Appendix A Details about Case Study

Problem definition. In case study (Sec. 3), we design a two-player navigation problem, which essentially is a grid-world problem. The whole map is a grid world with size $[15, 15]$, and in each episode two agents are randomly initialized in the centering initial region which is a 3×3 sub-area. The agents can observe local information within a sight range of 1 and are expected to reach a specific goal position for the current task. Each task corresponds to a specific goal position and there exist eight source tasks as marked in Fig. A1.

For the reward function, we define the reward r_t at each timestep t as the opposite of the L_1 distance from the agents to the target position:

$$r_t = -\sum_{i=1}^{2} \left[\texttt{abs}(agent\_pos_i[0] - goal\_pos[0]) + \texttt{abs}(agent\_pos_i[1] - goal\_pos[1])\right].$$

Application of MATTAR. Note that the transition functions are all the same in different tasks. To better capture the similarity and distinctions between tasks, we only optimize the reward prediction loss for forward model in the task representation learning step. The training phase is conducted on the eight source tasks, where the representation explainer and the agent policy are trained. When transferring to the unseen task, we learn the task representation by optimizing the reward prediction loss on the new task and insert the learned task representation into the trained policy network to obtain the transferred policy.

Two results in the main text. For Fig. 3(a), we conduct transfer phase for multiple times independently. Each time, we record the learned coefficients for the target task, and show the results in the form of a bar chart where each bar corresponds to a specific source task. Besides, each time we obtain a target task representation, we insert it into the representation explainer and obtain a forward model. We test the prediction loss of this forward model on eight source tasks and present the results in the form of a plot chart. Besides, note that the samples for computing model prediction error are collected with random policies. For Fig. 3(b), we first insert source task representations to the trained agent policy and roll out several trajectories. For simplicity of presentation, we only visualize the trajectories of agent 1 and selectively show trajectories for five source tasks ($1, 2, 4, 5$ and 8). We visualize trajectories for two independent runs. To avoid overlapping of arrows, we place an arrow at a random position within a grid to represent that the agent once passed this grid and the direction of arrow indicate the moving direction of agent. For the trajectory of the unseen target task, we visualize one trajectory and represent it with a dashed line.

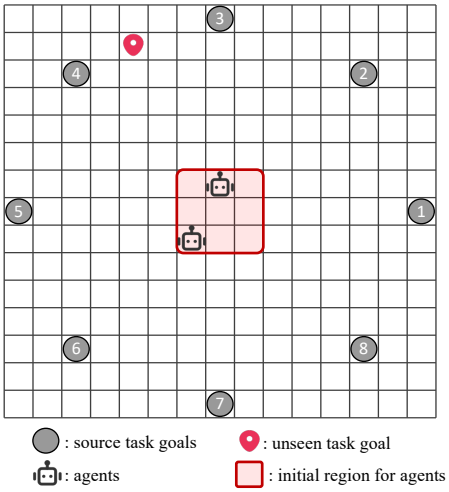


Figure A1 Tow-player Navigation problem for Case Study.

Appendix B Details about the Benchmarks

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

Figure B1 Snapshots of the experimental environments used in this paper.

SMAC (Fig. B1(a)∼(b)) StarCraft II Micromanagement Benchmark [5] 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. B1∼B3.

MPE (Fig. B1(c)∼(d)) Multi-Agent Particle Environment [3] 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. C2 and C3.

Table B1 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

Appendix C Network Architecture and Hyperparameters

Our implementation of MATTAR is based on PyMARL [1)] 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 RODE [7], ASN [8], QPLEX [6], QMIX [4], and UPDeT [1], we use the codes provided by the authors with the default hyperparameters settings. We describe our network structure in Tab. C1. This network architecture is used for all experiments in the paper.

1) We use PyMARL with SC2.4.6.2.6923. Performance is not always comparable among versions.

Table B2 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

Appendix 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∼3 are the mean and variance of these 5 random seeds. Specifically, for MATTAR we first train the agent policy and task explainer network on all source tasks. When it comes to the testing phase, we optimize the task representation on the new task using a small number of samples. Subsequently, we incorporate the task representation into the policy and perform rollout evaluation to assess the generalization performance on the unseen target task.

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. Similar to the generalization performance testing, the fine-tuning experiment is also decomposed of two stages: training phase and transfer phase. For the former, we adopt the same practice as that in the generalization performance testing; for the later, besides learning task representation on the new task, MATTAR additionally performs online fine-tuning of the agent policy.

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 series of 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. In this process, MATTAR firstly define task representations for each source task, and then train the agent policy on all source tasks. In other words, the multi-task learning is similar to the training phase in the previous two experiments except for that we do not train the task explainer network here.

Single-task learning For this experiment, we tested each baseline and ablation algorithm with 5 random seeds. In specific, for our algorithm MATTAR, we discard the task representation module and only retain the PIN network structure. Thus, this experiment is aimed to investigate the effect of PIN network structure on multi-agent cooperation learning.

Appendix E Implementation Details

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. E1(a)).

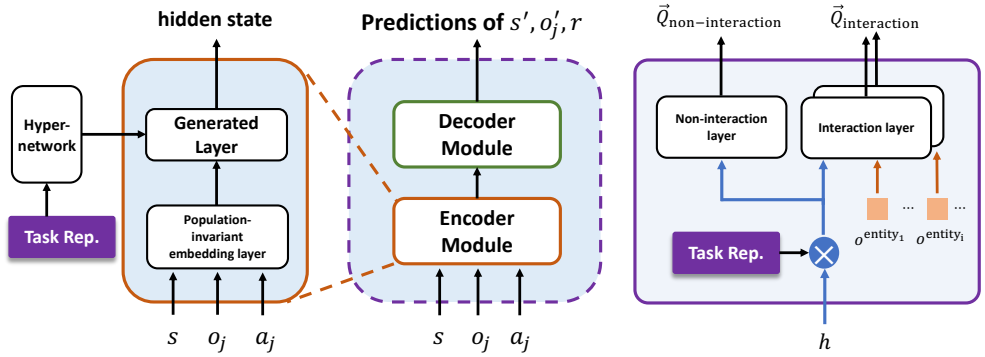
For the encoder network, we first use the population-invariant embedding layer to get a fixed-dimensional embedding vector and feed it into a fully-connected layer whose parameters are generated by the representation explainer. The output hidden variables are fed into the decoder to predict the next 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. The

Table B3 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 C1 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 E1 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.

Table C2 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

Table C3 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

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.

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. E1(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.

Appendix F The overall flow of MATTAR

Our proposed algorithm, MATTAR, consists of two stages, respectively the training phase and the transfer phase. In this section, we provide the pseudo-code of these two phases separately to provide a more detailed and clear description of our algorithm process, which are respectively show in Alg. F1 and Alg. F2.

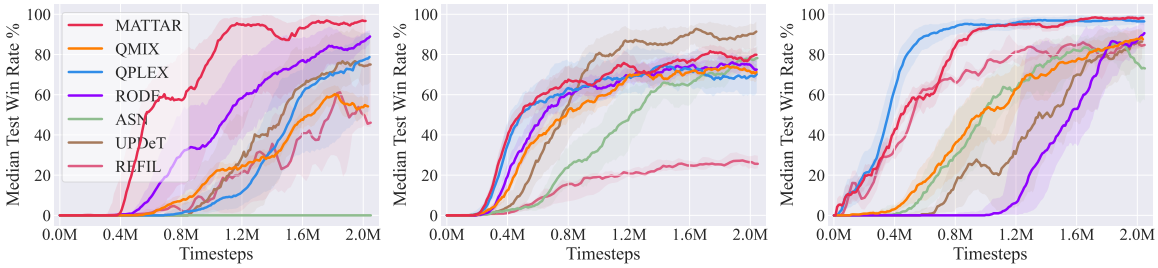
Appendix G Bonus: performance on single-task training

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 [4] and QPLEX [6]), a role-based learning algorithm (RODE) [7], and one underlying algorithm of MATTAR which considers the Q-values of interaction actions separately (ASN) [8]. 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 [1] and REFIL [2]).

Fig. G1 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

Algorithm F1 Training phase of MATTAR

Input: The source tasks $\{\mathcal{S}_i\}_{N_{src}}$, the learning rate α, initialized explainer network f_θ parameterized by θ, initialized PIN network Q_{tot} parameterized by ψ, empty replay buffers $\{\mathcal{D}_i\}_{N_{src}}$ for all source tasks $\{\mathcal{S}_i\}_{N_{src}}$
Initialize task representations $\{z_i\}_{N_{src}}$ for all source tasks $\{\mathcal{S}_i\}_{N_{src}}$ as orthogonal unit vectors
/*Below we train the representation explainer network f_θ*/
Set $k = 0$
for $k <$ rep_max_iteration **do**
  for each source task $\mathcal{S}_i$ **do**
    Roll out one trajectory τ by interacting with the environment of task $\mathcal{S}_i$ using uniform random policy, and add τ into the buffer $\mathcal{D}_i$
    Sample a batch of trajectories from buffer $\mathcal{D}_i$
    Compute the model prediction loss $J_{\mathcal{S}_i}(\theta)$ for source task $\mathcal{S}_i$
    Update representation explainer network parameters: $\theta \leftarrow \theta - \alpha \cdot \nabla_\theta J_{\mathcal{S}_i}(\theta)$
  end for
  Set $k = k + 1$
end for
Clear all the buffers $\{\mathcal{D}_i\}_{N_{src}}$
/*Below we train the agent policy network f_θ*/
Set $k = 0$
for $k <$ policy_max_iteration **do**
  for each source task $\mathcal{S}_i$ **do**
    Roll out one trajectory τ by interacting with the environment of task $\mathcal{S}_i$ using the current policy with ϵ-greedy method, and add τ into the buffer $\mathcal{D}_i$
    Sample a batch of trajectories $\mathcal{B}$ from buffer $\mathcal{D}_i$
    Compute the TD loss $\mathcal{L}_{\mathcal{S}_i}^{\text{TD}}(\psi) = \mathbb{E}_{(\tau,\boldsymbol{a},\tau',r)\sim\mathcal{B}}\left[\left(r + \gamma \max_{\boldsymbol{a}'} Q_{\text{tot}}(\tau', \boldsymbol{a}'; \psi^-) - Q_{tot}(\tau, \boldsymbol{a}; \psi)\right)^2\right]$
    Update the PIN network parameters $\psi \leftarrow \psi - \alpha \cdot \nabla_\psi \mathcal{L}_{\mathcal{S}_i}^{\text{TD}}(\psi)$
  end for
  Set $k = k + 1$
end for



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

Figure G1 A bonus: when learning from scratch on single tasks, MATTAR architecture exhibits good performance. For performance on more SMAC benchmark, please refer to Fig. G2.

on the super hard map `MMM2`. In Fig. G2, 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.

Appendix H Testing of one alternative for source task definition

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. H1. It is

Algorithm F2 Transfer phase of MATTAR

Input: The target unseen task $\mathcal{T}$, the learning rate α, the pre-trained explainer network f_θ parameterized by θ, task representations $\{z_i\}_{N_{src}}$ for the source tasks, the pre-trained PIN network Q_{tot} parameterized by ψ, empty replay buffer $\mathcal{D}$
Initialize one vector c with dimension of N_{src}, and initialize the task representation of the target unseen task $\mathcal{T}$ as $z = \sum_{i=1}^{N_{src}} \mu_i z_i$, where $\mu = \texttt{softmax}(c)$
/*Below we optimize the coefficient parameter c*/
Set $k = 0$
for $k <$ rep_max_iteration **do**
  Roll out one trajectory τ by interacting with the environment of task $\mathcal{T}$ using uniform random policy, and add τ into the buffer $\mathcal{D}$
  Sample a batch of trajectories from buffer $\mathcal{D}$
  Compute the model prediction loss $J_{\mathcal{T}}(\mu)$
  Update the coefficient parameter: $c \leftarrow c - \alpha \cdot \nabla_\mu J_{\mathcal{T}}(\mu) \nabla_c \mu$
  Set $\mu = \texttt{softmax}(c)$
  Set $k = k + 1$
end for
/*Below I conduct policy performance test or policy fine-tuning*/
if we need do policy fine-tuning **then**
  Clear the buffer $\mathcal{D}$
  for $k <$ finetuning_max_iteration **do**
    Roll out one trajectory τ by interacting with the environment of task $\mathcal{T}$ using uniform random policy, and add τ into the buffer $\mathcal{D}$
    Sample a batch of trajectories $\mathcal{B}$ from buffer $\mathcal{D}$
    Compute the TD loss $\mathcal{L}_{\mathcal{T}}^{\text{TD}}(\psi) = \mathbb{E}_{(\tau,\boldsymbol{a},\tau',r)\sim\mathcal{B}}\left[\left(r + \gamma \max_{\boldsymbol{a}'} Q_{\text{tot}}(\tau', \boldsymbol{a}'; \psi^-) - Q_{\text{tot}}(\tau, \boldsymbol{a}; \psi)\right)^2\right]$
    Update the PIN network parameters $\psi \leftarrow \psi - \alpha \cdot \nabla_\psi \mathcal{L}_{\mathcal{T}}^{\text{TD}}(\psi)$
  end for
  Conduct policy performance test
else
  We insert the obtained task representation $z = \sum_{i=1}^{N_{src}} \mu_i z_i$ into the PIN network, and conduct policy performance test
end if

Table I1 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

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.

Appendix 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. I1.
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

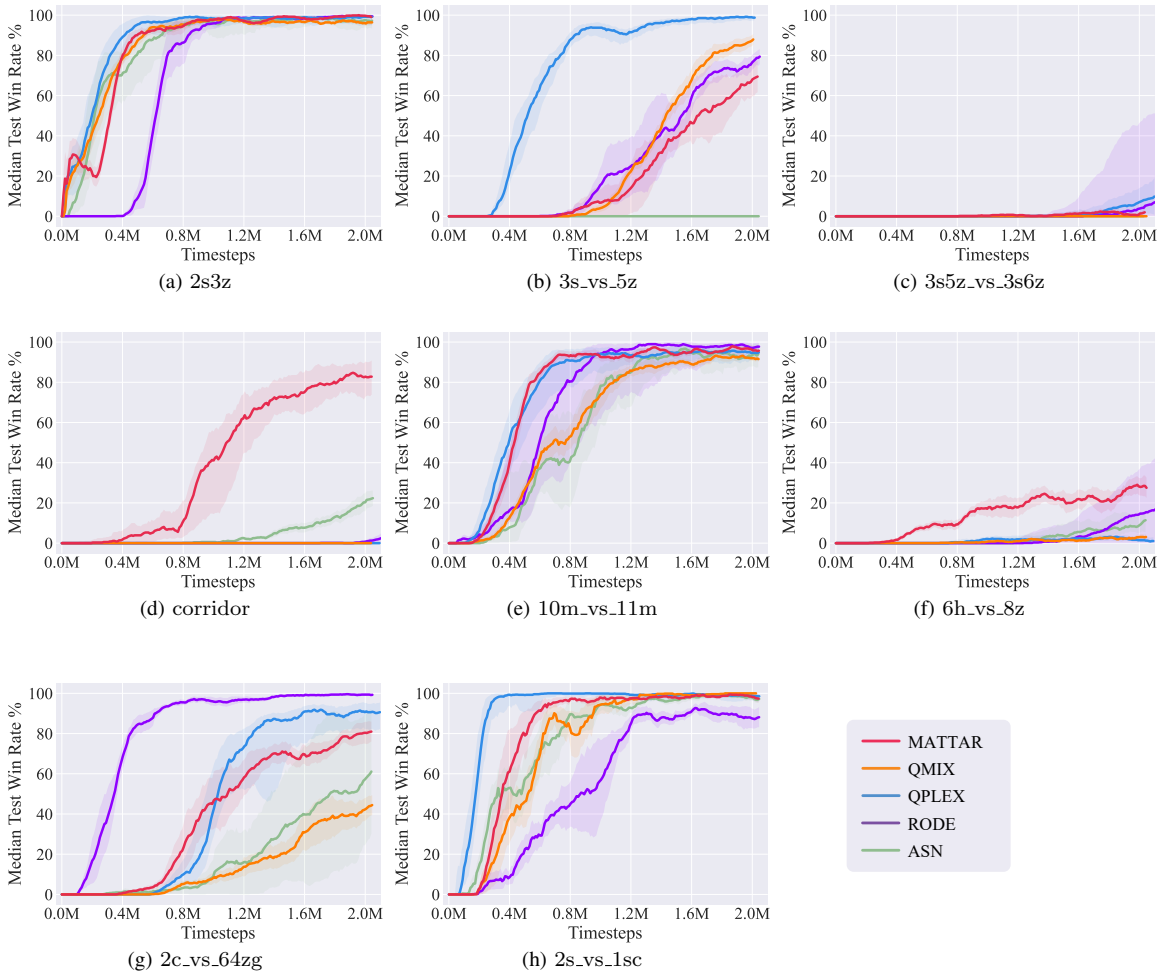


Figure G2 More results for the performance of MATTAR on the SMAC benchmark when learning from scratch on single tasks.

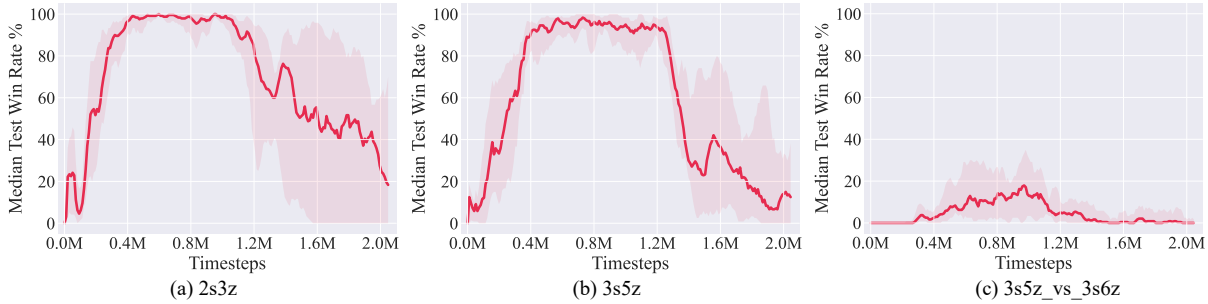


Figure H1 Testing of one alternative for source task definition. In this experiment, we apply a normalization layer on top of the joint learned vectors with the representation explainer, and use it as the task representations for source tasks.

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.

References

1 Siyi Hu, Fengda Zhu, Xiaojun Chang, and Xiaodan Liang. 2021. UPDeT: Universal multi-agent reinforcement learning via policy decoupling with transformers. In *International Conference on Learning Representations*.

2 Shariq Iqbal, Christian A Schroeder De Witt, Bei Peng, Wendelin Böhmer, Shimon Whiteson, and Fei Sha. 2021. Randomized Entity-wise Factorization for Multi-Agent Reinforcement Learning. In *International Conference on Machine Learning*. 4596–4606.

3 Ryan Lowe, Yi Wu, Aviv Tamar, Jean Harb, OpenAI Pieter Abbeel, and Igor Mordatch. 2017. Multi-agent actor-critic for mixed cooperative-competitive environments. In *Advances in Neural Information Processing Systems*. 6379–6390.

4 Tabish Rashid, Mikayel Samvelyan, Christian Schroeder Witt, Gregory Farquhar, Jakob Foerster, and Shimon Whiteson. 2018. QMIX: Monotonic Value Function Factorisation for Deep Multi-Agent Reinforcement Learning. In *International Conference on Machine Learning*. 4292–4301.

5 Mikayel Samvelyan, Tabish Rashid, Christian Schroeder de Witt, Gregory Farquhar, Nantas Nardelli, Tim GJ Rudner, Chia-Man Hung, Philip HS Torr, Jakob Foerster, and Shimon Whiteson. 2019. The StarCraft Multi-Agent Challenge. In *International Conference on Autonomous Agents and MultiAgent Systems*. 2186–2188.

6 Jianhao Wang, Zhizhou Ren, Terry Liu, Yang Yu, and Chongjie Zhang. 2021. QPLEX: Duplex Dueling Multi-Agent Q-Learning. In *International Conference on Learning Representations*.

7 Tonghan Wang, Tarun Gupta, Anuj Mahajan, Bei Peng, Shimon Whiteson, and Chongjie Zhang. 2021. RODE: Learning Roles to Decompose Multi-Agent Tasks. In *International Conference on Learning Representations*.

8 Weixun Wang, Tianpei Yang Yong Liu, Jianye Hao, Xiaotian Hao, Yujing Hu, Yingfeng Chen, Changjie Fan, and Yang Gao. 2020. Action Semantics Network: Considering the Effects of Actions in Multiagent Systems. In *International Conference on Learning Representations*.