +2 reward for every landmark they occupy. The wolves’ reward will be deducted proportional to the minimum distance to the sheep $R_{\text{distance}} = 0.2 \min(\text{Distance}(\text{self}, \text{Sheep}_{\text{all}}))$. On the other hand, the sheep’s reward will be deducted proportional to the minimum distance to the landmarks $R_{\text{distance}} = 0.2 \min(\text{Distance}(\text{self}, \text{Landmark}_{\text{all}}))$. For Navigation environment, the group reward is defined as +5 reward for every landmark an agent occupied. For Tag environment, when any wolf catches a sheep, all wolves received +5 points and all sheep receive -5.

Observation: In the Spread scenario, *MAPPO* and *Self-Att* agents receive information regarding their own position, velocity, other agents’ positions, and landmarks’ positions. They use these information to generate gradient field $\{\text{velocity}, g_f\}$ as observation. *Inverse-Att* agents also receive the past actions and past observations of their teammates and have $\{$

Table C.1: Hyperparameters for the gradient field

	lr	sigma	t_0	hidden size	optimizer	optimizer betas	network
Agent GF	2e-4	25	1	64	Adam	[0.5, 0.999]	GNN
Boundary GF	2e-4	25	1	64	Adam	[0.5, 0.999]	MLP

Table C.2: Hyperparameters for agent training

	lr	gain	share policy	optimizer	critic lr	ppo epoch
MAPPO	7e-4	0.01	False	Adam	7e-4	10
IPPO	7e-4	0.01	False	Adam	7e-4	10
MAA2C	7e-4	0.01	False	Adam	7e-4	10
ToM2C*	7e-4	0.01	False	Adam	7e-4	10
Self-Att	7e-4	0.01	False	Adam	7e-4	10
Inverse-Att	7e-4	0.01	False	Adam	7e-4	10

teammate past action, teammate past observation } in addition to the gradient fields. In the Adversary scenario, the observation is the same except for the omission of $gf_{\text{landmarks}}$, and Grassland, Navigation and Tag have exactly the same observation space.

Action: The agent’s action is represented by a two-dimensional continuous vector, which describes the force applied to an entity, considering both magnitude and angular direction.

C.2 Training Details

We present the hyperparameters for gradient field as well as for all agent training.

70% of the dataset is used for training. 10% of the dataset is split into a validation dataset, which is used for early stopping. 20% of the dataset is used for testing.

Table C.3: Hyperparameters for the inverse network

lr	batch size	hidden dim	patience threshold	num epoch	Optimizer
0.001	64	64	100	3000	Adam

C.3 Gradient Field Synthetic Data Generation

We use synthetically generated data to train the Agent Gradient Field as well as the Boundary Gradient Field. Detailed instructions for synthetic data generation are as follows:

Entity Gradient Field: We randomly generated 10,000 two dimensional points within a 2×2 grid as one entity’s position. We then randomly generated another entity’s position close to the previous position such that the L1 distance is less than 10^{-5} . We mark this gf representation as gf_e . This gf function takes in the agent’s location and a relative position of another entity.

Boundary Gradient Field: We randomly generated 10000 positions $(x, y) \in [-0.8, 0.8] \times [-0.8, 0.8]$. We mark this gf representation as gf_{wall} . This gf function only takes in the position of the agent’s position.

C.4 Gradient Field Representation of the Environment

For the spread, adversary and grassland environments, we applied entity gradient field for all the other entities around that agent as entity attentions, as well as a boundary gradient field used as the environment attention. Thus we transform the observation from $\{o_{i,1}, o_{i,2}, \dots, o_{i,n}\}$ which is the information of N entities to $\{gf_e(o_{i,1}), gf_e(o_{i,2}), \dots, gf_e(o_{i,n}), gf_{\text{wall}}(o_{i,i})\}$.

C.4.1 Qualitative Results

Figure C.1 provides example screenshots of a representative match involving *Inverse-Att* agents in the Spread scenario. In these images, the blue balls represent *Inverse-Att* agents, while the small black balls are landmarks. The left figure shows the initial state of all agents. The middle figure demonstrates the *Inverse-Att* agents successfully navigating to their respective landmarks. The right figure shows the final result, where all three *Inverse-Att* agents occupy their own landmarks without conflict.

Additionally, we visualize two representative games in the Adversary and Grassland

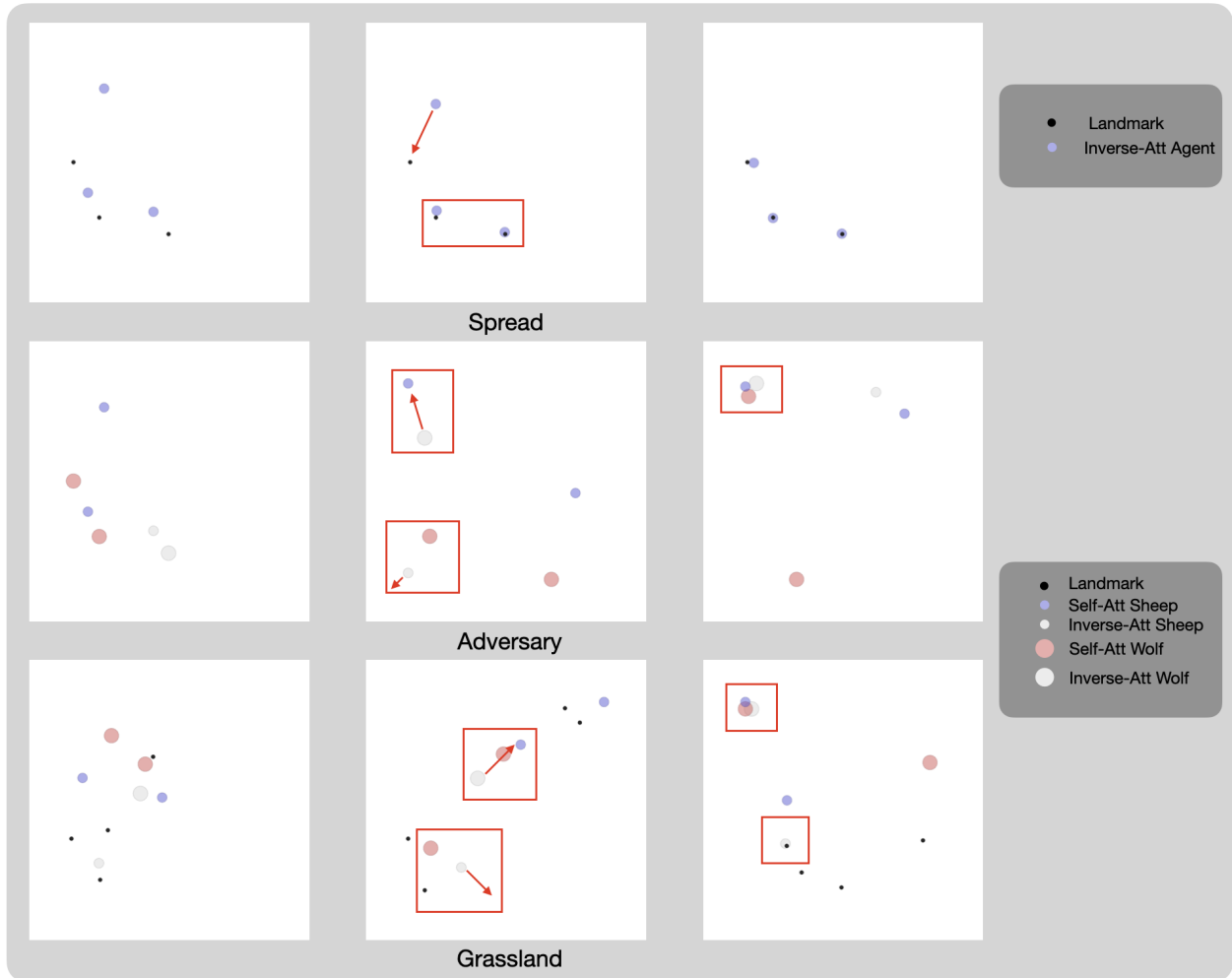


Figure C.1: Qualitative results in spread, adversary and grassland games in MPE demonstrate that Inverse-Att agents can successfully adapt to unseen agents.

scenarios. In these visualizations, the smallest black balls are landmarks. Large balls represent wolves, with red indicating *Self-Att* agents and white indicating *Inverse-Att* agents. Small balls represent sheep, with blue indicating *Self-Att* sheep and white indicating *Inverse-Att* sheep. The screenshots illustrate that *Inverse-Att* wolves cooperate with unknown *Self-Att* agents to corner the sheep. *Inverse-Att* sheep successfully avoid wolves and capture as many landmarks as possible.

These results suggest that *Inverse-Att* agents can adapt successfully to new teammates, indicating that our approach is an effective method for creating adaptive agents that excel when paired with various types of teammates and opponents.

C.4.2 Human Experiment Details

There are five experiments in total: Spread, Adversary (human plays wolf), Adversary (human plays sheep), Grassland (human plays wolf), Grassland (human plays sheep). In each experiment, human players will cooperate with three types of agents, *MAPPO*, *Self-Att*, and *Inverse-Att*, for five rounds. The episode rewards are averaged for these rounds. Human players are not informed about the type of agents that they are playing with. In the Adversary and Cooperate games, the opponents are always *MAPPO* agents.

Five participants (4 males and 1 females) participated in this experiment. All were between the ages of 21 and 28 with normal or corrected-to-normal visual acuity. All participants were given the Experiment Instructions below and received rewards (free food) for their participation. All participants were instructed to use the UP, DOWN, LEFT, and RIGHT keys on the keyboard to control their movements.

Experiment Instructions for Participants

Thank you for participating in this research experiment designed to evaluate the performance of various MARL agents in adapting to human players. There are three scenarios: Spread, Adversary, and Grassland. In the Adversary and Grassland, you will play both as wolf and sheep. In total there are five scenarios to play. In each scenario, you will cooperate with three types of MARL agent, and you will play five rounds with each type of agents.

Detailed Game Descriptions

- **Spread Game:**

- *Setup*: The game area contains several small black balls, each representing a landmark. You are represented by a white ball, and your teammates by blue balls.
- *Objective*: Each player must navigate to occupy a unique landmark. The number of landmarks equals the number of players.
- *Gameplay*: Using directional controls, navigate towards the landmarks.

Coordination with teammates might be necessary to ensure all landmarks are covered.

– *Scoring*: You earn 5 points for every timestep you occupy a landmark.

- **Adversary Game:**

– *Setup*: In this scenario, you are assigned the role of either sheep or wolves. The color of your ball is always white, with large ball indicating wolves and small ball indicating sheep. Large red balls are MARL wolves and small blue balls are MARL sheep.

– *Objective*:

- * As a Sheep: Avoid the wolves.

- * As a Wolf: Work together with other wolves to catch the sheep.

– *Gameplay*: Sheep must use speed and agility (as they move faster) to escape from wolves, while wolves need to coordinate their movements to catch sheep.

– *Scoring*:

- * As a Sheep: Each wolf catch results in a deduction of 5 points from your score.

- * As a Wolf: You gain 5 points for every sheep you catch.

- **Grassland Game:**

– *Setup*: The game environment includes the same players and roles as the Sheep Wolf Game, but now includes four small black balls (landmarks) that appear randomly in the game area.

– *Objective*:

- * As a Sheep: Collect as many landmarks as possible while avoiding wolves.

- * As a Wolf: Catch the sheep while they attempt to collect landmarks.
- *Gameplay*: Sheep must balance between quickly moving towards landmarks and evading wolves. Wolves, while trying to catch sheep, must also strategize to guard landmarks indirectly, making them risky spots for sheep.
- *Scoring*:
 - * As a Sheep: Gain 3 points for each landmark collected and lose 5 points each time you are caught by a wolf.
 - * As a Wolf: Gain 5 points for each sheep caught. Landmarks disappear once collected and respawn randomly.

Instructions for Each Turn:

- At the start of each round, your role (sheep or wolf) and the specific game scenario will be communicated.
- Use the directional controls to navigate your character according to the game's objectives. You can move in four directions: up, down, left, and right.
- The games are within a 2×2 grid. You will not be able to move out of this grid.
- The game ends until the time expires.

C.4.3 Computational Resources

Hardware specifications: All experiments are run on servers/workstations with the following configurations:

- 128 CPU cores, 692GB RAM
- 128 CPU cores, 1.0TB RAM
- 32 CPU cores, 120GB RAM

- 32 CPU cores, 120GB RAM
- 32 CPU cores, 120GB RAM
- 24 CPU cores, 80GB RAM, 1 NVIDIA 3090 GPU.
- 24 CPU cores, 80GB RAM, 1 NVIDIA 3090 GPU.
- 24 CPU cores, 80GB RAM, 1 NVIDIA 3090 GPU.
- 24 CPU cores, 80GB RAM, 1 NVIDIA 3090 GPU.

All experiments can be run on a single server with 24 CPU cores, 80GB RAM, 1 NVIDIA 3090 GPU.

APPENDIX D

TeamCraft Details

D.1 High Level Skills

The action space of agents mainly involves high-level self-explanatory skills such as *obtainBlock* and *farmWork*. We provided 8 such skills. Most skills take three input parameters, including 1) agent name such as *bot1*, as the action executing entity, 2) item name such as *dirt*, which strongly associated with task goal or agent’s inventory, 3) a vector indicating the position of the target on the test field.

For example, `obtainBlock(bot1, new Vec3(1, 0, 1))` takes the agent name `bot1` and a 3D vector `(1, 0, 1)` as its arguments. It directs `bot1` to perform multiple actions in Minecraft via APIs provided by Mineflayer. First, it controls `bot1` to `goto` a diggable position for block `(1, 0, 1)`, then has `bot1`’s vision ray cast to the block at `(1, 0, 1)` using the `lookAt` action. Next, it commands `bot1` to equip a proper tool that can dig the block at `(1, 0, 1)` most efficiently, and then instructs `bot1` to dig the target block. Once the target block has been mined, `bot1` will `goto` the position where the block item dropped and collect it.

Similarly, `farmWork(bot2, "sow", "potato", new Vec3(2, 0, 4))` takes the agent name `bot2`, action type `"sow"` (as opposed to `"harvest"`), crop seed item `"potato"`, and a 3D vector `(2, 0, 4)` as its arguments. It directs `bot2` to `goto` a placeable position for farmland at `(2, 0, 4)`, then check if the seed is a valid item—that is, a crop seed available within `bot2`’s inventory. It then checks if the farmland at `(2, 0, 4)` is plantable. Finally, it instructs `bot2` to `lookAt` the farmland and `sow` it with the seed `"potato"`.

Table D.3 documents all the skills, which are implemented in JavaScript code with Mineflayer APIs.

D.2 Low Level Atomic Actions

High level skills are processed through multiple stages before reaching the final execution APIs. At each time step, *TeamCraft* accepts a list of skills as input, with a maximum length equal to the number of agents involved in the current task and a minimum length of zero. Each agent can perform at most one skill per time step. The updated list of skills is then passed into the JavaScript environment along with the predefined atomic actions. Each atomic action is processed simultaneously, meaning that agents' actions are executed concurrently rather than sequentially. This avoids the dependency issue that might occur in sequential execution. For example, if one agent's action is executed ahead of another's, the first agent may block the location where the next agent intends to place a block. The agent whose atomic action is executed first will have a higher chance of success, potentially altering the dynamics of the multi-agent setting. Executing actions concurrently ensures fairness among agents and maintains the equivalence of the multi-agent environment.

D.3 Visual Diversity

TeamCraft uses a set of visual variates to provide a visually rich environment. Each task is constructed from a random number of agents, in a randomly selected scene, achieving different goals on a playground built from different base blocks.

D.3.1 Shared Elements

Each task begins with a basic setting involving multiple agents on a playground. Each agent has a unique skin, as illustrated in [Figure D.1](#), and is rendered as a two-block-high character. The playground combines a base platform spawned within a Minecraft biome. The base block is also randomly selected from a pool, shown in [Figure D.1](#). Each biome offers variations in special surrounding blocks, designs, and environments.

For example, the `seaside_village` biome is a village near the sea with houses made of oak wood and cobblestone, decorated with flowers and cow sheds, as shown in [Figure D.6](#). It

also features a nearby farm surrounded by oak logs (Figure D.7). Another variation of village is the **desert village** biome, built from acacia planks, acacia logs, and sandstone, blending seamlessly with the desert’s arid terrain, shown in Figure D.8. Figure D.9 illustrates a biome that is located on half of the mountains, where a small flat land protruding from a cliff. Additional examples of biomes used are shown in Figure D.10, Figure D.11, and Figure D.12.

D.3.2 Task Specific Diversity

Clearing task uses a random set of blocks as its targets, illustrated in Figure D.2. **Building** task also uses a random set of blocks as its target, with some blocks shared with clearing task, as illustrated in Figure D.3. Unlike other tasks, the **Farming** task does not use a regular base. The playground is constructed from a combination of farmland for planting crops, water blocks, and randomly selected unfarmable blockers from the base that replace some of the farmland. An example is shown in Figure D.17. Each crops used in farming task has its own grown stage with different appearances, shown in Figure D.4. **Smelting** task requires a wide varieties of resources to achieve its goal. Resources could be either entity, block, or item. Shown in Figure D.5.

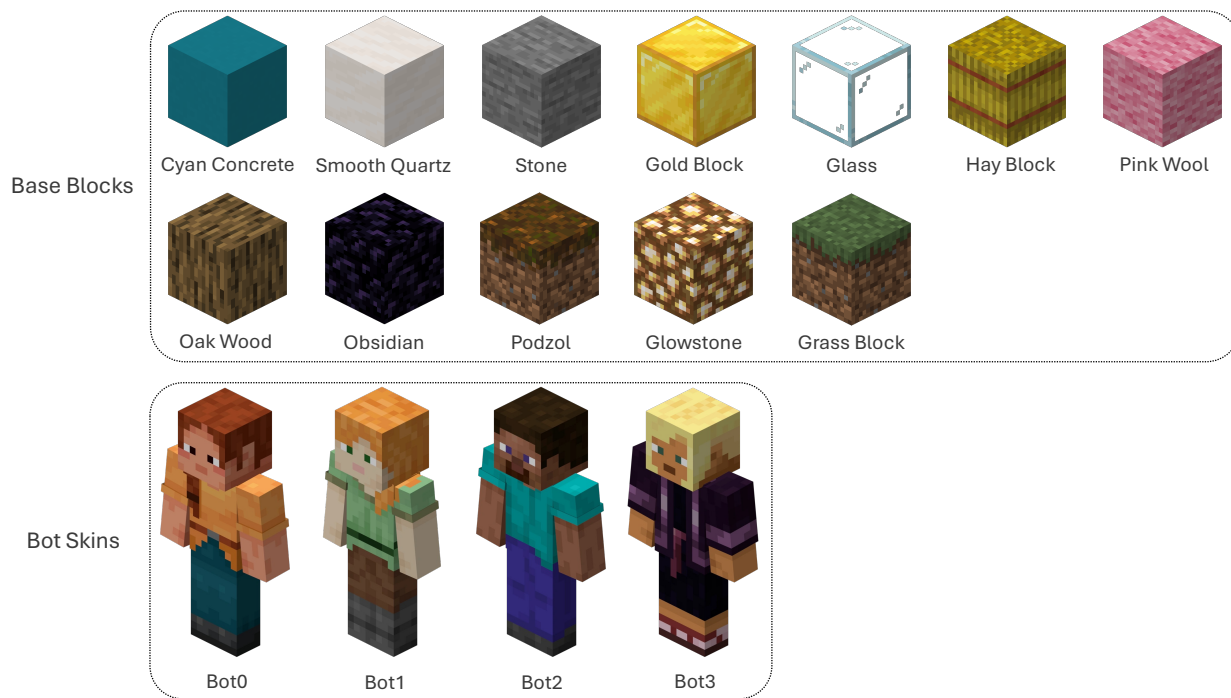


Figure D.1: Visualization of the shared visual diversity in the tasks



Figure D.2: Visualization of the visual diversity in Clearing tasks



Figure D.3: Visualization of the visual diversity in Building tasks

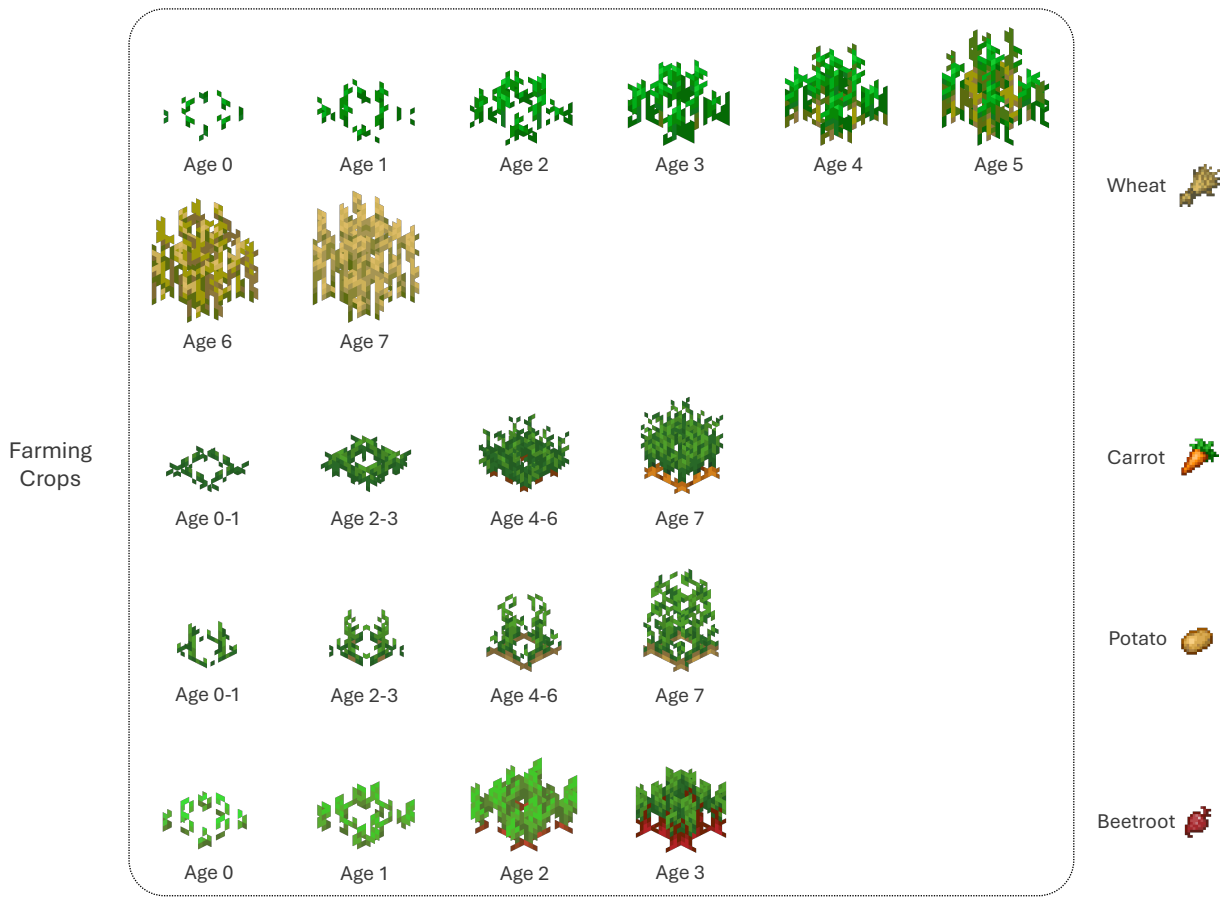


Figure D.4: Visualization of crop appearances across various growing stages in Farming tasks

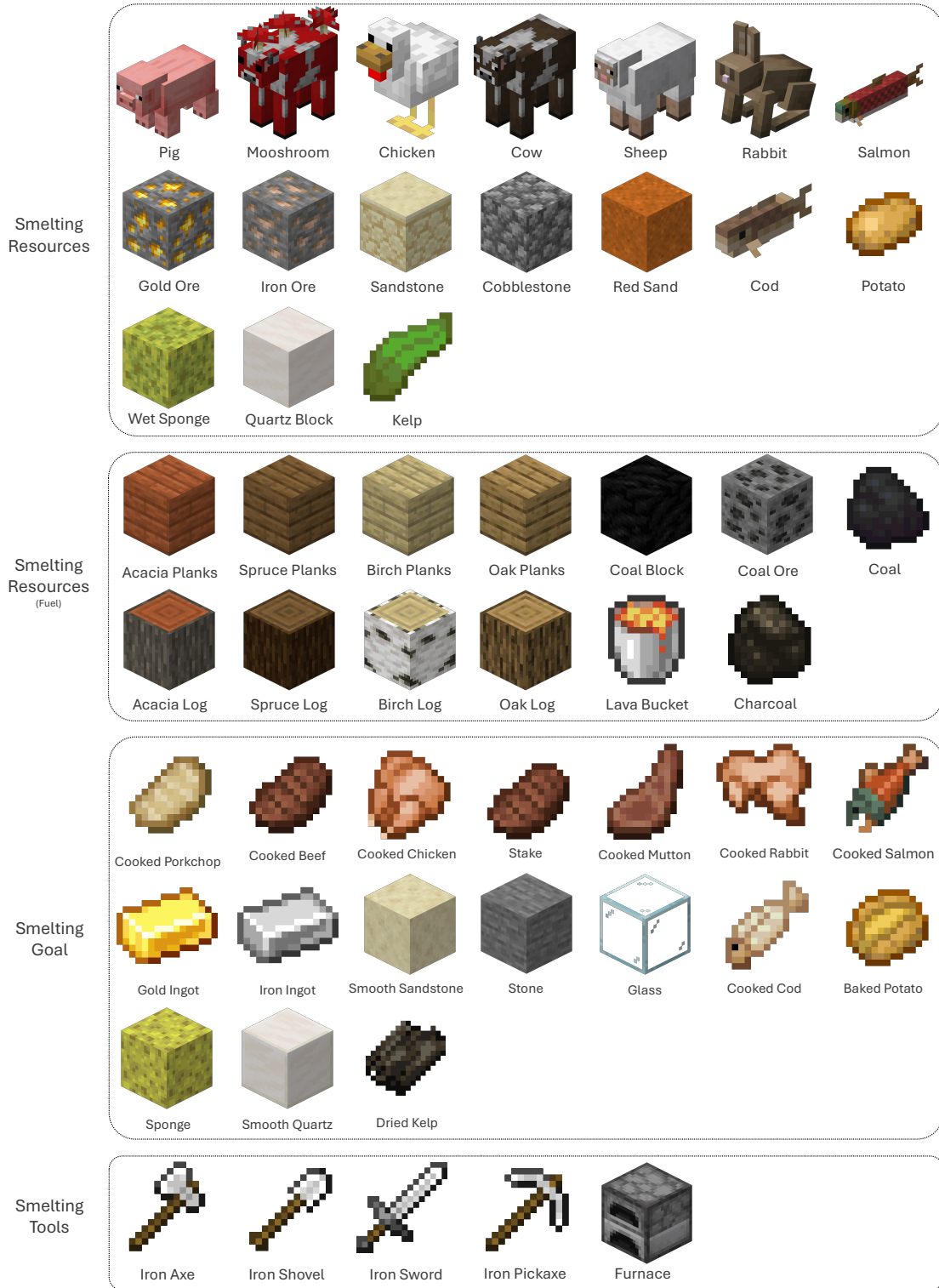


Figure D.5: Visualization of the visual diversity in Smelting tasks

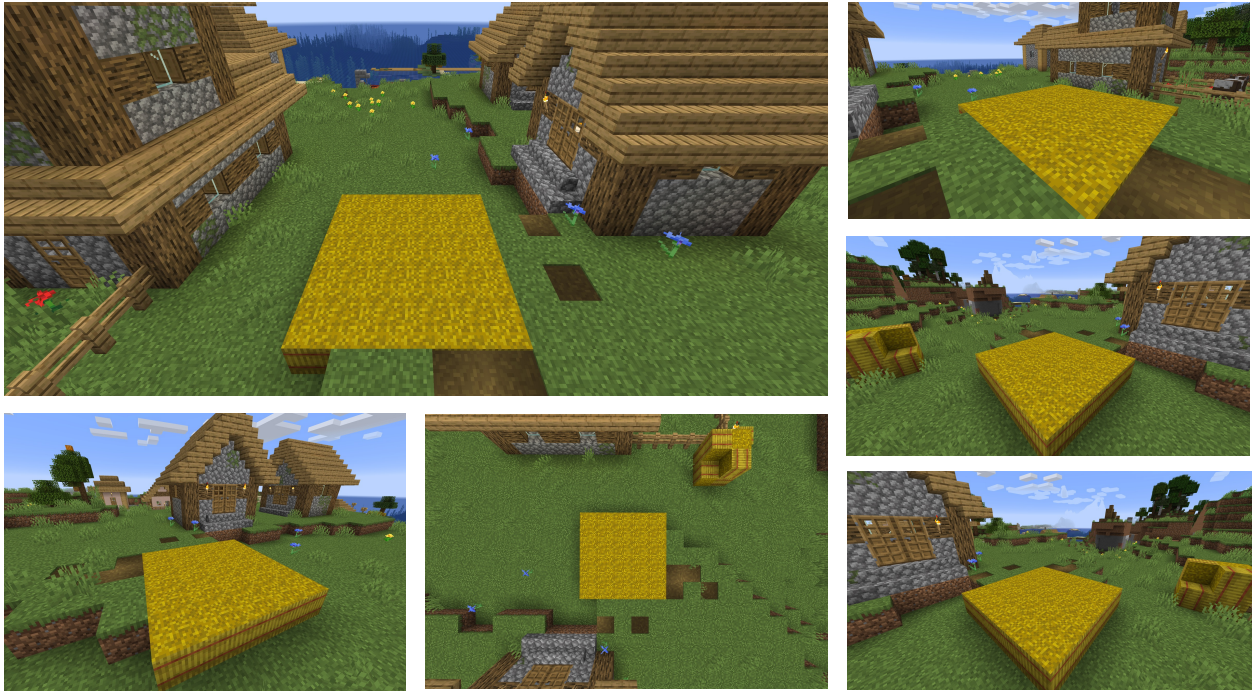


Figure D.6: An example scene in the Seaside Village biome



Figure D.7: An example scene in the Grass Village biome

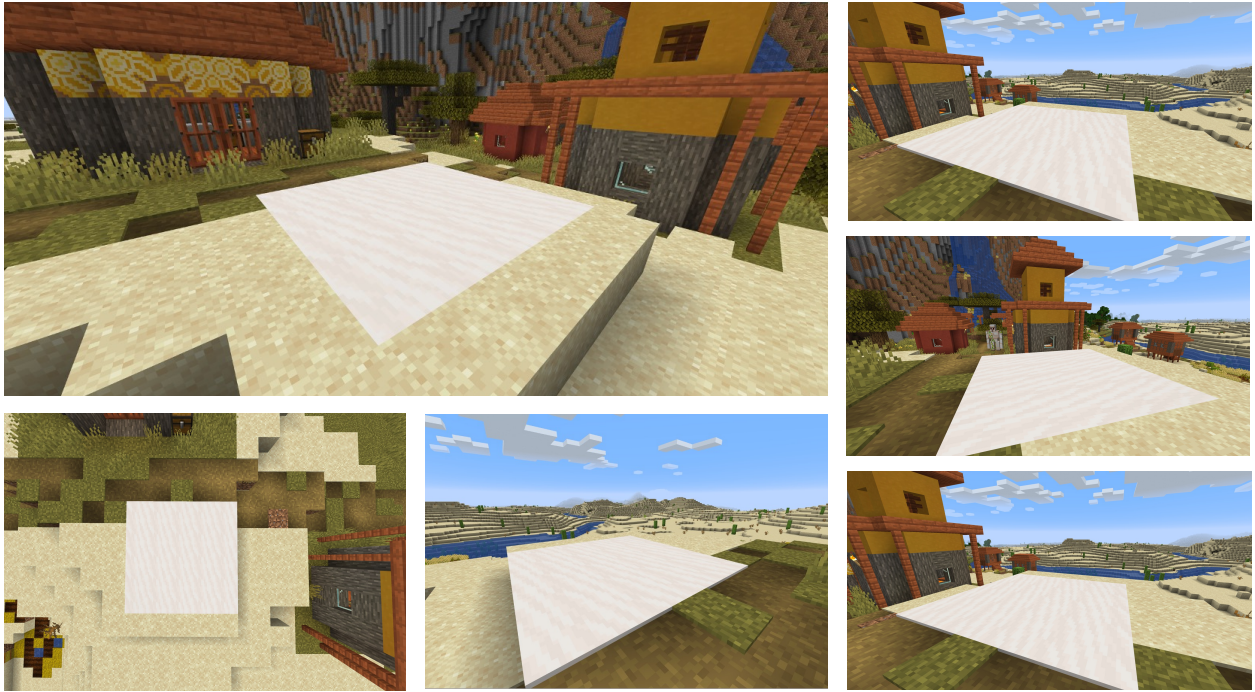


Figure D.8: An example scene in the Desert Village biome

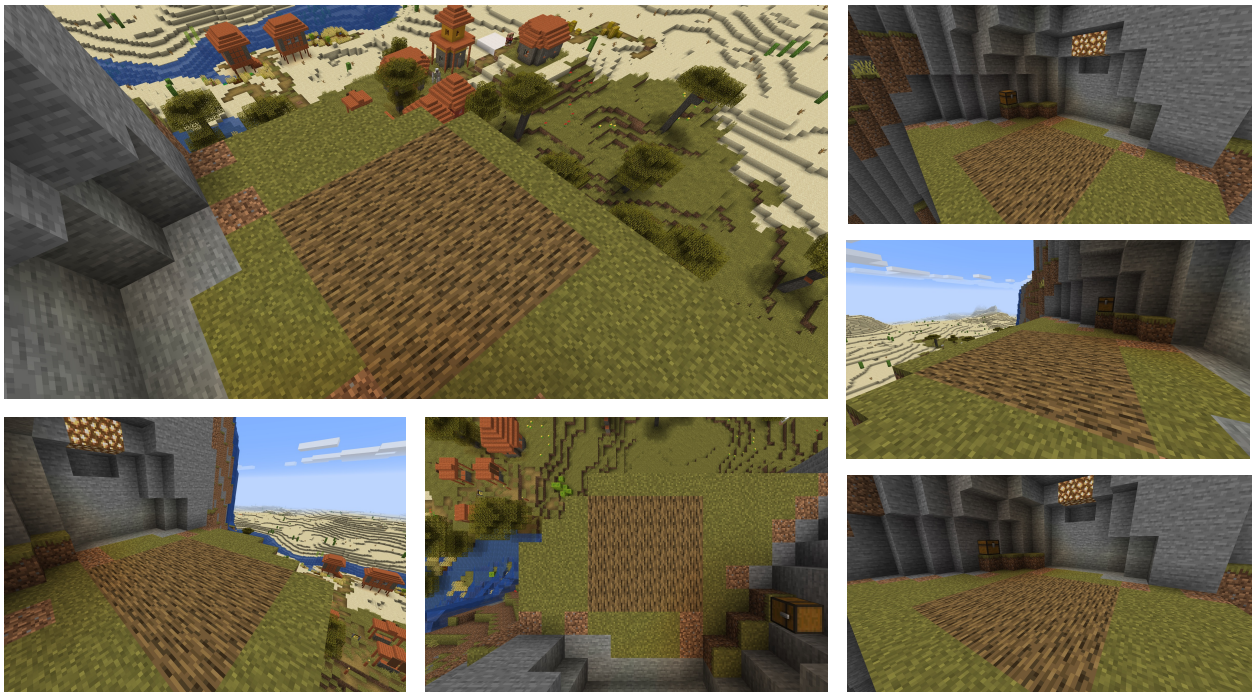


Figure D.9: An example scene in the Half Mountain biome



Figure D.10: An example scene in the Swamp biome

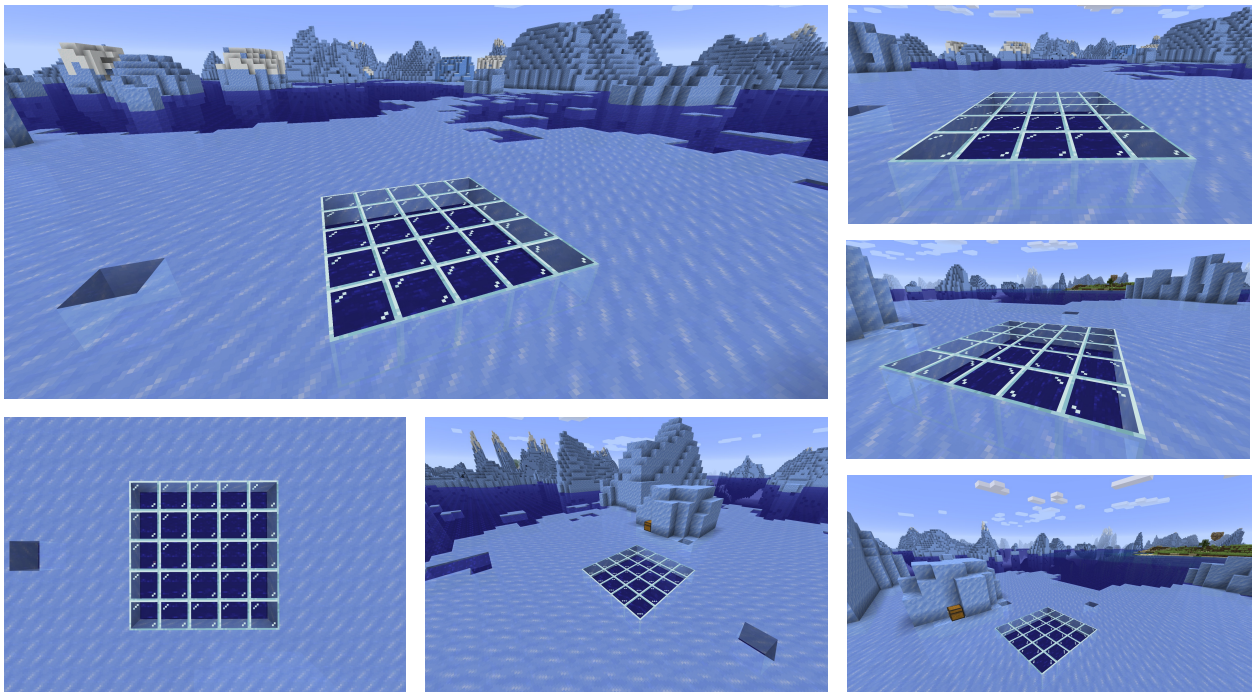


Figure D.11: An example scene in the Iceberg biome

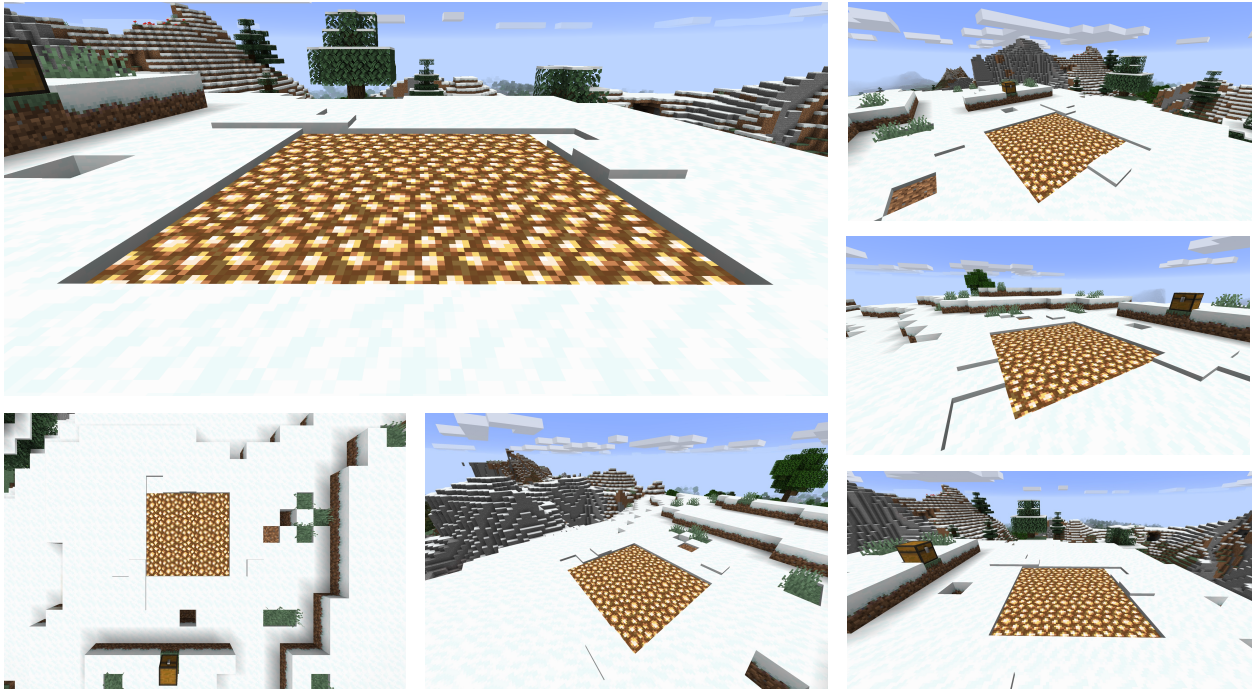


Figure D.12: An example scene in the Snow Mountain biome

D.4 Planner for Expert Demonstration

TeamCraft employed a planner to assign actions to each agent at every time step, utilizing perfect knowledge of the task including goal object positions, agents’ inventories, and each agent’s efficiency in performing actions. The planner optimizes actions using a cost function designed to minimize the total time to complete the task, reduce idle times for agents, minimize action dependencies to prevent agents from waiting on others, maximize parallelism of actions, assign tasks to the most efficient agents, and eliminate redundant or unnecessary actions. The cost function considers the following components:

Minimize Total Task Completion Time T : Denoted by $\min T$, our primary objective is to reduce the overall time required to complete the task, measured in time steps until the last agent finishes their final action.

Minimize Idle Actions for Each Agent E : Denoted by $\min \sum_{i=1}^N E_i$, we minimize the total empty actions, the sum of empty action E_i preformed by agent i .

Minimize Action Dependencies Across Agents D : Denoted by $\min D$, we minimize dependencies cause agents to wait for others to complete certain actions.

Minimize Redundant or Useless Actions U : Denoted by $\min U$, we minimize the total number of redundant or unnecessary actions performed by all agents.

Maximize Action Efficiency: Denoted by $\min \sum_{i=1}^N \sum_{j \in A_i} c_{ij}$, we assign actions to agents with higher capabilities to reduce the overall cost, where c_{ij} be the cost (inverse of efficiency) for agent i to perform action j .

We assign each component a weight:

$$C = w_1 T + w_2 \sum_{i=1}^N E_i + w_3 D + w_4 \sum_{i=1}^N \sum_{j \in A_i} c_{ij} + w_5 U \quad (\text{D.1})$$

where w_1, w_2, w_3, w_4, w_5 are weighting coefficients, and adjusted for each tasks.

Building: In the building task, where dependencies are moderate and parallelization is preferred, we place greater emphasis on minimizing idle actions by setting $w_2 = 1.4$ and assign a weight of 0.9 to the other components. This encourages agents to remain active and reduces idle time, enhancing overall efficiency.

Clearing: In the clearing task, using the correct tools can significantly speed up block removal (up to a threefold increase). Therefore, we assign a higher weight of $w_4 = 1.8$ to maximize action efficiency by assigning tasks to the most capable agents. The other weights are set to 0.8 to maintain overall performance while focusing on efficient tool usage.

Farming: Farming task is not heavily constrained by action dependencies, we assign equal weights of 1 to all components, ensuring a balanced consideration of time minimization, idle actions, action dependencies, action efficiency, and redundancy elimination.

Smelting: In the smelting task, which involves comparatively long and highly dependent action sequences, we prioritize minimizing action dependencies by setting $w_3 = 1.8$. The other weights are assigned a value of 0.8 to support this focus, facilitating smoother coordination among agents and reducing waiting times.


D.4.1 Example Expert Demonstrations

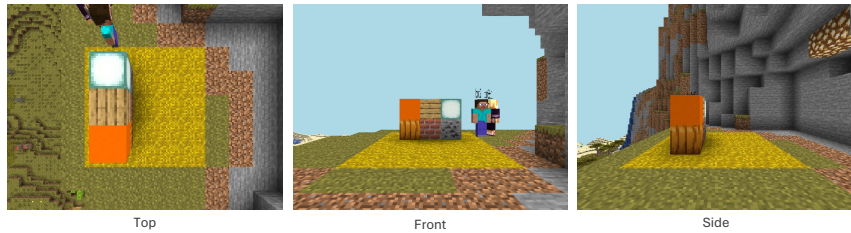
Figure D.13 and Figure D.14 show a classic example of the building task, which involves three agents building a 2x3 building on the mountain half. Each of the agents has some of the needed blocks in their inventory to build the building. For every time step after step 0, each of the three agents build one block, from bottom level to the second level.

Figure D.15 shows an example of the clearing task. Two agents are assigned to clean the blocks on a 6x6 platform. Each of them has a stone pickaxe in their inventory, which is the efficient tool to break "stone-like" blocks. In this case, they are able to break brick and

Building 01:

Bot1 Bot2 Bot3 build a building on the playground base  halfway up the mountain, following the blueprint given below.

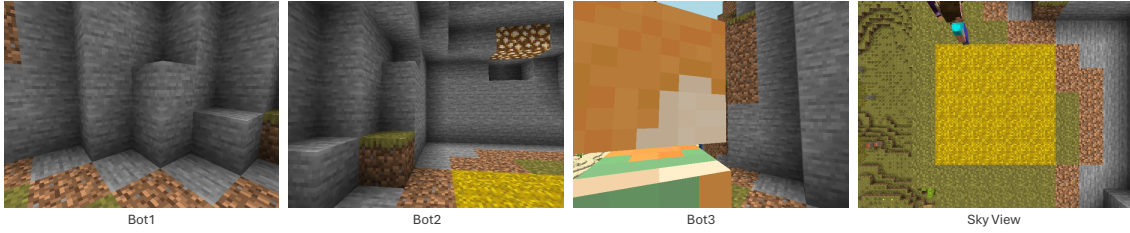
Bot1 has 5 , 2 , 1 . Bot2 has 3 , 2 , 2 . Bot3 has 3 , 4 . The blueprint looks like:



Step 0:

Actions: None

Reward: 0



Step 1:

Actions: `placeltem(bot1, 'pumpkin', new Vec3(-1,0,-1))`, `placeltem(bot2, 'coa_ore', new Vec3(1,0,-1))`, `placeltem(bot3, 'bricks', new Vec3(0,0,-1))`

Reward: 0.5

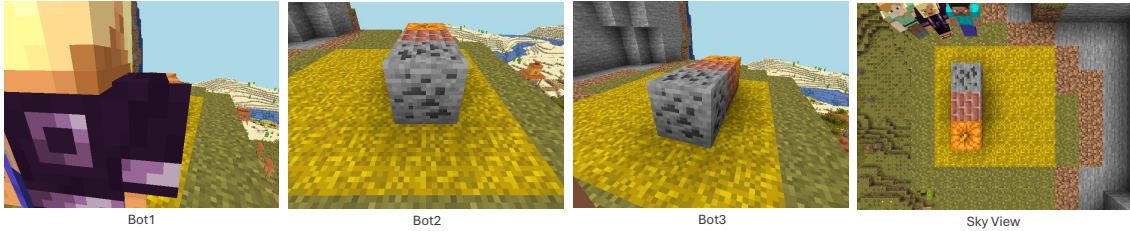


Figure D.13: An example demonstration in the Building task, Part I

Step 2:

Actions: `placeltem(bot1, 'sea_lantern', new Vec3(1,1,-1))`, `placeltem(bot2, 'orange_concrete', new Vec3(-1,1,-1))`,
`placeltem(bot3, 'oak_planks', new Vec3(0,1,-1))`

Reward: 1.0

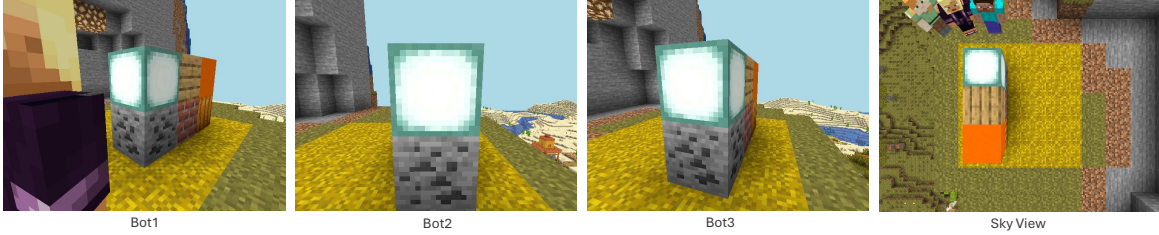
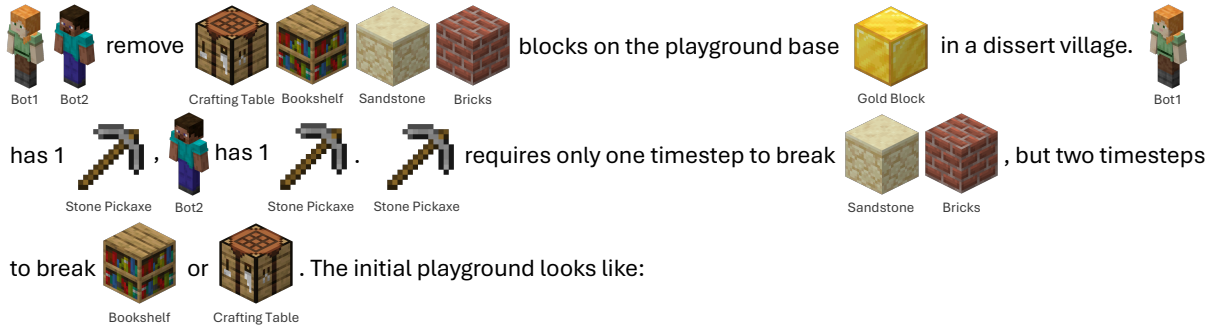


Figure D.14: An example demonstration in the Building task, Part II

Clearing 01:



Step 0:

Actions: None

Reward: 0

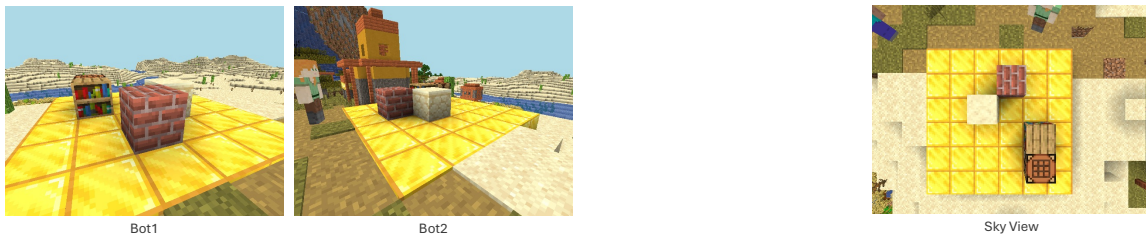
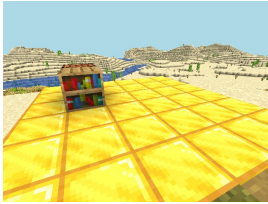


Figure D.15: An example demonstration in the Clearing task, Part I

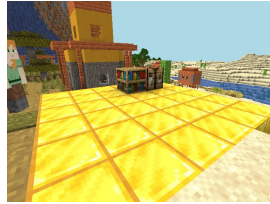
Step 1:

Actions: mineBlock(bot1, new Vec3(1,0,0)), mineBlock(bot2, new Vec3(0,0,-1))

Reward: 0.5



Bot1



Bot2



Sky View

Step 2:

Actions: mineBlock(bot1, new Vec3(-1,0,1)), mineBlock(bot2, new Vec3(-2,0,1))

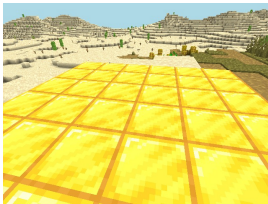
Reward: 0.5

Same visual observation as it requires two timesteps to break.

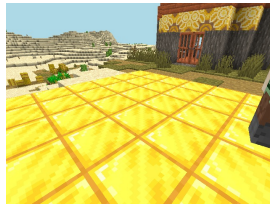
Step 3:

Actions: mineBlock(bot1, new Vec3(-1,0,1)), mineBlock(bot2, new Vec3(-2,0,1))

Reward: 1.0



Bot1



Bot2







Sky View

Figure D.16: An example demonstration in the Clearing task, Part II

Farming 01:



 sow and harvest for 2 more  on  on an iceberg. Some  are blocked by  .

Crops take up to three timesteps to grow from  to  to be harvestable. Each  gives 2  .

 has 2  , 3  .  has 1  , 1  . The initial playground looks like:



Figure D.17: An example demonstration in the Farming task, Part I

Step 0:

Actions: None

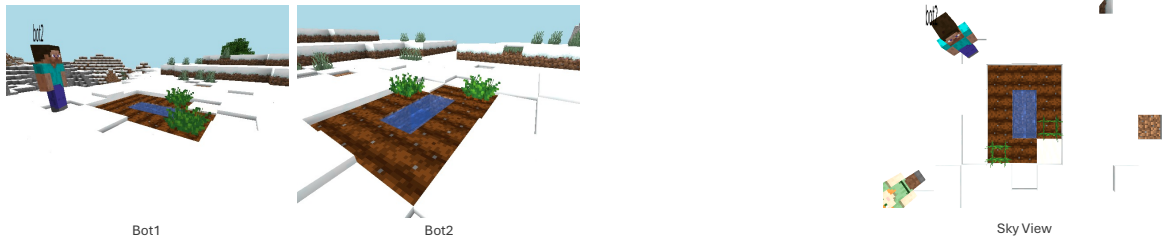
Reward: 0



Step 1:

Actions: farm_work(bot1, new Vec3(-2,-1,-1), 'sow', 'carrot'), farm_work(bot2, new Vec3(-1,-1,1), 'sow', 'carrot')

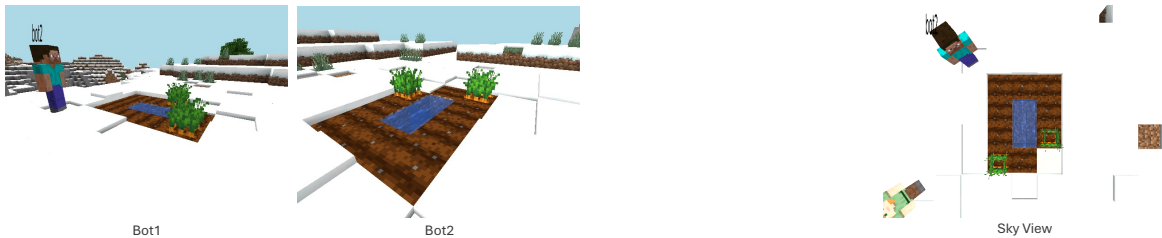
Reward: 0



Step 2:

Actions: None

Reward: 0.5



Step 3:

Actions: farm_work(bot1, new Vec3(-2,0,-1), 'harvest'), farm_work(bot2, new Vec3(-1,0,1), 'harvest')

Reward: 1.0

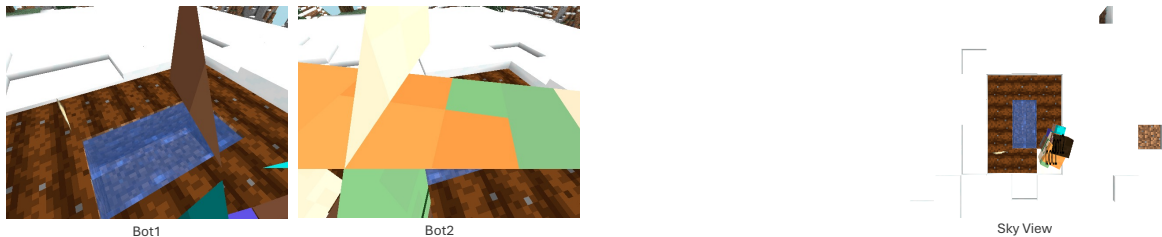


Figure D.18: An example demonstration in the Farming task, Part II

Smelting 01:



 cook for 2  in a dessert village. To get  , smelt  with  as fuel in  .

Resources are on base  or in inventory. To get  , kill  with  . To get  , collect with

 .  has 1  .  has 1  , 1  , and  . The initial playground looks like:



Top



Front



Side

Step 0:

Actions: None

Reward: 0



Step 1:

Actions: killMob(bot1, new Vec3(2,0,-3)), putFuelFurnace(bot2, 'birch_log', new Vec3(0,0,0))

Reward: 0



Figure D.19: An example demonstration in the Smelting task, Part I

Step 2:

Actions: putItemFurnace(bot1, 'porkchop', new Vec3(0,0,0)), putItemFurnace(bot2, 'porkchop', new Vec3(0,0,0))

Reward: 0.5



Step 3:

Actions: obtainBlock(bot1, new Vec3(-3,0,-3))

Reward: 1.0



Bot1



Bot2



Sky View

Step 4:

Actions: putFuelFurnace(bot1, 'birch_log', new Vec3(0,0,0))

Reward: 1.0



Bot1



Bot2



Sky View

Step 5:

Actions: takeOutFurnace(bot1, new Vec3(0,0,0))

Reward: 1.0

Same visual observation as step 4.

Figure D.20: An example demonstration in the Smelting task, Part II

Table D.1: Comparison of TeamCraft-VLA redundancy rates

	Test	Goal	Scene	Agents	Average
TeamCraft-VLA-7B-Cen	0.01	0.02	0.01	0.01	0.01
TeamCraft-VLA-13B-Cen	0.01	0.00	0.01	0.01	0.01
TeamCraft-VLA-7B-Dec	0.13	0.12	0.13	0.24	0.15
TeamCraft-VLA-13B-Dec	0.11	0.11	0.12	0.22	0.14

Table D.2: Comparison of TeamCraft-VLA action sequence length

	Test	Goal	Scene	Agents	Average
TeamCraft-VLA-7B-Cen	6.62	7.63	5.93	6.35	6.63
TeamCraft-VLA-13B-Cen	6.25	7.44	6.46	6.47	6.65
TeamCraft-VLA-7B-Dec	8.42	8.53	8.06	7.38	8.1
TeamCraft-VLA-13B-Dec	8.62	8.46	8.41	6.71	8.04

sandstone in just one time step with pickaxe but requires two time step to break "wood-made" blocks like bookshelf and crafting table. This resulted time step 2 and 3 has exactly same visual observation, shown in [Figure D.16](#).

[Figure D.17](#) and [Figure D.18](#) shows an example of two agents farming on a snow mountain for two extra carrots. In step 1, agent1 and agent2 both sow the carrots on the open ground. In step 2 they saw that the carrots are ready to collect and they both collect one carrot in step 3 and eventually they collected two carrots.

[Figure D.19](#) and [Figure D.20](#) shows an example of smelting task where two agents need to get two cooked porkchops. In step 1, one agent is in charge of adding the fuel to the furnace and the other agent tries to kill the pork to get the raw porkchop. Since bot2 already has one porkchop, it only requires one additional porkchop. In step 2, both agents put the porkchop to the furnace and in step 3, they got 2 cooked porkcops.

D.5 Grid-World Settings

Under the grid-world setting, we replace the three orthographic view images and first person view images with text descriptions of the task goal and current environment states, and provide them as input to the model. Here we show one example of the prompt construction

Table D.3: Action space within *TeamCraft*

Type	Arguments	Description
placeItem	BotID, ItemType, Location	BotID places an item of ItemType at the specified 3D Location.
mineBlock	BotID, Location	BotID mines a block at the specified 3D Location.
farmWork	BotID, Location, Action, ItemType	BotID performs an Action (sow or harvest) on ItemType at the specified 3D Location.
obtainBlock	BotID, Location	BotID obtains a block from the specified 3D Location.
putFuelFurnace	BotID, ItemType, Location	BotID places an ItemType as fuel into a furnace at the specified 3D Location.
putItemFurnace	BotID, ItemType, Location	BotID inserts an ItemType into a furnace at the specified 3D Location.
takeOutFurnace	BotID, ItemType, Location	BotID removes an ItemType from a furnace at the specified 3D Location.
killMob	BotID, Location	BotID engages and eliminates a mob at the specified 3D Location.

System Prompt

Three bots need to build a building on the platform. Target building is: Put sea_lantern on [0 ,1 ,0]. Put oak_fence on [-1 ,1 ,0]. Put sponge on [0 ,1 ,-1]. Put emerald_block on [-1 ,1 ,-1]. Put dirt on [0 ,0 ,0]. Put bricks on [-1 ,0 ,0]. Put emerald_block on [0 ,0 ,-1]. Put clay on [-1 ,0 ,-1].

User Prompt

bot1 has 4 dirt. bot1 has 3 clay. bot1 has 7 emerald_block. bot1 has 1 oak_fence. bot1 has 3 sponge. bot1 has 1 bricks. bot1 has 3 sea_lantern. bot2 has 4 bricks. bot2 has 2 sponge. bot2 has 6 sea_lantern. bot2 has 2 oak_fence. bot2 has 4 emerald_block. bot2 has 1 dirt. bot2 has 3 clay. bot3 has 6 emerald_block. bot3 has 4 oak_fence. bot3 has 2 dirt. bot3 has 2 sponge. bot3 has 3 clay. bot3 has 2 sea_lantern. bricks is on [-1 ,0 ,0]. dirt is on [0 ,0 ,0]. Write the actions for bot1, bot2 and bot3 based on this given observation.

Figure D.21: Prompt example for Building task under the grid-world setting

in each task.

Building: As shown in Figure D.21, the system prompt consists of both task description and the target building coordination of each block. The user prompt consists of the built blocks and the inventories of the agents.

System Prompt

Three bots need to break everything on the platform. clay is on [-2 ,0 ,-2]. birch_log is on [-2 ,0 ,0]. dirt is on [-1 ,0 ,-2]. crafting_table is on [-1 ,0 ,1]. anvil is on [-1 ,1 ,1]. anvil is on [0 ,0 ,-2]. iron_ore is on [0 ,0 ,1]. cobweb is on [1 ,0 ,1].

User Prompt

bot1 has 1 stone_pickaxe. bot1 has 1 anvil. bot2 has 1 stone_axe. bot2 has 1 crafting_table. bot3 has 1 stone_pickaxe. bot3 has 1 dirt. clay is on [-2 ,0 ,-2]. birch_log is on [-2 ,0 ,0]. iron_ore is on [0 ,0 ,1]. cobweb is on [1 ,0 ,1]. Write the actions for bot1, bot2 and bot3 based on this given observation.

Figure D.22: Prompt example for Clearing task under the grid-world setting

System Prompt

Two bots need to grow on the platform. The goal is to get 5 carrot. farmland is on [-3 ,-1 ,-2] with value of 7. cyan_concrete is on [-3 ,-1 ,-1]. water is on [-3 ,-1 ,0]. cyan_concrete is on [-3 ,-1 ,1]. cyan_concrete is on [-3 ,-1 ,2]. farmland is on [-2 ,-1 ,-2] with value of 7. cyan_concrete is on [-2 ,-1 ,-1]. water is on [-2 ,-1 ,0]. farmland is on [-2 ,-1 ,1] with value of 7. cyan_concrete is on [-2 ,-1 ,2]. cyan_concrete is on [-1 ,-1 ,-2]. cyan_concrete is on [-1 ,-1 ,-1]. water is on [-1 ,-1 ,0]. farmland is on [-1 ,-1 ,1] with value of 7. farmland is on [-1 ,-1 ,2] with value of 7. cyan_concrete is on [0 ,-1 ,-2]. farmland is on [0 ,-1 ,-1] with value of 7. water is on [0 ,-1 ,0]. cyan_concrete is on [0 ,-1 ,1]. cyan_concrete is on [0 ,-1 ,2]. cyan_concrete is on [1 ,-1 ,-2]. cyan_concrete is on [1 ,-1 ,-1]. water is on [1 ,-1 ,0]. farmland is on [1 ,-1 ,1] with value of 7. cyan_concrete is on [1 ,-1 ,2]. cyan_concrete is on [2 ,-1 ,-2]. cyan_concrete is on [2 ,-1 ,-1]. water is on [2 ,-1 ,0]. cyan_concrete is on [2 ,-1 ,1]. farmland is on [2 ,-1 ,2] with value of 7. cyan_concrete is on [3 ,-1 ,-2]. farmland is on [3 ,-1 ,-1] with value of 7. water is on [3 ,-1 ,0]. cyan_concrete is on [3 ,-1 ,1]. farmland is on [3 ,-1 ,2] with value of 7.

User Prompt

bot1 has 5 carrot. bot1 has 2 beetroot. bot1 has 3 potato. bot2 has 2 carrot. bot2 has 2 beetroot. bot2 has 2 wheat_seeds. farmland is on [-3 ,-1 ,-2] with value of 7. cyan_concrete is on [-3 ,-1 ,-1]. water is on [-3 ,-1 ,0]. cyan_concrete is on [-3 ,-1 ,1]. cyan_concrete is on [-3 ,-1 ,2]. farmland is on [-2 ,-1 ,-2] with value of 7. cyan_concrete is on [-2 ,-1 ,-1]. water is on [-2 ,-1 ,0]. farmland is on [-2 ,-1 ,1] with value of 7. cyan_concrete is on [-2 ,-1 ,2]. cyan_concrete is on [-1 ,-1 ,-2]. cyan_concrete is on [-1 ,-1 ,-1]. water is on [-1 ,-1 ,0]. farmland is on [-1 ,-1 ,1] with value of 7. farmland is on [-1 ,-1 ,2] with value of 7. cyan_concrete is on [0 ,-1 ,-2]. farmland is on [0 ,-1 ,-1] with value of 7. water is on [0 ,-1 ,0]. cyan_concrete is on [0 ,-1 ,1]. cyan_concrete is on [0 ,-1 ,2]. cyan_concrete is on [1 ,-1 ,-2]. cyan_concrete is on [1 ,-1 ,-1]. water is on [1 ,-1 ,0]. farmland is on [1 ,-1 ,1] with value of 7. cyan_concrete is on [1 ,-1 ,2]. cyan_concrete is on [2 ,-1 ,-2]. cyan_concrete is on [2 ,-1 ,-1]. water is on [2 ,-1 ,0]. cyan_concrete is on [2 ,-1 ,1]. farmland is on [2 ,-1 ,2] with value of 7. cyan_concrete is on [3 ,-1 ,-2]. farmland is on [3 ,-1 ,-1] with value of 7. water is on [3 ,-1 ,0]. cyan_concrete is on [3 ,-1 ,1]. farmland is on [3 ,-1 ,2] with value of 7. carrots is on [3 ,0 ,-1] with value of 0. carrots is on [3 ,0 ,2] with value of 0. Write the actions for bot1, bot2 based on this given observation.

Figure D.23: Prompt example for Farming task under the grid-world setting

System Prompt

Two bots need to craft 2 stone. Here are the instructions: Cooking Food: 1. To cook a 'cooked_beef'... cobblestone is on [-2 ,0 ,2]. furnace is on [0 ,0 ,1]. spruce_planks is on [2 ,0 ,-3]. cobblestone is on [2 ,0 ,-1].

User Prompt

bot1 has 1 iron_sword. bot1 has 1 iron_shovel. bot1 has 1 iron_pickaxe. bot1 has 1 cobblestone. bot1 has 1 spruce_planks. bot2 has 1 spruce_planks. bot2 has 1 iron_shovel. bot2 has 2 iron_pickaxe. cobblestone is on [-2 ,0 ,2]. furnace is on [0 ,0 ,1]. spruce_planks is on [2 ,0 ,-3]. cobblestone is on [2 ,0 ,-1]. Write the actions for bot1, bot2 based on this given observation.

Figure D.24: Prompt example for Smelting task under the grid-world setting

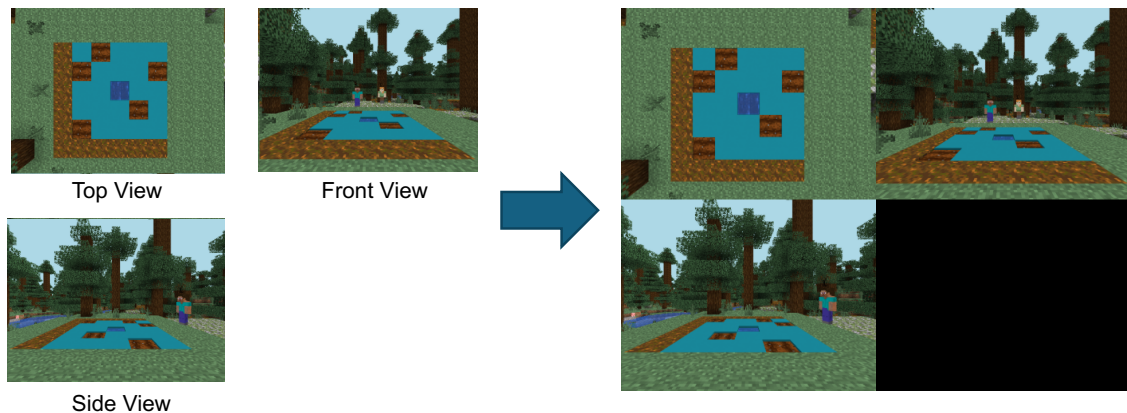


Figure D.25: Combining three orthogonal view images into a single composite image

Table D.4: Hyperparameters for TeamCraft-VLA

lr	model max length	vision tower	patch size	resolution	language model	optimizer	lr scheduler type	warmup ratio
2e-5	4096	openai-clip-vit-large	14	336*336	Vicuna-v1.5	AdamW	constant_with_warmup	0.03

Clearing: As shown in Figure D.22, the system prompt consists of both task description and the blocks that appeared on the platform initially. The user prompt consists of the blocks that appeared on the platform at current time step and the inventories of the agents.

Farming: As shown in Figure D.23, the system prompt consists of both task description and the blocks in the farmland. The user prompt consists of the blocks in the farmland and crops information at current time step and the inventories of the agents.

Smelting: As shown in Figure D.24, the system prompt consists of both task description, instructions to craft different items and the blocks in the field. The user prompt consists of the blocks locations at current time step and the inventories of the agents.

D.6 TeamCraft-VLA Implementation Details

We use Vicuna-v1.5 as the LLM backbone. For the visual encoder, we employ CLIP ViT-L/14 to process all input images, including three orthogonal views and the first-person view of the agents. The image embeddings are then projected into the LLM space with a linear

projection layer and concatenated with the text embeddings. The combined embeddings are fed into the LLM, which outputs the final action. During training, we froze the visual encoder and projector and only finetune the LLM. All image embeddings are positioned before the text embeddings, separated by "image start" and "image end" tokens. In centralized settings, where the number of images varies depending on the number of agents, we pad a dummy image at the end for training stability if the task involves only two agents. In decentralized settings, the number of image inputs remains unaffected, as the model processes only the first-person view of the current agent, excluding views from others.

We train each model for 3 epochs using the training split, leveraging 8 A100 GPUs with a global batch size of 16. In the centralized setting, training the 7B model takes 36 hours, while the 13B model requires 72 hours. In the decentralized setting, the training duration doubles, with the 7B model requiring 72 hours and the 13B model taking 144 hours. In the grid-world setting, training the 7B model takes 20 hours.

D.6.1 Arrangement of Three Orthogonal Views

For training and evaluation, we combine the three orthogonal view images into a single composite image by arranging them to the upper-left top-left corner, top-right corner, and the lower-left corner of the composite image. An example of this arrangement is shown below [Figure D.25](#). This process is to reduce the number of images provided to the model to conform with the 4096 context length limit.

D.6.2 Hyperparameters

We present the hyperparameters for VLA training in [Table D.4](#).

D.6.3 Model Output Parsing

The output of the model is a string which will be parsed into the pre-defined high level skills. The string will be first processed by removing special sentence begin token, `<s>`, and ending token `</s>`. It will then be split into a list, where each item is parsed as the skill of one agent.

Table D.5: Task success rates and subgoal success rates of the TeamCraft-VLA-7B-Cen and TeamCraft-VLA-7B-Dec models. Subgoal success rates are given in parentheses.

Tasks	Condition	Centralized			Decentralized		
		10%	50%	100%	10%	50%	100%
Building	Test	0.00 (0.12)	0.38 (0.76)	0.42 (0.81)	0.00 (0.18)	0.00 (0.28)	0.00 (0.38)
	Shape	0.00 (0.12)	0.20 (0.67)	0.30 (0.75)	0.00 (0.15)	0.00 (0.25)	0.00 (0.40)
	Material	0.00 (0.13)	0.18 (0.64)	0.30 (0.74)	0.00 (0.13)	0.00 (0.20)	0.00 (0.34)
	Scene	0.00 (0.15)	0.36 (0.73)	0.40 (0.83)	0.00 (0.16)	0.00 (0.21)	0.00 (0.36)
	Agents	0.00 (0.18)	0.02 (0.50)	0.02 (0.57)	0.00 (0.12)	0.00 (0.20)	0.00 (0.14)
Clearing	Test	0.00 (0.13)	0.08 (0.43)	0.64 (0.91)	0.00 (0.45)	0.02 (0.35)	0.20 (0.68)
	Shape	0.00 (0.09)	0.08 (0.34)	0.56 (0.91)	0.00 (0.47)	0.02 (0.27)	0.16 (0.74)
	Material	0.00 (0.10)	0.12 (0.45)	0.56 (0.90)	0.00 (0.48)	0.00 (0.22)	0.16 (0.67)
	Scene	0.00 (0.11)	0.10 (0.44)	0.58 (0.92)	0.00 (0.41)	0.04 (0.37)	0.10 (0.64)
	Agents	0.00 (0.16)	0.14 (0.64)	0.36 (0.81)	0.02 (0.50)	0.02 (0.54)	0.12 (0.60)
Farming	Test	0.14 (0.43)	0.34 (0.60)	0.36 (0.63)	0.02 (0.07)	0.02 (0.14)	0.00 (0.09)
	Crop	0.00 (0.00)	0.00 (0.00)	0.00 (0.00)	0.00 (0.00)	0.00 (0.00)	0.00 (0.00)
	Scene	0.16 (0.39)	0.34 (0.65)	0.38 (0.67)	0.00 (0.05)	0.00 (0.11)	0.02 (0.07)
	Agents	0.02 (0.18)	0.18 (0.61)	0.38 (0.68)	0.00 (0.08)	0.00 (0.11)	0.04 (0.27)
Smelting	Test	0.06 (0.17)	0.20 (0.36)	0.24 (0.28)	0.08 (0.13)	0.08 (0.09)	0.16 (0.29)
	Goal	0.08 (0.21)	0.04 (0.07)	0.00 (0.00)	0.08 (0.17)	0.00 (0.00)	0.00 (0.00)
	Furnace	0.10 (0.28)	0.10 (0.20)	0.18 (0.20)	0.06 (0.07)	0.06 (0.06)	0.06 (0.16)
	Scene	0.08 (0.19)	0.14 (0.28)	0.18 (0.23)	0.08 (0.19)	0.14 (0.19)	0.12 (0.28)
	Agents	0.00 (0.15)	0.02 (0.24)	0.06 (0.13)	0.04 (0.05)	0.00 (0.02)	0.02 (0.28)

D.7 Additional Results of TeamCraft-VLA

D.7.1 Task Success Rate and Subgoal Success Rate

We show task success rate and subgoal success rate of centralized and decentralized 7B models with different data scales in Table D.5, and those of 13B models in Table D.6. We compare among different centralized models in Table D.7.

D.7.2 Redundancy Rate

We present a more detailed analysis of redundancy rates, including the 13B models, in Table D.1. Both the 7B and 13B models exhibit redundancy issues in decentralized settings. Increasing model size alone does not resolve the redundancy problem in such scenarios.

Table D.6: Task success rates and subgoal success rates of the TeamCraft-VLA-13B-Cen and TeamCraft-VLA-13B-Dec models. Subgoal success rates are given in parentheses.

Tasks	Condition	Centralized			Decentralized		
		10%	50%	100%	10%	50%	100%
Building	Test	0.00 (0.18)	0.46 (0.80)	0.48 (0.79)	0.00 (0.13)	0.00 (0.18)	0.00 (0.31)
	Shape	0.00 (0.16)	0.30 (0.73)	0.26 (0.69)	0.00 (0.15)	0.00 (0.15)	0.00 (0.32)
	Material	0.00 (0.15)	0.24 (0.65)	0.08 (0.63)	0.00 (0.14)	0.00 (0.14)	0.00 (0.31)
	Scene	0.00 (0.16)	0.38 (0.75)	0.48 (0.83)	0.00 (0.17)	0.00 (0.17)	0.00 (0.28)
	Agents	0.00 (0.16)	0.00 (0.49)	0.04 (0.59)	0.00 (0.14)	0.00 (0.16)	0.00 (0.23)
Clearing	Test	0.04 (0.37)	0.42 (0.83)	0.64 (0.94)	0.00 (0.46)	0.02 (0.62)	0.02 (0.60)
	Shape	0.00 (0.26)	0.42 (0.85)	0.78 (0.96)	0.00 (0.47)	0.00 (0.57)	0.04 (0.58)
	Material	0.04 (0.36)	0.36 (0.83)	0.56 (0.92)	0.02 (0.53)	0.00 (0.60)	0.02 (0.58)
	Scene	0.06 (0.35)	0.44 (0.88)	0.48 (0.90)	0.00 (0.55)	0.02 (0.59)	0.08 (0.64)
	Agents	0.02 (0.55)	0.16 (0.65)	0.16 (0.77)	0.02 (0.50)	0.02 (0.52)	0.02 (0.50)
Farming	Test	0.4 (0.72)	0.62 (0.79)	0.46 (0.73)	0.08 (0.39)	0.04 (0.23)	0.02 (0.33)
	Crop	0.00 (0.00)	0.00 (0.00)	0.00 (0.00)	0.00 (0.00)	0.00 (0.00)	0.00 (0.00)
	Scene	0.30 (0.69)	0.52 (0.76)	0.44 (0.75)	0.04 (0.32)	0.06 (0.29)	0.10 (0.33)
	Agents	0.12 (0.54)	0.44 (0.79)	0.36 (0.72)	0.02 (0.22)	0.00 (0.19)	0.02 (0.23)
Smelting	Test	0.06 (0.08)	0.22 (0.44)	0.32 (0.59)	0.10 (0.25)	0.06 (0.09)	0.10 (0.19)
	Goal	0.00 (0.00)	0.00 (0.00)	0.00 (0.00)	0.00 (0.10)	0.00 (0.00)	0.00 (0.00)
	Furnace	0.06 (0.08)	0.20 (0.40)	0.18 (0.38)	0.06 (0.12)	0.04 (0.08)	0.04 (0.07)
	Scene	0.04 (0.08)	0.16 (0.43)	0.24 (0.56)	0.12 (0.28)	0.04 (0.09)	0.08 (0.18)
	Agents	0.00 (0.03)	0.00 (0.26)	0.04 (0.37)	0.00 (0.02)	0.00 (0.01)	0.00 (0.00)

Table D.7: Task success rates and subgoal success rates of various centralized models. Subgoal success rates are given in parentheses. All models are trained with the full training data except GPT-4o.

Tasks	Condition	TeamCraft-VLA-7B	TeamCraft-VLA-13B	GPT-4o	TeamCraft-7B-GridWorld
Building	Test	0.42 (0.81)	0.48 (0.79)	0.00 (0.07)	0.42 (0.88)
	Shape	0.30 (0.75)	0.26 (0.69)	0.00 (0.08)	0.50 (0.90)
	Material	0.30 (0.74)	0.08 (0.63)	0.00 (0.07)	0.26 (0.82)
	Scene	0.40 (0.83)	0.48 (0.83)	0.00 (0.07)	0.48 (0.89)
	Agents	0.02 (0.57)	0.04 (0.59)	0.00 (0.00)	0.12 (0.71)
Clearing	Test	0.64 (0.91)	0.64 (0.94)	0.00 (0.03)	1.00 (1.00)
	Shape	0.56 (0.91)	0.78 (0.96)	0.00 (0.04)	1.00 (1.00)
	Material	0.56 (0.91)	0.56 (0.92)	0.00 (0.12)	1.00 (1.00)
	Scene	0.58 (0.92)	0.48 (0.90)	0.00 (0.06)	1.00 (1.00)
	Agents	0.36 (0.81)	0.16 (0.77)	0.00 (0.00)	0.84 (0.97)
Farming	Test	0.36 (0.64)	0.46 (0.73)	0.00 (0.00)	0.78 (0.86)
	Crop	0.00 (0.00)	0.00 (0.00)	0.00 (0.00)	0.00 (0.00)
	Scene	0.38 (0.67)	0.44 (0.75)	0.00 (0.00)	0.90 (0.96)
	Agents	0.38 (0.68)	0.36 (0.72)	0.00 (0.00)	0.40 (0.73)
Smelting	Test	0.24 (0.28)	0.32 (0.59)	0.02 (0.02)	0.24 (0.51)
	Goal	0.00 (0.00)	0.00 (0.00)	0.08 (0.08)	0.00 (0.00)
	Furnace	0.18 (0.20)	0.18 (0.38)	0.00 (0.00)	0.24 (0.39)
	Scene	0.18 (0.23)	0.24 (0.56)	0.00 (0.00)	0.36 (0.58)
	Agents	0.06 (0.13)	0.04 (0.37)	0.00 (0.00)	0.00 (0.31)

D.7.3 Action Sequence Length

We compared the average action lengths across different splits between the 7B and 13B models under both centralized and decentralized settings, as shown in [Table D.2](#). In general, decentralized settings require longer action sequences to complete tasks. Among the splits, the *Goal* split is the most challenging, as it demands more actions to accomplish the tasks.

REFERENCES

- Agarwal, A., Kumar, S., and Sycara, K. (2019). Learning transferable cooperative behavior in multi-agent teams. *5*, 15
- Albrecht, S. V. and Stone, P. (2019). Reasoning about hypothetical agent behaviours and their parameters. *arXiv preprint arXiv:1906.11064*. 18
- Amresh, A., Cooke, N., and Fouse, A. (2023). A Minecraft based simulated task environment for human AI teaming. In *Proceedings of the 23rd ACM International Conference on Intelligent Virtual Agents*, pages 1–3. 78
- Anderson, P., Wu, Q., Teney, D., Bruce, J., Johnson, M., Sünderhauf, N., Reid, I., Gould, S., and Van Den Hengel, A. (2018). Vision-and-language navigation: Interpreting visually-grounded navigation instructions in real environments. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3674–3683. 20
- Bäck, T. and Schwefel, H.-P. (1993). An overview of evolutionary algorithms for parameter optimization. *Evolutionary Computation*, 1(1):1–23. 15
- Baker, B., Akkaya, I., Zhokov, P., Huizinga, J., Tang, J., Ecoffet, A., Houghton, B., Sampedro, R., and Clune, J. (2022). Video pretraining (vpt): Learning to act by watching unlabeled online videos. *Advances in Neural Information Processing Systems*, 35:24639–24654. 77
- Baker, B., Kanitscheider, I., Markov, T., Wu, Y., Powell, G., McGrew, B., and Mordatch, I. (2019). Emergent tool use from multi-agent autotutorials. *arXiv preprint arXiv:1909.07528*. 14
- Baker, C. L., Saxe, R., and Tenenbaum, J. B. (2009). Action understanding as inverse planning. *Cognition*, 113(3):329–349. 8
- Bansal, T., Pachocki, J., Sidor, S., Sutskever, I., and Mordatch, I. (2018). Emergent complexity via multi-agent competition. In *International Conference on Learning Representations*. 3
- Barrett, S., Agmon, N., Hazon, N., Kraus, S., and Stone, P. (2014). Communicating with unknown teammates. In *ECAI*, pages 45–50. 18
- Barrett, S., Rosenfeld, A., Kraus, S., and Stone, P. (2017). Making friends on the fly: Cooperating with new teammates. *Artificial Intelligence*, 242:132–171. 18
- Bengio, Y., Louradour, J., Collobert, R., and Weston, J. (2009). Curriculum learning. In *International Conference on Machine Learning*, pages 41–48. ACM. 14
- Bratman, M. (1987). *Intention, plans, and practical reason*. Harvard University Press, Cambridge, Mass. 8, 17

- Cai, S., Wang, Z., Lian, K., Mu, Z., Ma, X., Liu, A., and Liang, Y. (2024). ROCKET-1: Master open-world interaction with visual-temporal context prompting. *arXiv preprint arXiv:2410.17856*. 11
- Carroll, M., Shah, R., Ho, M. K., Griffiths, T. L., Seshia, S. A., Abbeel, P., and Dragan, A. (2020). On the utility of learning about humans for human-AI coordination. 19, 20, 78
- Chang, M., Chhablani, G., Clegg, A., Cote, M. D., Desai, R., Hlavac, M., Karashchuk, V., Krantz, J., Mottaghi, R., Parashar, P., et al. (2024). PARTNR: A benchmark for planning and reasoning in embodied multi-agent tasks. *arXiv preprint arXiv:2411.00081*. 19, 20
- Chen, B., Song, S., Lipson, H., and Vondrick, C. (2020). Visual hide and seek. In *Artificial Life Conference Proceedings 32*, pages 645–655. MIT Press One Rogers Street, Cambridge, MA 02142-1209, USA journals-info 10, 11
- Chen, L., Hu, B., Guan, Z.-H., Zhao, L., and Shen, X. (2021). Multiagent meta-reinforcement learning for adaptive multipath routing optimization. *IEEE Transactions on Neural Networks and Learning Systems*, 33(10):5374–5386. 5
- Chen, X., Treiber, M., Kanagaraj, V., and Li, H. (2018). Social force models for pedestrian traffic: State of the art. *Transport Reviews*, 38(5):625–653. 16
- Cheng, S., Zhao, M., Tang, N., Zhao, Y., Zhou, J., Shen, M., and Gao, T. (2023). Intention beyond desire: Spontaneous intentional commitment regulates conflicting desires. *Cognition*, 238:105513. 19
- Ci, H., Wu, M., Zhu, W., Ma, X., Dong, H., Zhong, F., and Wang, Y. (2022). GFPose: Learning 3d human pose prior with gradient fields. *arXiv preprint arXiv:2212.08641*. 17
- Conti, E., Madhavan, V., Such, F. P., Lehman, J., Stanley, K., and Clune, J. (2018). Improving exploration in evolution strategies for deep reinforcement learning via a population of novelty-seeking agents. In *Advances in Neural Information Processing Systems*, pages 5027–5038. 15
- Czarnecki, W. M., Jayakumar, S. M., Jaderberg, M., Hasenclever, L., Teh, Y. W., Heess, N., Osindero, S., and Pascanu, R. (2018). Mix & match agent curricula for reinforcement learning. In *International Conference on Machine Learning*, pages 1095–1103. 15
- Das, A., Datta, S., Gkioxari, G., Lee, S., Parikh, D., and Batra, D. (2018). Embodied question answering. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–10. 20
- Das, A., Gervet, T., Romoff, J., Batra, D., Parikh, D., Rabbat, M., and Pineau, J. (2019). TarMAC: Targeted multi-agent communication. In *International Conference on Machine Learning*, pages 1538–1546. PMLR. 11
- Dong, Y., Zhu, X., Pan, Z., Zhu, L., and Yang, Y. (2024). VillagerAgent: A graph-based multi-agent framework for coordinating complex task dependencies in Minecraft. In Ku, L.-W., Martins, A., and Srikumar, V., editors, *Findings of the Association for*

- Computational Linguistics: ACL 2024*, pages 16290–16314, Bangkok, Thailand. Association for Computational Linguistics. 10, 21
- Duan, Y., Andrychowicz, M., Stadie, B., Ho, J., Schneider, J., Sutskever, I., Abbeel, P., and Zaremba, W. (2017). One-shot imitation learning. In *Advances in Neural Information Processing Systems*, pages 1087–1098. 14
- Elman, J. L. (1993). Learning and development in neural networks: The importance of starting small. *Cognition*, 48(1):71–99. 14
- Fan, L., Wang, G., Jiang, Y., Mandlkar, 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. 20, 21, 77
- Florensa, C., Held, D., Wulfmeier, M., Zhang, M., and Abbeel, P. (2017). Reverse curriculum generation for reinforcement learning. In *CoRL*, pages 482–495. 14
- Foerster, J., Assael, I. A., de Freitas, N., and Whiteson, S. (2016). Learning to communicate with deep multi-agent reinforcement learning. In *Advances in Neural Information Processing Systems*, pages 2137–2145. 3, 13
- Foerster, J. N., Farquhar, G., Afouras, T., Nardelli, N., and Whiteson, S. (2018). Counterfactual multi-agent policy gradients. In *Proceedings of the AAAI Conference on Artificial Intelligence*. 3, 13
- Gangwani, T. and Peng, J. (2018). Genetic policy optimization. In *International Conference on Learning Representations*. 15
- Gao, Q., Thattai, G., Gao, X., Shakiah, S., Pansare, S., Sharma, V., Sukhatme, G., Shi, H., Yang, B., Zheng, D., et al. (2023). Alexa Arena: A user-centric interactive platform for embodied AI. *arXiv preprint arXiv:2303.01586*. 20
- Gao, X., Gao, Q., Gong, R., Lin, K., Thattai, G., and Sukhatme, G. S. (2022). DialFRED: Dialogue-enabled agents for embodied instruction following. *IEEE Robotics and Automation Letters*, 7(4):10049–10056. 20
- Gao, X., Gong, R., Zhao, Y., Wang, S., Shu, T., and Zhu, S.-C. (2020). Joint mind modeling for explanation generation in complex human-robot collaborative tasks. In *2020 29th IEEE International Conference on Robot and Human Interactive Communication (RO-MAN)*, pages 1119–1126. IEEE. 8, 78
- Gil, Ó., Garrell, A., and Sanfeliu, A. (2021). Social robot navigation tasks: Combining machine learning techniques and social force model. *Sensors*, 21(21):7087. 16
- Gong, R., Gao, X., Gao, Q., Shakiah, S., Thattai, G., and Sukhatme, G. S. (2023a). LEMMA: Learning language-conditioned multi-robot manipulation. *IEEE Robotics and Automation Letters*. 19

- Gong, R., Huang, J., Zhao, Y., Geng, H., Gao, X., Wu, Q., Ai, W., Zhou, Z., Terzopoulos, D., Zhu, S.-C., et al. (2023b). ARNOLD: A benchmark for language-grounded task learning with continuous states in realistic 3d scenes. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*. 20
- Gong, R., Huang, Q., Ma, X., Vo, H., Durante, Z., Noda, Y., Zheng, Z., Zhu, S.-C., Terzopoulos, D., Fei-Fei, L., and Gao, J. (2023c). MindAgent: Emergent gaming interaction. 10, 20, 21
- Gordon, D., Kembhavi, A., Rastegari, M., Redmon, J., Fox, D., and Farhadi, A. (2018). IQA: Visual question answering in interactive environments. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4089–4098. 20
- Grover, A., Al-Shedivat, M., Gupta, J., Burda, Y., and Edwards, H. (2018). Learning policy representations in multiagent systems. In *International Conference on Machine Learning*, pages 1802–1811. PMLR. 5
- Guldner, J. and Utkin, V. I. (1995). Sliding mode control for gradient tracking and robot navigation using artificial potential fields. *IEEE Transactions on Robotics and Automation*, 11(2):247–254. 16
- Guss, W. H., Houghton, B., Topin, N., Wang, P., Codel, C., Veloso, M., and Salakhutdinov, R. (2019). MineRL: A large-scale dataset of minecraft demonstrations. 21
- Hausknecht, M. and Stone, P. (2015). Deep reinforcement learning in parameterized action space. *arXiv preprint arXiv:1511.04143*. 3
- He, H., Boyd-Graber, J., Kwok, K., and Daumé III, H. (2016). Opponent modeling in deep reinforcement learning. In *International Conference on Machine Learning*, pages 1804–1813. 13
- Helbing, D. and Molnár, P. (1995). Social force model for pedestrian dynamics. *Physical Review E*, 51(5):4282–4286. 6, 16, 36
- Houthoofd, R., Chen, Y., Isola, P., Stadie, B., Wolski, F., Ho, O. J., and Abbeel, P. (2018). Evolved policy gradients. In *Advances in Neural Information Processing Systems*, pages 5400–5409. 15
- Hu, S., Zhu, F., Chang, X., and Liang, X. (2021). UPDeT: Universal multi-agent reinforcement learning via policy decoupling with transformers. 5, 15
- Huang, L., Gong, J., Li, W., Xu, T., Shen, S., Liang, J., Feng, Q., Zhang, D., and Sun, J. (2018). Social force model-based group behavior simulation in virtual geographic environments. *ISPRS International Journal of Geo-Information*, 7(2):79. 16
- Iqbal, S. and Sha, F. (2019). Actor-attention-critic for multi-agent reinforcement learning. In *International Conference on Machine Learning*, pages 2961–2970. 14

- Jaderberg, M., Czarnecki, W. M., Dunning, I., Marris, L., Lever, G., Castaneda, A. G., Beattie, C., Rabinowitz, N. C., Morcos, A. S., Ruderman, A., et al. (2019). Human-level performance in 3D multiplayer games with population-based reinforcement learning. *Science*, 364(6443):859–865. 10
- 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*. 15
- Jain, U., Weihs, L., Kolve, E., Farhadi, A., Lazebnik, S., Kembhavi, A., and Schwing, A. (2020). A cordial sync: Going beyond marginal policies for multi-agent embodied tasks. 10, 11, 20
- Jain, U., Weihs, L., Kolve, E., Rastegari, M., Lazebnik, S., Farhadi, A., Schwing, A., and Kembhavi, A. (2019). Two body problem: Collaborative visual task completion. 10, 11, 19, 77
- Jayannavar, P., Narayan-Chen, A., and Hockenmaier, J. (2020). Learning to execute instructions in a Minecraft dialogue. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 2589–2602. 20, 77
- Jiang, J. and Lu, Z. (2018). Learning attentional communication for multi-agent cooperation. In *Advances in Neural Information Processing Systems*. 5, 6, 13, 14
- 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. *arXiv preprint arXiv:2210.03094*, 2(3):6. 11, 20
- Johnson, M., Hofmann, K., Hutton, T., and Bignell, D. (2016). The malmo platform for artificial intelligence experimentation. In *International Joint Conference on Artificial Intelligence*, pages 4246–4247. 21
- Karras, T., Aila, T., Laine, S., and Lehtinen, J. (2017). Progressive growing of gans for improved quality, stability, and variation. *arXiv preprint arXiv:1710.10196*. 14
- Khadka, S. and Tumer, K. (2018). Evolution-guided policy gradient in reinforcement learning. In *Advances in Neural Information Processing Systems*, pages 1188–1200. 15
- Klančar, G., Zdešar, A., and Krishnan, M. (2022). Robot navigation based on potential field and gradient obtained by bilinear interpolation and a grid-based search. *Sensors*, 22(9):3295. 6, 16
- Kleiman-Weiner, M., Ho, M. K., Austerweil, J. L., Littman, M. L., and Tenenbaum, J. B. (2016). Coordinate to cooperate or compete: abstract goals and joint intentions in social interaction. In *Proceedings of the Annual Meeting of the Cognitive Science Society*. 8, 17
- Kolivand, H., Rahim, M. S., Sunar, M. S., Fata, A. Z. A., and Wren, C. (2021). An integration of enhanced social force and crowd control models for high-density crowd simulation. *Neural Computing and Applications*, 33:6095–6117. 6, 16

- Konolige, K. (2000). A gradient method for realtime robot control. In *Proceedings. 2000 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2000)(Cat. No. 00CH37113)*, volume 1, pages 639–646. IEEE. 6, 16
- Leibo, J. Z., Duñez-Guzmán, E., Vezhnevets, A. S., Agapiou, J. P., Sunehag, P., Koster, R., Matyas, J., Beattie, C., Mordatch, I., and Graepel, T. (2021). Scalable evaluation of multi-agent reinforcement learning with Melting Pot. 10, 19
- Levine, S., Finn, C., Darrell, T., and Abbeel, P. (2016). End-to-end training of deep visuomotor policies. *The Journal of Machine Learning Research*, 17(1):1334–1373. 3
- Li, G., Hammoud, H., Itani, H., Khizbullin, D., and Ghanem, B. (2023). CAMEL: Communicative agents for “mind” exploration of large language model society. *Advances in Neural Information Processing Systems*, 36:51991–52008. 20
- Lin, K., Zhao, R., Xu, Z., and Zhou, J. (2018). Efficient large-scale fleet management via multi-agent deep reinforcement learning. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 1774–1783. ACM. 13
- Littman, M. L. (1994). Markov games as a framework for multi-agent reinforcement learning. In *International Conference on Machine Learning*, volume 157, pages 157–163. 13
- Liu, H., Li, C., Wu, Q., and Lee, Y. J. (2024). Visual instruction tuning. *Advances in Neural Information Processing Systems*, 36. 69
- Liu, S., Lever, G., Heess, N., Merel, J., Tunyasuvunakool, S., and Graepel, T. (2019a). Emergent coordination through competition. In *International Conference on Learning Representations*. 3
- Liu, X., Guo, D., Liu, H., and Sun, F. (2022a). Multi-agent embodied visual semantic navigation with scene prior knowledge. *IEEE Robotics and Automation Letters*, 7(2):3154–3161. 19
- Liu, X., Li, X., Guo, D., Tan, S., Liu, H., and Sun, F. (2022b). Embodied multi-agent task planning from ambiguous instruction. *Proceedings of Robotics: Science and Systems, New York City, NY, USA*, pages 1–14. 11, 19
- Liu, Y., Hu, Y., Gao, Y., Chen, Y., and Fan, C. (2019b). Value function transfer for deep multi-agent reinforcement learning based on N-step returns. In *International Joint Conference on Artificial Intelligence*, pages 457–463. 16
- Long, Q., Li, R., Zhao, M., Gao, T., and Terzopoulos, D. (2024a). Inverse attention agents for multi-agent systems. *arXiv preprint. arXiv:2410.21794 [cs.AI]*. 2, 10
- Long, Q., Li, Z., Gong, R., Wu, Y. N., Terzopoulos, D., and Gao, X. (2024b). Team-Craft: A benchmark for multi-modal multi-agent systems in Minecraft. *arXiv preprint. arXiv:2412.05255 [cs.AI]*. 2, 12

- Long, Q., Zhong, F., Wu, M., Wang, Y., and Zhu, S.-C. (2024c). SocialGFs: Learning social gradient fields for multi-agent reinforcement learning. *arXiv preprint. arXiv:2405.01839 [cs.AI]*. 2, 8, 19
- Long, Q., Zhou, Z., Gupta, A., Fang, F., Wu, Y., and Wang, X. (2020). Evolutionary population curriculum for scaling multi-agent reinforcement learning. *arXiv preprint. arXiv:2003.10423 [cs.LG]*. 2, 5, 6, 15, 19, 44, 50
- Lowe, R., Wu, Y., Tamar, A., Harb, J., Abbeel, P., and Mordatch, I. (2017). Multi-agent actor-critic for mixed cooperative-competitive environments. *Advances in Neural Information Processing Systems*. 3, 5, 9, 13, 22, 29, 54
- Lowe, R., Wu, Y., Tamar, A., Harb, J., Abbeel, P., and Mordatch, I. (2020). Multi-agent actor-critic for mixed cooperative-competitive environments. 19
- Lupu, A., Cui, B., Hu, H., and Foerster, J. (2021). Trajectory diversity for zero-shot coordination. In *International Conference on Machine Learning*, pages 7204–7213. PMLR. 19
- Ma, X., Yong, S., Zheng, Z., Li, Q., Liang, Y., Zhu, S.-C., and Huang, S. (2023). SQA3D: Situated question answering in 3D scenes. 20
- Macke, W., Mirsky, R., and Stone, P. (2021). Expected value of communication for planning in ad hoc teamwork. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 11290–11298. 19
- Majumdar, A., Ajay, A., Zhang, X., Putta, P., Yenamandra, S., Henaff, M., Silwal, S., Mcvay, P., Maksymets, O., Arnaud, S., et al. (2024). OpenEQA: Embodied question answering in the era of foundation models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 16488–16498. 20
- Malysheva, A., Sung, T. T., Sohn, C.-B., Kudenko, D., and Shpilman, A. (2018). Deep multi-agent reinforcement learning with relevance graphs. *arXiv preprint arXiv:1811.12557*. 14
- Mandi, Z., Jain, S., and Song, S. (2024). RoCo: Dialectic multi-robot collaboration with large language models. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pages 286–299. IEEE. 11, 20, 77
- Matoui, F., Boussaid, B., and Abdelkrim, M. N. (2019). Distributed path planning of a multi-robot system based on the neighborhood artificial potential field approach. *Simulation*, 95(7):637–657. 6
- Mees, O., Hermann, L., Rosete-Beas, E., and Burgard, W. (2022). CALVIN: A benchmark for language-conditioned policy learning for long-horizon robot manipulation tasks. *IEEE Robotics and Automation Letters*, 7(3):7327–7334. 20
- Meier, P. (2015). *Digital humanitarians: How big data is changing the face of humanitarian response*. Routledge. 3

- Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T. P., Harley, T., Silver, D., and Kavukcuoglu, K. (2016). Asynchronous methods for deep reinforcement learning. *arXiv preprint arXiv:1602.01783*. 55
- Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. (2013). Playing Atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*. 3
- Mordatch, I. and Abbeel, P. (2017). Emergence of grounded compositional language in multi-agent populations. *arXiv preprint arXiv:1703.04908*. 19, 54
- Mordatch, I. and Abbeel, P. (2018). Emergence of grounded compositional language in multi-agent populations. In *Proceedings of the AAAI Conference on Artificial Intelligence*. 3, 28, 42
- Murali, A., Pinto, L., Gandhi, D., and Gupta, A. (2018). CASSL: Curriculum accelerated self-supervised learning. In *IEEE International Conference on Robotics and Automation*, pages 6453–6460. IEEE. 14
- Narayan-Chen, A., Jayannavar, P., and Hockenmaier, J. (2019). Collaborative dialogue in Minecraft. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 5405–5415. 20, 77
- Omidshafiei, S., Kim, D.-K., Liu, M., Tesauro, G., Riemer, M., Amato, C., Campbell, M., and How, J. P. (2019). Learning to teach in cooperative multiagent reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 6128–6136. 5
- Omidshafiei, S., Pazis, J., Amato, C., How, J. P., and Vian, J. (2017). Deep decentralized multi-task multi-agent reinforcement learning under partial observability. In *International Conference on Machine Learning*, pages 2681–2690. PMLR. 5
- OpenAI, :, Berner, C., Brockman, G., Chan, B., Cheung, V., Dębiak, P., Dennison, C., Farhi, D., Fischer, Q., Hashme, S., Hesse, C., Józefowicz, R., Gray, S., Olsson, C., Pachocki, J., Petrov, M., d. O. Pinto, H. P., Raiman, J., Salimans, T., Schlatter, J., Schneider, J., Sidor, S., Sutskever, I., Tang, J., Wolski, F., and Zhang, S. (2019). Dota 2 with large scale deep reinforcement learning. 5
- OpenAI (2018). OpenAI Five. <https://blog.openai.com/openai-five/>. 3
- Padmakumar, A., Thomason, J., Shrivastava, A., Lange, P., Narayan-Chen, A., Gella, S., Piramuthu, R., Tur, G., and Hakkani-Tur, D. (2022). TEACH: Task-driven embodied agents that chat. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pages 2017–2025. 20
- Panait, L. and Luke, S. (2005). Cooperative multi-agent learning: The state of the art. *AAMAS*, 11(3):387–434. 13

- Park, J. S., O'Brien, J., Cai, C. J., Morris, M. R., Liang, P., and Bernstein, M. S. (2023). Generative agents: Interactive simulacra of human behavior. In *Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology*, pages 1–22. [10](#), [19](#)
- Patten, S. N. (1896). *The theory of social forces*, volume 7. American Academy of Political and Social Science. [36](#)
- Peng, P., Yuan, Q., Wen, Y., Yang, Y., Tang, Z., Long, H., and Wang, J. (2017). Multiagent bidirectionally-coordinated nets for learning to play starcraft combat games. *arXiv preprint arXiv:1703.10069*, 2. [13](#)
- Perez-Liebana, D., Hofmann, K., Mohanty, S. P., Kuno, N., Kramer, A., Devlin, S., Gaina, R. D., and Ionita, D. (2019). The multi-agent reinforcement learning in MalmÖ (MARLÖ) competition. *arXiv preprint arXiv:1901.08129*. [11](#), [20](#), [21](#)
- Puig, X., Shu, T., Li, S., Wang, Z., Liao, Y.-H., Tenenbaum, J. B., Fidler, S., and Torralba, A. (2021). Watch-And-Help: A challenge for social perception and human-AI collaboration. [10](#), [11](#), [19](#), [20](#)
- Puig, X., Shu, T., Tenenbaum, J. B., and Torralba, A. (2023). NOPA: Neurally-guided online probabilistic assistance for building socially intelligent home assistants. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pages 7628–7634. IEEE. [19](#)
- Qiu, W., Zhong, F., Zhang, Y., Qiao, S., Xiao, Z., Kim, T. S., and Wang, Y. (2017). UnrealCV: Virtual worlds for computer vision. In *Proceedings of the 25th ACM International Conference on Multimedia*, pages 1221–1224. [77](#)
- Rabinowitz, N., Perbet, F., Song, F., Zhang, C., Eslami, S. A., and Botvinick, M. (2018). Machine theory of mind. In *International Conference on Machine Learning*, pages 4218–4227. PMLR. [18](#)
- Rahman, M. A., Hopner, N., Christianos, F., and Albrecht, S. V. (2021). Towards open ad hoc teamwork using graph-based policy learning. In *International Conference on Machine Learning*, pages 8776–8786. PMLR. [18](#)
- Rusu, A. A., Colmenarejo, S. G., Gulcehre, C., Desjardins, G., Kirkpatrick, J., Pascanu, R., Mnih, V., Kavukcuoglu, K., and Hadsell, R. (2016). Policy distillation. [16](#)
- Salimans, T., Ho, J., Chen, X., Sidor, S., and Sutskever, I. (2017). Evolution strategies as a scalable alternative to reinforcement learning. *arXiv preprint arXiv:1703.03864*. [15](#)
- Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. (2017). Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*. [55](#)
- Shalev-Shwartz, S., Shammah, S., and Shashua, A. (2016). Safe, multi-agent, reinforcement learning for autonomous driving. [5](#)
- Shoham, Y., Powers, R., and Grenager, T. (2003). Multi-agent reinforcement learning: A critical survey. *Web Manuscript*. [13](#)

- Shridhar, M., Manuelli, L., and Fox, D. (2021). CLIPort: What and where pathways for robotic manipulation. In *Proceedings of the 5th Conference on Robot Learning (CoRL)*. 20
- Shridhar, M., Thomason, J., Gordon, D., Bisk, Y., Han, W., Mottaghi, R., Zettlemoyer, L., and Fox, D. (2020a). ALFRED: A benchmark for interpreting grounded instructions for everyday tasks. 20
- Shridhar, M., Yuan, X., Côté, M.-A., Bisk, Y., Trischler, A., and Hausknecht, M. (2020b). ALFWorld: Aligning text and embodied environments for interactive learning. *arXiv preprint arXiv:2010.03768*. 20
- Shum, M., Kleiman-Weiner, M., Littman, M. L., and Tenenbaum, J. B. (2019). Theory of minds: Understanding behavior in groups through inverse planning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 6163–6170. 8, 17
- Smith, L. and Gasser, M. (2005). The development of embodied cognition: Six lessons from babies. *Artificial Life*, 11(1-2):13–29. 10
- Song, Y., Sohl-Dickstein, J., Kingma, D. P., Kumar, A., Ermon, S., and Poole, B. (2020). Score-based generative modeling through stochastic differential equations. *arXiv preprint arXiv:2011.13456*. 37
- Song, Y., Sohl-Dickstein, J., Kingma, D. P., Kumar, A., Ermon, S., and Poole, B. (2021). Score-based generative modeling through stochastic differential equations. 6, 17
- Stacy, S., Li, C., Zhao, M., Yun, Y., Zhao, Q., Kleiman-Weiner, M., and Gao, T. (2021). Modeling communication to coordinate perspectives in cooperation. In *Proceedings of the Annual Meeting of the Cognitive Science Society*. 17
- Stone, P., Kaminka, G., Kraus, S., and Rosenschein, J. (2010). Ad hoc autonomous agent teams: Collaboration without pre-coordination. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 24, pages 1504–1509. 18
- Stone, P. and Veloso, M. (2000). Multiagent systems: A survey from a machine learning perspective. *Autonomous Robots*, 8:345–383. 10
- Suárez, J., Bloomin, D., Choe, K. W., Li, H. X., Sullivan, R., Kanna, N., Scott, D., Shuman, R., Bradley, H., Castricato, L., et al. (2024). Neural MMO 2.0: a massively multi-task addition to massively multi-agent learning. *Advances in Neural Information Processing Systems*, 36. 20
- Suarez, J., Du, Y., Isola, P., and Mordatch, I. (2019). Neural MMO: A massively multiagent game environment for training and evaluating intelligent agents. *arXiv preprint arXiv:1903.00784*. 13
- Suarez, J., Du, Y., Zhu, C., Mordatch, I., and Isola, P. (2021). The neural MMO platform for massively multiagent research. In Vanschoren, J. and Yeung, S., editors, *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks*, volume 1. 10, 19

- Sukhbaatar, S., Fergus, R., et al. (2016). Learning multiagent communication with backpropagation. In *Advances in Neural Information Processing Systems*, pages 2244–2252. 3
- Sukhbaatar, S., Lin, Z., Kostrikov, I., Synnaeve, G., Szlam, A., and Fergus, R. (2017). Intrinsic motivation and automatic curricula via asymmetric self-play. *arXiv preprint arXiv:1703.05407*. 14
- Szot, A., Clegg, A., Undersander, E., Wijmans, E., Zhao, Y., Turner, J., Maestre, N., Mukadam, M., Chaplot, D. S., Maksymets, O., et al. (2021). Habitat 2.0: Training home assistants to rearrange their habitat. *Advances in neural information processing systems*, 34:251–266. 19
- Tan, S., Xiang, W., Liu, H., Guo, D., and Sun, F. (2020). Multi-agent embodied question answering in interactive environments. In *Computer Vision—ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XIII 16*, pages 663–678. Springer. 19
- Tang, N., Gong, S., Zhao, M., Gu, C., Zhou, J., Shen, M., and Gao, T. (2022). Exploring an imagined “we” in human collective hunting: Joint commitment within shared intentionality. In *Proceedings of the Annual Meeting of the Cognitive Science Society*, volume 44. 8
- Tang, N., Stacy, S., Zhao, M., Marquez, G., and Gao, T. (2020). Bootstrapping an imagined we for cooperation. In *Proceedings of the Annual Meeting of the Cognitive Science Society*. 17
- Vail, D. and Veloso, M. (2003). Multi-robot dynamic role assignment and coordination through shared potential fields. *Multi-Robot Systems*, 2:87–98. 16
- Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. (2017). Attention is all you need. In *Advances in Neural Information Processing Systems*, pages 5998–6008. 4, 14, 22, 49
- Vincent, P. (2011). A connection between score matching and denoising autoencoders. *Neural Computation*, 23(7):1661–1674. 37
- Vinitsky, E., Köster, R., Agapiou, J. P., Duéñez-Guzmán, E., Vezhnevets, A. S., and Leibo, J. Z. (2022). A learning agent that acquires social norms from public sanctions in decentralized multi-agent settings. 5
- Vinyals, O., Babuschkin, I., Czarnecki, W. M., Mathieu, M., Dudzik, A., Chung, J., Choi, D. H., Powell, R., Ewalds, T., Georgiev, P., et al. (2019). Grandmaster level in StarCraft II using multi-agent reinforcement learning. *Nature*, 575(7782):350–354. 5, 19
- Wan, J., Sui, J., and Yu, H. (2014). Research on evacuation in the subway station in China based on the combined social force model. *Physica A: Statistical Mechanics and its Applications*, 394:33–46. 6, 16

- Wan, Y., Mao, J., and Tenenbaum, J. (2022). HandMeThat: Human-robot communication in physical and social environments. *Advances in Neural Information Processing Systems*, 35:12014–12026. 20
- Wang, G., Xie, Y., Jiang, Y., Mandlekar, A., Xiao, C., Zhu, Y., Fan, L., and Anandkumar, A. (2023a). Voyager: An open-ended embodied agent with large language models. 11, 21
- Wang, R., Lehman, J., Clune, J., and Stanley, K. O. (2019). Paired open-ended trailblazer (poet): Endlessly generating increasingly complex and diverse learning environments and their solutions. *arXiv preprint arXiv:1901.01753*. 14
- Wang, X., Girshick, R., Gupta, A., and He, K. (2018). Non-local neural networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 14
- Wang, Y., Zhong, F., Xu, J., and Wang, Y. (2022). ToM2C: Target-oriented multi-agent communication and cooperation with theory of mind. In *International Conference on Learning Representations*. 5, 55
- Wang, Z., Cai, S., Liu, A., Jin, Y., Hou, J., Zhang, B., Lin, H., He, Z., Zheng, Z., Yang, Y., et al. (2023b). JARVIS-1: Open-world multi-task agents with memory-augmented multimodal language models. *arXiv preprint arXiv:2311.05997*. 77
- Wang, Z., Cai, S., Liu, A., Ma, X., and Liang, Y. (2023c). Describe, explain, plan and select: Interactive planning with large language models enables open-world multi-task agents. 77
- Warren, C. W. (1989). Global path planning using artificial potential fields. In *IEEE International Conference on Robotics and Automation*, pages 316–317. IEEE Computer Society. 6
- Wellman, M. P., Reeves, D. M., Lochner, K. M., Cheng, S.-F., and Suri, R. (2005). Approximate strategic reasoning through hierarchical reduction of large symmetric games. In *Proceedings of the AAAI Conference on Artificial Intelligence*. 3
- Wen, M., Kuba, J., Lin, R., Zhang, W., Wen, Y., Wang, J., and Yang, Y. (2022). Multi-agent reinforcement learning is a sequence modeling problem. *Advances in Neural Information Processing Systems*, 35:16509–16521. 5
- Wright, M., Wang, Y., and Wellman, M. P. (2019). Iterated deep reinforcement learning in games: History-aware training for improved stability. In *Proceedings of the 2019 ACM Conference on Economics and Computation*, pages 617–636. ACM. 13
- Wu, C., Kreidieh, A., Vinitzky, E., and Bayen, A. M. (2017). Emergent behaviors in mixed-autonomy traffic. In *CoRL*, pages 398–407. 3
- Wu, M., Zhong, F., Xia, Y., and Dong, H. (2022). TarGF: Learning target gradient field for object rearrangement. *arXiv preprint arXiv:2209.00853*. 17, 83, 84

- Wu, S. A., Wang, R. E., Evans, J. A., Tenenbaum, J. B., Parkes, D. C., and Kleiman-Weiner, M. (2021). Too many cooks: Bayesian inference for coordinating multi-agent collaboration. *Topics in Cognitive Science*, 13(2):414–432. 10, 17
- Wu, T., Wu, M., Zhang, J., Gan, Y., and Dong, H. (2024). Learning score-based grasping primitive for human-assisting dexterous grasping. *Advances in Neural Information Processing Systems*, 36. 17
- Wu, Y. and Tian, Y. (2016). Training agent for first-person shooter game with actor-critic curriculum learning. In *International Conference on Learning Representations*. 14
- Wu, Y., Wu, Y., Gkioxari, G., and Tian, Y. (2018). Building generalizable agents with a realistic and rich 3D environment. *arXiv preprint arXiv:1801.02209*. 3
- Xie, A., Losey, D., Tolsma, R., Finn, C., and Sadigh, D. (2021). Learning latent representations to influence multi-agent interaction. In *Conference on Robot Learning*, pages 575–588. PMLR. 18
- Xu, J., Zhong, F., and Wang, Y. (2020). Learning multi-agent coordination for enhancing target coverage in directional sensor networks. *Advances in Neural Information Processing Systems*, 33:10053–10064. 6
- Xue, T., Wu, M., Lu, L., Wang, H., Dong, H., and Chen, B. (2023). Learning gradient fields for scalable and generalizable irregular packing. In *SIGGRAPH Asia 2023 Conference Papers*, pages 1–11. 17
- Yaman, A., Leibo, J. Z., Iacca, G., and Lee, S. W. (2022). The emergence of division of labor through decentralized social sanctioning. 5
- Yang, W., Wang, X., Farhadi, A., Gupta, A., and Mottaghi, R. (2019). Visual semantic navigation using scene priors. In *International Conference on Learning Representations*. 3, 14
- Yang, Y., Luo, R., Li, M., Zhou, M., Zhang, W., and Wang, J. (2018). Mean field multi-agent reinforcement learning. In *International Conference on Machine Learning*, pages 5567–5576. 3, 13, 29
- Yang, Y., Luo, R., Li, M., Zhou, M., Zhang, W., and Wang, J. (2020). Mean field multi-agent reinforcement learning. 15
- Yu, C., Velu, A., Vinitzky, E., Gao, J., Wang, Y., Bayen, A., and Wu, Y. (2021). The surprising effectiveness of PPO in cooperative multi-agent games. 19, 44, 83, 85
- Yu, C., Velu, A., Vinitzky, E., Gao, J., Wang, Y., Bayen, A., and Wu, Y. (2022). The surprising effectiveness of PPO in cooperative multi-agent games. In *Advances in Neural Information Processing Systems*, volume 35, pages 24611–24624. 53, 55
- Yu, X., Fu, J., Deng, R., and Han, W. (2024). MineLand: Simulating large-scale multi-agent interactions with limited multimodal senses and physical needs. 11

- Zambaldi, V., Raposo, D., Santoro, A., Bapst, V., Li, Y., Babuschkin, I., Tuyls, K., Reichert, D., Lillicrap, T., Lockhart, E., et al. (2018). Relational deep reinforcement learning. *arXiv preprint arXiv:1806.01830*. 14
- Zeng, A., Florence, P., Tompson, J., Welker, S., Chien, J., Attarian, M., Armstrong, T., Krasin, I., Duong, D., Sindhvani, V., and Lee, J. (2020). Transporter networks: Rearranging the visual world for robotic manipulation. In *Conference on Robot Learning (CoRL)*. 20
- 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 agents with large language models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 17591–17599. 10
- 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 *The Twelfth International Conference on Learning Representations*. 19
- Zhang, H., Wang, Z., Lyu, Q., Zhang, Z., Chen, S., Shu, T., Du, Y., and Gan, C. (2024c). COMBO: Compositional world models for embodied multi-agent cooperation. *arXiv preprint arXiv:2404.10775*. 11
- Zhao, Y.-y., Yang, Z., Kong, W.-r., Piao, H.-y., Huang, J.-c., Lv, X.-f., and Zhou, D.-y. (2022). Hybrid gradient vector fields for path-following guidance. *Defence Technology*. 6, 16
- Zheng, K., Chen, X., Jenkins, O. C., and Wang, X. (2022). VLMbench: A compositional benchmark for vision-and-language manipulation. *Advances in Neural Information Processing Systems*, 35:665–678. 20
- Zheng, Y., Li, B., An, D., and Li, N. (2015). A multi-agent path planning algorithm based on hierarchical reinforcement learning and artificial potential field. In *2015 11th International Conference on Natural Computation (ICNC)*, pages 363–369. IEEE. 6
- Zhong, F., Sun, P., Luo, W., Yan, T., and Wang, Y. (2021). Towards distraction-robust active visual tracking. In *International Conference on Machine Learning*, pages 12782–12792. PMLR. 5
- Zhou, T., Zhang, F., Shao, K., Li, K., Huang, W., Luo, J., Wang, W., Yang, Y., Mao, H., Wang, B., Li, D., Liu, W., and Hao, J. (2021). Cooperative multi-agent transfer learning with level-adaptive credit assignment. 5, 15
- Zhu, Y., Mottaghi, R., Kolve, E., Lim, J. J., Gupta, A., Fei-Fei, L., and Farhadi, A. (2017). Target-driven visual navigation in indoor scenes using deep reinforcement learning. In *IEEE International Conference on Robotics and Automation*, pages 3357–3364. IEEE. 3
- Zintgraf, L., Devlin, S., Ciosek, K., Whiteson, S., and Hofmann, K. (2021). Deep interactive bayesian reinforcement learning via meta-learning. *arXiv preprint arXiv:2101.03864*. 19