Appendix

A More Related Work

Transfer learning Transfer learning holds the promise of improving the sample efficiency of reinforcement learning [65], which learns knowledge from source tasks to accelerate the learning efficiency in the unseen task. Previous work uses successor features, decouples the transition dynamics and reward function, and learns faster in simulated navigation and robotic arm settings [3]. DSE [31] models the transfer process as variational inference and further learns a latent space to transfer skills across different dynamics. [37] apply a model-based regularizer to learn task-level transfer across various observation spaces. Related to our work, REPAINT [39] combines task representations with on-policy learning and uses an advantage-based experience selection approach to transfer useful samples. Our method differs from these works by (1) its mechanism for alternately updating the task representation and the representation explainer and (2) its mechanism for handling different input/output sizes.

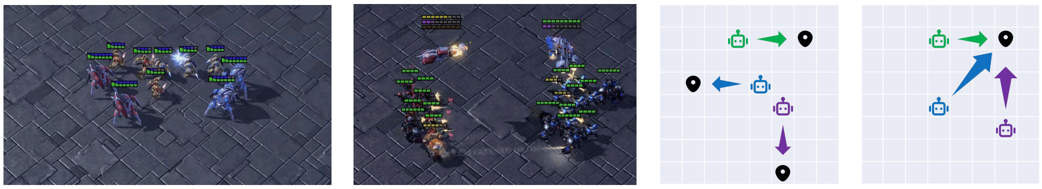
The basic idea behind **multi-agent transfer learning** [35] is to reuse knowledge from other tasks or other learning agents, corresponding to intra-agent transfer and inter-agent transfer, respectively. It is expected that the knowledge reuse can accelerate coordination compared to learning from scratch. The inter-agent transfer paradigm aims at reusing knowledge from other agents with different sensors or (possibly) internal representations via communication. DVM [41] treats the multi-agent problem as a multi-task problem to combine knowledge from multiple tasks and then distills the knowledge by a value matching mechanism. LeCTR [29] learns to teach in a multi-agent environment and learns to advise others in a peer-to-peer manner. MAPTF [55] takes a further step by proposing an option-based policy transfer for multi-agent cooperation, and it significantly boosts the performance of existing methods in both discrete and continuous state spaces. On the other hand, intra-agent transfer refers to reusing knowledge from previous tasks, focusing on transferring knowledge across multi-agent tasks. The varying populations and input lengths impede the transfer among agents, with which the graph neural networks and the transformer play promising roles. DyMA-CL [49] designs various transfer mechanisms across curricula to accelerate the learning process based on a dynamic agent-number network. EPC [26] proposes a curriculum learning paradigm via an evolutionary approach to scale up the population number of agents. UPDeT [17] and PIT [64] make use of the generalization ability of the transformer to accomplish the multi-agent cooperation and transfer between tasks. Although these methods can accelerate the learning efficiency of MARL algorithms, they do not exploit task similarity for better transfer performance. By contrast, our method explicitly models task relationship by learning a hidden space in which tasks with similar dynamics have similar representations.

Multi-task reinforcement learning is another relevant research topic that enables an RL agent to leverage experience from multiple tasks to improve sample efficiency and avoid learning from scratch on every single task. Various approaches have been proposed to achieve multi-task learning, such as distilling separate tasks' knowledge into a shared policy [25, 12, 52], conditioning policies on tasks [8], mapping tasks to parameters of a policy [7, 22, 54], and solving the problem of negative interference [9, 38, 57] meaning that gradients of different tasks may conflict.

Multi-agent representation learning Learning effective representation in MARL is receiving significant attention for its effectiveness in solving many important problems. CQ-Learning [16] learns to adapt the state representation for multi-agent systems to coordinate with other agents. [13] learn useful policy representations to model agent's behavior in a multi-agent system. LILI [51] learns latent representations to capture the relationship between ego-agent's behavior and the other agent's future strategy. RODE [46] uses an action encoder to learn action representations and applies clustering methods to decompose the joint action space into restricted role action spaces to reduce the policy search space of multi-agent cooperation. MAR [62] learns meta representation for multi-agent generalization. MACC [58] uses local information to anticipate the task embedding for decentralized decision-making. NDQ [47] and MAIC [59] capture the relationships between agents by maximizing the mutual information, then achieve efficient multi-agent communication. and Our approach differs from these works by learning representations for tasks and using a representation explainer for efficient policy transfer.

Another line of research on single-agent multi-task learning that has the potential to be effective in multi-agent settings is **modular RL**. Modular RL decentralizes the control of multi-joint robots by learning policies for each actuator and thus holds the promise to deal with input and output with varying lengths. Each joint has its controlling policy, and they coordinate with each other via various message passing schemes. To do so, [44] and [30] represent the robot's morphology as a graph and use GNNs as policy and message passing networks. [18] use both bottom-up and top-down message passing schemes through the links between joints for coordinating. All of these GNN-like works show the benefits of modular policies over a monolithic policy in tasks tackling different morphologies. However, recently, [23] validated a hypothesis that any benefit GNNs can extract from morphological structures is outweighed by the difficulty of message passing across multiple hops. They further propose a transformer-based method, AMORPHEUS, that utilizes self-attention mechanisms as a message passing approach. Although modular RL can deal with varying action numbers of different tasks and can implicitly model the interaction between agents through GNN, Transformer, or message passing, it still cannot cope with varying observation lengths and does not incorporate task-level context information compared to our method. In summary, our work distinguishes itself from previous multi-task work by (1) its flexibility of handling varying lengths of observation and varying numbers of actions; (2) its utilization of agent-interaction modeling in capturing task relation; and (3) its alternatively fixed task representation learning scheme.

B Details about the Benchmarks



(a) SMAC: 2s3z (b) SMAC: MMM2 (c) MPE: Spread (d) MPE: Gather

Figure 5: Snapshots of the experimental environments used in this paper.

SMAC (Fig. 5(a)∼(b)) StarCraft II Micromanagement Benchmark [34] contains combat scenarios of StarCraft II unit micromanagement tasks and is a popular benchmark for multi-agent reinforcement learning. We consider a partially observable setting, where an agent can only see a circular area around it with a radius equal to its sight range, which is default to 9. We train ally units with MATTAR to fight against enemy units controlled by the built-in AI. At the beginning of each episode, allies and enemies spawn in pre-defined regions on the map. Every agent takes actions from a discrete action space including `no-op`, `move[direction]`, `attack[enemyid]`, and `stop`. Under the control of these actions, agents can move and attack on a continuous map. Agents will get a shared reward equal to the total damage dealt to enemy units at each timestep. Killing each enemy unit and winning the combat (killing all the enemies) will bring additional bonuses of 10 and 200, respectively. We consider three series of SMAC tasks, each including various maps. The detailed descriptions are shown in Tab. 7∼9.

MPE (Fig. 5(c)∼(d)) Multi-Agent Particle Environment [27] is a multi-agent particle world containing several navigation and communication tasks. In our experiments, we consider a discrete version of MPE and use two tasks, `Spread` and `Gather`, to evaluate our method. In both tasks, there are `n_agent` agents on a field with size `[field_size, field_size]` tasked to reach landmarks. The agents can observe objects around it within a distance of `sight_range`. When a landmark is within a `reach_range`× `reach_range` sub-field around an agent, we say the agent has reached the landmark. In `Spread`, we require each agent to reach a landmark that is not occupied by any other agents, while in `Gather`, the agents share a common landmark. In both of these tasks, only when all agents reach the landmark, a collective reward of 1 is given. For both `Spread` and `Gather`, we test several tasks with different numbers of agents. The detailed settings of these tasks are listed in Tab. 10 and 11.

Table 7: Settings of tasks in the MMM series. The bolded names indicate the source tasks.

Map Name	Ally Units	Enemy Units	Type	Difficulty
MMM0	1 Medivac, 2 Marauders, 5 Marines	1 Medivac, 2 Marauders, 5 Marines	Asymmetric & Heterogeneous	Easy
MMM	1 Medivac, 2 Marauders, 7 Marines	1 Medivac, 2 Marauders, 7 Marines	Asymmetric & Heterogeneous	Easy
MMM1	1 Medivac, 1 Marauders, 7 Marines	1 Medivac, 2 Marauders, 7 Marines	Asymmetric & Heterogeneous	Hard
MMM2	1 Medivac, 2 Marauders, 7 Marines	1 Medivac, 3 Marauders, 8 Marines	Asymmetric & Heterogeneous	Super Hard
MMM3	1 Medivac, 2 Marauders, 8 Marines	1 Medivac, 3 Marauders, 9 Marines	Asymmetric & Heterogeneous	Super Hard
MMM4	1 Medivac, 3 Marauders, 8 Marines	1 Medivac, 4 Marauders, 9 Marines	Asymmetric & Heterogeneous	Super Hard
MMM5	1 Medivac, 3 Marauders, 8 Marines	1 Medivac, 4 Marauders, 10 Marines	Asymmetric & Heterogeneous	Super Hard
MMM6	1 Medivac, 3 Marauders, 8 Marines	1 Medivac, 4 Marauders, 11 Marines	Asymmetric & Heterogeneous	Super Hard

Table 8: Settings of tasks in the SZ series. The bolded names indicate the source tasks.

Map Name	Ally Units	Enemy Units	Type	Difficulty
1s8z	1 Stalkers, 8 Zealots	1 Stalkers, 8 Zealots	Symmetric & Heterogeneous	Easy
1s9z	1 Stalkers, 9 Zealots	1 Stalkers, 9 Zealots	Symmetric & Heterogeneous	Easy
2s3z	2 Stalkers, 3 Zealots	2 Stalkers, 3 Zealots	Symmetric & Heterogeneous	Easy
2s8z	2 Stalkers, 8 Zealots	2 Stalkers, 8 Zealots	Symmetric & Heterogeneous	Easy
2s9z	2 Stalkers, 9 Zealots	2 Stalkers, 9 Zealots	Symmetric & Heterogeneous	Easy
3s5z	3 Stalkers, 5 Zealots	3 Stalkers, 5 Zealots	Symmetric & Heterogeneous	Easy
3s5z_vs_3s6z	3 Stalkers, 5 Zealots	3 Stalkers, 6 Zealots	Symmetric & Heterogeneous	Super Hard
7s3z	7 Stalkers, 3 Zealots	7 Stalkers, 3 Zealots	Symmetric & Heterogeneous	Easy

C Network Architecture and Hyperparameters

Our implementation of MATTAR is based on PyMARL [3] with StarCraft 2.4.6.2.69232 and uses its default hyperparameter settings. We apply the default ϵ-greedy action selection algorithm to every algorithm, as ϵ decays from 1 to 0.05 in 50K timesteps. We also adopt typical Q-learning training tricks like the target network and double Q-learning. MATTAR has hyperparameters λ_1, λ_2, and λ as the scaling factors of the observation prediction loss, the reward prediction loss, and the entropy regularization term, respectively. We set them to $1, 10$, and 0.1 across all experiments. For other hyper-parameters, we use the default settings of QMIX presented in the PyMARL framework. For

---

[3] We use PyMARL with SC2.4.6.2.6923. Performance is not always comparable among versions.

Table 9: Settings of tasks in the M series. The bolded names indicate the source tasks.

Map Name	Ally Units	Enemy Units	Type	Difficulty
3m	3 Marines	5 Marines	Symmetric & Homogeneous	Easy
4m	4 Marines	5 Marines	Symmetric & Homogeneous	Easy
4m_vs_5m	4 Marines	5 Marines	Asymmetric & Homogeneous	Hard
5m	5 Marines	5 Marines	Symmetric & Homogeneous	Easy
5m_vs_6m	5 Marines	6 Marines	Asymmetric & Homogeneous	Hard
6m	6 Marines	6 Marines	Symmetric & Homogeneous	Easy
6m_vs_7m	6 Marines	7 Marines	Asymmetric & Homogeneous	Hard
7m	7 Marines	7 Marines	Symmetric & Homogeneous	Easy
7m_vs_8m	7 Marines	8 Marines	Asymmetric & Homogeneous	Hard
8m	8 Marines	8 Marines	Symmetric & Homogeneous	Easy
8m_vs_9m	8 Marines	9 Marines	Asymmetric & Homogeneous	Easy
9m	9 Marines	9 Marines	Symmetric & Homogeneous	Easy
9m_vs_10m	9 Marines	10 Marines	Asymmetric & Homogeneous	Easy
10m	10 Marines	10 Marines	Symmetric & Homogeneous	Easy
10m_vs_11m	10 Marines	11 Marines	Asymmetric & Homogeneous	Easy
10m_vs_12m	10 Marines	12 Marines	Asymmetric & Homogeneous	Super Hard

Table 10: Settings of the `Spread` tasks. The bolded identities indicate the source tasks.

Task Identity	# of Agents	# of Landmarks	Field Size	Sight Range	Reach Range
2	2	2	6	5	2
3	3	3	8	7	2
4	4	4	10	9	2
5	5	5	10	9	2
6	6	6	15	14	2
7	7	7	15	14	2
8	8	8	15	14	2
9	9	9	15	15	2

RODE [46], ASN [48], QPLEX [43], QMIX [33], and UPDeT [17], we use the codes provided by the authors with the default hyperparameters settings. We describe our network structure in Tab. 12. This network architecture is used for all experiments in the paper.

D Experimental Details

Our experiments were performed on 2 NVIDIA GTX 2080 Ti GPUs. For all the performance curves in our paper, we pause training every 10K timesteps and evaluate for 32 episodes with decentralized greedy action selection. We present the percentage of episodes in which the agents defeat all enemies within the time limit. We now provide details about each part of our experiments.

Generalization to unseen tasks For baselines and ablations, we carried out experiments with 5 different random seeds. In each experiment, we evaluate the trained model for 32 episodes on each unseen task. The results recorded in Tab. 1$\sim$3 are the mean and variance of these 5 random seeds.

Fine-tuning For the performance of fine-tuning MATTAR, we trained 2 source models with different random seeds for each unseen map and used 2 random seeds for each source model for fine-tuning. For learning from scratch, we carried out experiments with 4 different random seeds for each map.

Multi-task learning We carried out experiments with 5 different random seeds for both multi-task learning and learning on a single task. For the experiments of multi-task learning on three tasks shown in the paper, the training sets are {`5m`, `5m_vs_6m`, `8m_vs_9m`, `10m_vs_11m`}, {`2s3z`, `3s5z`, `3s5z_vs_3s6z`}, and {`MMM`, `MMM2`, `MMM4`}, respectively.

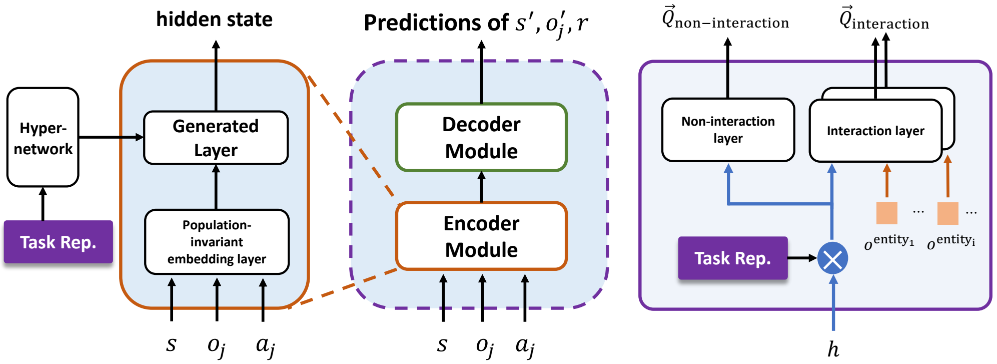
Single-task learning For this experiment, we tested each baseline and ablation algorithm with 5 random seeds.

Table 11: Settings of the `Gather` tasks. The bolded identities indicate the source tasks.

Task Identity	# of Agents	# of Landmarks	Field Size	Sight Range	Reach Range
2	2	1	6	5	2
3	3	1	8	7	2
4	4	1	10	9	2
5	5	1	10	9	2
7	7	1	15	14	2
9	9	1	15	15	2
10	10	1	20	19	2
15	15	1	20	19	2

Table 12: Hyperparameters about the network structure in our experiments.

name	value
The hidden dimension for mixing network	32
The number of layers for the hypernet in mixing network	2
The hidden dimension for the hypernet	64
The length of the encoding vector of agent ID	4
The dimension of task representations	32
The output dimension of the encoder in the forward model	32
The output dimension of the attention module	64
The hidden dimension for the query and key in attention module	8



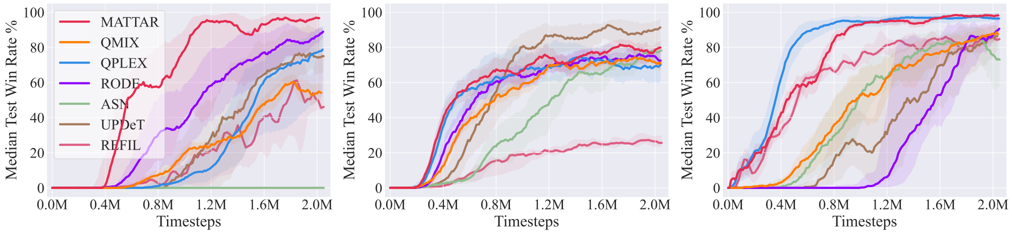
(a) Network architecture for the forward model (b) Q-network for interaction actions

Figure 6: The architecture of our forward model and the Q-network for interaction actions. In (b), o^{entity_i} denotes the observation component corresponding to the influence of the i-th interactive action, $\vec{Q}_{\text{non-interaction}}$ denotes the Q-values of non-interactive actions, and $\vec{Q}_{\text{interaction}}$ denotes the Q-values of interactive actions.

E Forward Model for Task Representation Learning

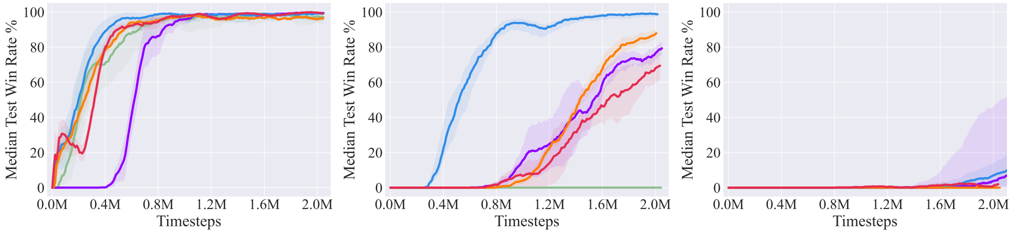
In our method, we utilize dynamics modelling to learn task representations which can capture the similarity between different tasks. We use a hypernetwork as the representation explainer to generate the parameters of the forward model. In practical implementation, the forward model consists two components, an encoder and a decoder (Fig. 6(a)).

For the encoder network, we first use the population-invariant embedding layer to get a fixed-dimensional embedding vector and feed it into a fully-connected layer whose parameters are generated by the representation explainer. The output hidden variables are fed into the decoder to predict the next state, the next observation, and the global reward. The encoder module and the representation explainer are shared among tasks and are fixed when learning representations for unseen tasks.

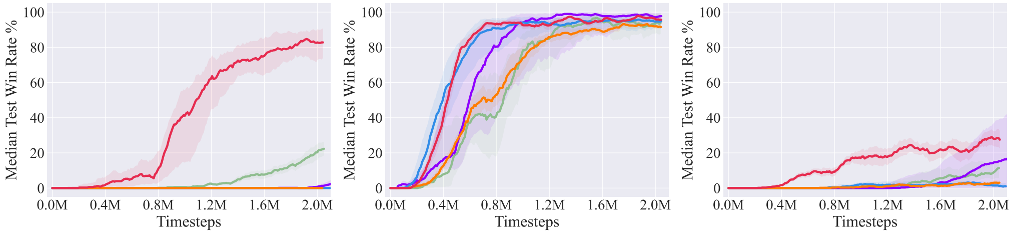


(a) MMM2 (b) 5m_vs_6m (c) 3s5z

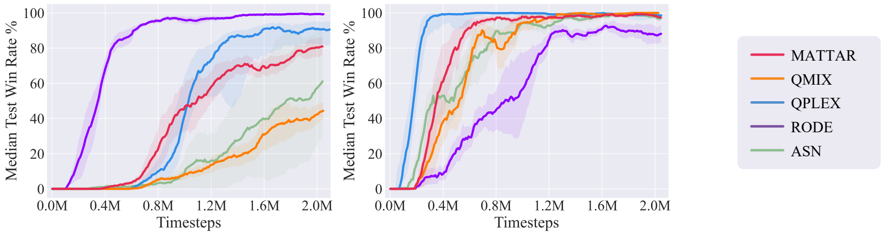
Figure 7: A bonus: when learning from scratch on single tasks, MATTAR architecture exhibits good performance. For performance on more SMAC benchmark, please refer to Fig. 8.



(a) 2s3z (b) 3s_vs_5z (c) 3s5z_vs_3s6z



(d) corridor (e) 10m_vs_11m (f) 6h_vs_8z



(g) 2c_vs_64zg (h) 2s_vs_1sc

Figure 8: More results for the performance of MATTAR on the SMAC benchmark when learning from scratch on single tasks.

The decoder module is task-specific, and we allow the decoder to be optimized together with task representations when adapting to unseen tasks.

For the population-invariant embedding layer in the encoder module, like in the policy, we decompose the input state and observation into several entity-specific components, pass them through an embedding layer, and do a pooling operation for output vectors. We also deal with the case of the varying number of actions in the input by incorporating actions into to observation o_i. We concatenate non-interaction actions with agent i's own observation component o_i^{own} and interaction actions with the observation components corresponding to each entity. We also note that other population-invariant structures can also be applied to our approach.

F Varying Numbers of Actions

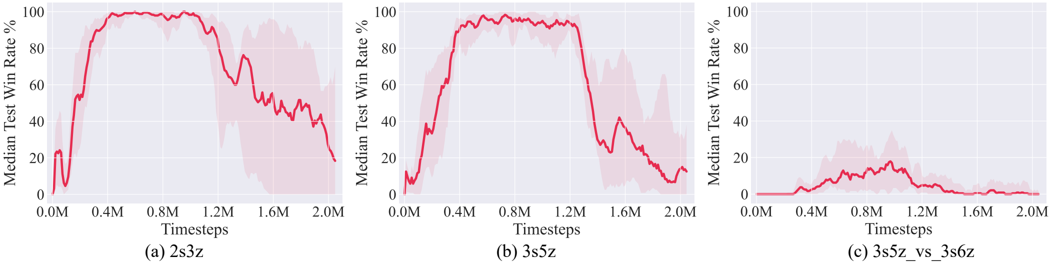
In some multi-agent environments, there are interaction actions that have semantics relating to other opponents in the environment. In this case, the action dimension is related to the number of opponents, preventing flexible transfer to unseen tasks. To deal with this problem, we adopt the structure shown in Fig. 6(b) for the estimation of Q-values for these interaction actions. In this structure, Q-values for interaction and non-interaction actions are estimated separately. For non-interaction actions, we use a fully-connected network whose input is the concatenation of observation encoding h and task representation z. For an interaction action, we use a network that takes as input the concatenation of h, task representation z, and the observation component relating to the corresponding opponent.

G Bonus: performance on single-task training

Although not designed for this goal, we find that MATTAR can outperform state-of-the-art MARL algorithms when trained on some single tasks. Specifically, we remove the task representation module and train MATTAR from scratch. We compare our method with two state-of-the-art value-based MARL baselines (QMIX [33] and QPLEX [43]), a role-based learning algorithm (RODE) [46], and one underlying algorithm of MATTAR which considers the Q-values of interaction actions separately (ASN) [48]. For the representative tasks of the three series in the main text (MMM2, 5m_vs_6m and 3s5z), we additionally compare with two methods with similar attentional mechanisms (UPDeT [17] and REFIL [19]).

Fig. 7 shows the learning curves of different methods. We find that our population-invariant network structure achieves comparable performance in all tasks. It is worth noting that this structure even significantly outperforms all other algorithms on the super hard map `MMM2`. In Fig. 8, we show the comparison on more SMAC maps, on which MATTAR also has comparable performance. Given that our underlying algorithm is QMIX, this is an inspiring result. We hypothesize that our self-attention scheme increases the representational capacity by learning to attend to appropriate entities in the environment.

H Testing of one alternative for source task definition



(a) 2s3z (b) 3s5z (c) 3s5z_vs_3s6z

Figure 9: Testing of one alternative for source task definition. In this experiment, we apply a normalization layer on top of the joint learned vectors with the representation explainer, and use it as the task representations for source tasks.

One possible method for determining the source task representations is to joint learn representation explainer and task representations together. However, this practice often brings representations with very small norms. A remedy for this problem is to additionally apply a normalization layer on top of the jointly learned vectors. We conduct experiments to validate this approach, and the results of learning performance on source tasks for the SZ series are shown in Fig. 9. It is interesting that the learning curves on source tasks begin to drop after about 1M samples for all these three tasks. These results show that it is hard to get a meaningful representation space when learning together with the representation explainer. We hypothesize the reason behind the phenomenon is that there are limited signals that can guarantee information about task relationship is encoded in the representation space.

I Discussions when tasks under great differences

Our approach shows great advantages over ablations and baselines in the experimental results reported in the main text. To further explore the ability of our approach, we additionally conduct two experiments, and the results are reported in Tab. 13.

Table 13: Two additional experiments where the unseen tasks are quite different.

	Source Tasks			Unseen Tasks			
	1s2z	1s3z	2s3z	3s5z	3s5z_3s6z	4s7z	4s7z_4s8z
MATTAR	0.94±0.04	0.97±0.03	0.91±0.07	0.68±0.12	0.01±0.01	0.39±0.19	0.00±0.00
	MMM	MMM2	MMM4	1s8z	2s3z	3s5z	7s3z
MATTAR	0.99±0.01	0.85±0.01	0.89±0.03	0.00±0.00	0.00±0.00	0.00±0.00	0.05±0.07

In the first experiment, we design source tasks and target tasks with a huge gap in terms of the number of agents. Specifically, We train MATTAR on three source tasks: `1s2z`, `1s3z`, and `2s3z`, each of which contains a small number of agents and has a difficulty level of "easy", and we test it on tasks with more agents and higher difficulty level. As we can see, the transfer performance drops when the number of agents has a huge increase, and the win rate is 0 on `4s7z_4s8z`.

In the second experiment, we train MATTAR on three tasks from the MMM series and test it on some tasks of the SZ series. We can see that when the source tasks are not diverse enough to cover unseen tasks, the transfer performance is close to 0.

From the results, we can see that the transfer performance faces a relatively large drop when our approach transfers from tasks with few agents to those with more agents, and a failure will appear when testing on those hard tasks as the decision skills required in these tasks cannot be acquired by training on those easy tasks. Besides, when we try to transfer across different series of tasks, our approach struggles even when testing on the 2s3z task, which is a quite easy task. This phenomenon indicates that policy transfer across quite different tasks is still a quite hard problem that is worth exploring, and our approach may struggle when the target task can not be well covered by the source tasks. How to overcome this challenge and achieve more general policy transfer in multi-agent reinforcement learning remains an open problem.