Learning Task Embeddings for Teamwork Adaptation in Multi-Agent Reinforcement Learning

Lukas Schäfer
University of Edinburgh
l.schaefer@ed.ac.uk

Filippos Christianos
University of Edinburgh
f.christianos@ed.ac.uk

Amos Storkey
University of Edinburgh
a.storkey@ed.ac.uk

Stefano V. Albrecht
University of Edinburgh
s.albrecht@ed.ac.uk

Abstract

Successful deployment of multi-agent reinforcement learning often requires agents to adapt their behaviour. In this work, we discuss the problem of teamwork adaptation in which a team of agents needs to adapt their policies to solve novel tasks with limited fine-tuning. Motivated by the intuition that agents need to be able to identify and distinguish tasks in order to adapt their behaviour to the current task, we propose to learn *multi-agent task embeddings* (MATE). These task embeddings are trained using an encoder-decoder architecture optimised for reconstruction of the transition and reward functions which uniquely identify tasks. We show that a team of agents is able to adapt to novel tasks when provided with task embeddings. We propose three MATE training paradigms: independent MATE, centralised MATE, and mixed MATE which vary in the information used for the task encoding. We show that the embeddings learned by MATE identify tasks and provide useful information which agents leverage during adaptation to novel tasks.

1 Introduction

Multi-agent reinforcement learning (MARL) is a machine learning paradigm which enables multiple agents to concurrently learn behaviour from interactions with the environment as well as interactions with each other. MARL methods have become increasingly capable of learning complex behaviour [Berner et al., 2019, Vinyals et al., 2019], but their learned behaviours are usually highly task-specific. This can be desirable to maximise effectiveness in specific tasks, but limits the applicability in the real-world, which often requires the learned behaviour to be robust to small perturbations and changes in the environment [Dulac-Arnold et al., 2021, Akkaya et al., 2019].

Our work addresses the challenge of **teamwork adaptation** in which a team of agents is trained in a set of training tasks, and then has to adapt to novel, previously unseen testing tasks. As an example, consider a warehouse environment in which a team of agents must navigate in the warehouse to collect and deliver shelves with requested items (Figure 1). Tasks can vary in their warehouse layout, and agents need to adjust their team strategy depending on the layout of the warehouse to optimise deliveries. Whereas the warehouses with shelves on one side (Figure 1, left) require all agents to move to the same side of the warehouse, the warehouse with shelves on both ends (Figure 1, right) requires agents to effectively split between both ends to minimise waiting and travel time.

The above example illustrates that a team of agents may need to adapt their individual behaviour and coordination to novel tasks in a non-trivial way. However, such adaptation requires agents to

NeurIPS 2023 Workshop on Generalization in Planning (GenPlan 2023).

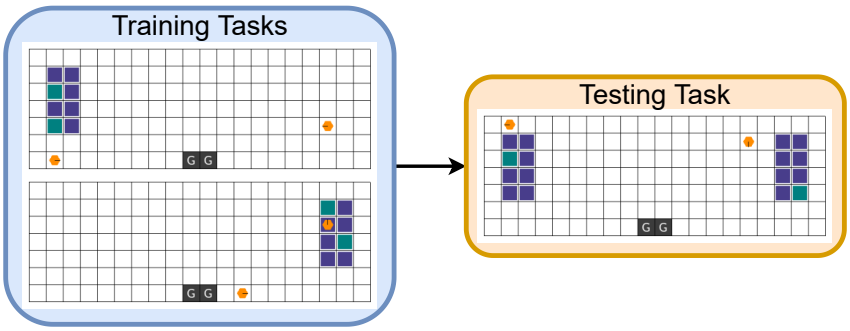


Figure 1: Teamwork adaptation in multi-robot warehouse environment: In the training tasks with different layouts, both agents (orange) need to deliver shelves (blue) with requested items (green) to the delivery zone (black). After training, the agents are fine-tuned and evaluated in the unseen testing task which requires novel coordination to effectively distribute the workload across both agents.

identify the task they are in. Current MARL approaches are not designed for such adaptation, and in our experiments we show their limitations at adapting to novel tasks.

Motivated by the challenge of teamwork adaptation, we propose to equip agents with the ability to infer the task they are in by learning **multi-agent task embeddings** (MATE). Through interacting with the environment, each agent builds up a task embedding using an encoder conditioned on the trajectories of agents in the task. Motivated by the observation that each task is uniquely identifiable by its transition and reward functions, the encoder is jointly trained with a separate decoder which conditioned on task embeddings is optimised to reconstruct transitions and rewards. Learned embeddings allow a team of agents to adapt to previously unseen but related tasks by conditioning their policies on task embeddings. We hypothesise that access to task embeddings simplifies teamwork adaptation. We consider and compare three paradigms of learning such embeddings: independent MATE, centralised MATE, and mixed MATE. For independent MATE, each agent independently encodes local information to obtain an embedding of the task, whereas centralised MATE trains a single encoder conditioned on the joint information for all agents. Mixed MATE trains independent encoders for agents but optimises them using a mixture of task embeddings with weights for each task embedding being conditioned on the state of the environment.

We empirically evaluate all MATE variants in four multi-agent environments which require increasing degrees of adaptation. We find that MATE improves teamwork adaptation in several cases compared to fine-tuning MARL agents without task embeddings or training agents in testing tasks with no prior training. In particular, in tasks where all agents need to coordinate, MATE provides useful information for agent adaptation, with mixed MATE performing the best among all three paradigms across a wide range of adaptation scenarios. We further demonstrate that learned task embeddings produce clusters which clearly identify tasks, and the learned mixture of mixed MATE focuses on task embeddings of agents which currently observe the most information about the task.

2 Related Work

Meta Reinforcement Learning: Meta reinforcement learning (Meta RL) aims to leverage concepts from meta learning [Hospedales et al., 2020] to learn policies which can adapt to novel testing tasks using limited interactions. Such adaptation can be achieved by computing meta gradients to find a model initialisation from which an effective policy in testing tasks can be obtained using few optimisation steps, e.g. using MAML [Finn et al., 2017] or REPTILE [Nichol et al., 2018]. Another common approach for meta RL implicitly builds up a latent context using recurrent neural networks which enable adaptation to new tasks [Duan et al., 2016, Wang et al., 2016, Fakoor et al., 2020]. However, these approaches rely on the recurrent context to learn useful information for adaptation without any explicit optimisation enforcing such usefulness. Rakelly et al. [2019] disentangle the objectives of task inference and learning control policies and leverage variational inference techniques to learn a context inference model. Zintgraf et al. [2020] also propose to learn a variational inference model to explicitly learn task beliefs and incorporate task uncertainty into the policies for adaptation. Their approach is similar in architecture and optimisation to MATE but is limited to the adaptation

of policies of individual agents. In contrast, we address the challenge of adapting the policies of multiple agents cooperating in a team.

Transfer Learning: Established techniques from transfer learning [Taylor and Stone, 2009] address a similar challenge to meta RL by extracting representations, action selection or other components from already learned models to improve the capabilities of agents in novel testing tasks. Unlike meta RL which typically adapts to novel tasks using fine-tuning over few episodes, transfer learning requires a dedicated transferring procedure for each new task. Da Silva and Costa [2019] provide an overview specifically for methods aiming to transfer policies of multiple agents. They distinguish between inter-agent transfer, aiming to leverage information from one agent with possibly more expertise to transfer another agent [Da Silva et al., 2017, 2018], and intra-agent transfer which aims to transfer knowledge across tasks for all involved agents. Related to transfer learning are approaches focusing on multi-task learning in which agents are trained to generalise over a set of multiple tasks [Vithayathil Varghese and Mahmoud, 2020, Omidshafiei et al., 2017, Caruana, 1997].

Multi-Agent Reinforcement Learning: There has been significant progress in MARL to solve challenging coordination problems [Papoudakis et al., 2021b] with many approaches focusing on the paradigm of centralised training and decentralised execution (CTDE) [Christianos et al., 2020, Rashid et al., 2018, Foerster et al., 2018, Sunehag et al., 2018, Lowe et al., 2017]. Under this paradigm, information is shared across agents at training time without conditioning agent policies on such joint information. In this way, training can leverage privileged information without preventing agent policies from being deployed in a decentralised manner. We train MATE using the reconstruction of transition and reward functions of tasks defined over all agents, so MATE also falls under the paradigm of CTDE. In multi-agent systems, it is appealing to explicitly learn models of the behaviour of other agents in the environment to infer their possible behaviour [Papoudakis et al., 2021a, Zintgraf et al., 2021, Albrecht and Stone, 2018]. Such approaches demonstrate the ability for a single agent within a multi-agent system to adapt to different agents to cooperate or compete with. Similarly, the challenges of ad hoc teamwork [Stone et al., 2010] and zero-shot coordination [Hu et al., 2020] address the problem of training agents to be able to coordinate with new partners without prior training in a team or established team strategies. Approaches in these settings prominently model other agents [Rahman et al., 2021, Barrett and Stone, 2015] or learn policies to effectively play alongside a diverse population of agents maintained during training [Lupu et al., 2021, Qiu et al., 2023]. However, few work exist which address the challenge of adapting the policies of multiple agents to novel tasks. Hu et al. [2021] propose a novel architecture leveraging transformer models and self-attention [Vaswani et al., 2017] to be able to reuse policies for varying team sizes, and Zhang et al. [2021] train a latent model to encode coordination information to generalise to varying team sizes. Vezhnevets et al. [2020] train agents using a hierarchical approach with a high-level policy choosing the low-level behaviour which should be deployed given currently available information about the task and other agents in the environment. Liang et al. [2022] apply the idea of REPTILE [Nichol et al., 2018] to MARL, finding an effective initialisation from training agent policies on multiple training tasks which is shown to generalise to new testing tasks. Mahajan et al. [2022] formalise the problem of combinatorial generalisation in which a team of agents needs to generalise over varying capabilities of agents. In contrast to all these approaches, we focus on the problem of adapting a fixed team of agents to varying tasks.

3 Problem Definition

Partially-Observable Stochastic Games We consider multi-agent tasks modelled as partially observable stochastic games (POSGs) for N agents [Hansen et al., 2004]. A POSG is given by the tuple $(\mathcal{I}, \mathcal{S}, \{\mathcal{O}^i\}_{i\in\mathcal{I}}, \{A^i\}_{i\in\mathcal{I}}, \Omega, \mathcal{P}, \{R^i\}_{i\in\mathcal{I}})$. Agents are indexed by $i \in \mathcal{I} = \{1, \dots, N\}$. $\mathcal{S}$ denotes the state space of the environment and $\mathcal{A} = A^1 \times \dots \times A^N$ denotes its joint action space. $\mathcal{P} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \mapsto [0, 1]$ denotes the transition function of the environment, defining a distribution of successor states given the current state and the applied joint action. At each time step t, each agent i receives an observation $o^i_t \in \mathcal{O}^i$ defined by the observation function $\Omega : \mathcal{S} \times \mathcal{A} \mapsto \Delta(\mathcal{O})$ conditioned on the current state and applied joint action for joint observation space $\mathcal{O} = \mathcal{O}^1 \times \dots \times \mathcal{O}^N$. Each agent learns a policy $\pi_i(a^i_t|o^i_{1:t})$ conditioned on its history of observations, denoted by $o^i_{1:t} = (o^i_1, \dots, o^i_t)$. After time step t, agent i receives a reward r^i_t given by its reward function $R^i : \mathcal{S} \times \mathcal{A} \mapsto \mathbb{R}$. The objective is to learn a joint policy $\boldsymbol{\pi} = (\pi_1, \dots, \pi_N)$ such that the discounted returns of each agent $G^i = \sum_{t=1}^{\infty} \gamma^{t-1} r^i_t$ are maximised with respect to the policies of all other agents, formally $\forall_i : \pi_i \in \arg\max_{\pi'_i} \mathbb{E}\left[G^i|\pi'_i, \pi_{-i}\right]$, with discount factor $\gamma \in [0; 1)$ and $\pi_{-i} = \boldsymbol{\pi} \setminus \{\pi_i\}$.

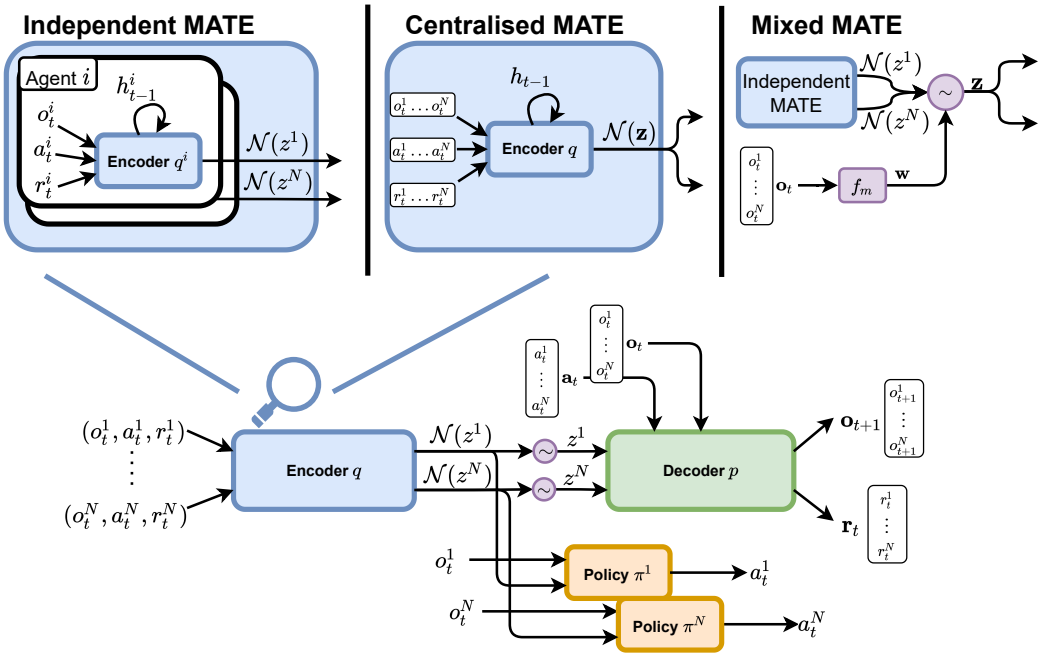


Figure 2: MATE recurrently encodes (blue) trajectories of states, actions and rewards into variational task embeddings. We consider three paradigms of MATE: independent MATE, centralised MATE and mixed MATE. Policies (orange) are conditioned on task embeddings and the decoder (green) receives sampled (purple) task embeddings, current observations and actions to predict the transition and reward function of all agents.

Teamwork Adaptation We consider the challenge of transferring the joint policies $\boldsymbol{\pi}$ of a fixed team of agents from a set of training tasks $\mathcal{T}_{\text{train}}$ to novel testing tasks $\mathcal{T}_{\text{test}}$ with $\mathcal{T}_{\text{train}} \cap \mathcal{T}_{\text{test}} = \emptyset$. Each task is represented as a POSG. Without any further assumptions, training and testing tasks could be arbitrarily different and hence no generalisation could be feasibly expected. We assume that training and testing tasks have identical number of agents N, action space $\mathcal{A}$ and identical dimensionality of observations. In our problem setting, agents are first trained in $\mathcal{T}_{\text{train}}$ for N_{train} time steps and can be trained in $\mathcal{T}_{\text{test}}$ for a limited number of time steps N_{test} to fine-tune the policy. We refer to this setting as teamwork adaptation which is similar to the problem of domain adaptation for RL [Eysenbach et al., 2021].

4 Learning Multi-Agent Task Embeddings

For agents to be able to adapt their behaviour to the current task, they need to infer the task they are in as they interact with their environment. Therefore, we propose to learn multi-agent task embeddings (MATE) $\mathcal{N}(\mathbf{z})$ from agent trajectories, consisting of observations, actions and rewards, which provide agent policies $\pi_i(a_t^i | o_{1:t}^i, \mathcal{N}(\mathbf{z}))$ with information about the task as well as a measure of uncertainty about the task. We hypothesise that explicitly modelling and learning task embeddings is advantageous for teamwork adaptation.

In order to learn task embeddings, we train a variational encoder-decoder architecture [Kingma and Welling, 2013]. Agents use an encoder q to compute a variational distribution $q(\mathbf{z}|\boldsymbol{\tau}_{1:t}) = \mathcal{N}(\mathbf{z}; \boldsymbol{\mu}, \text{diag}(\boldsymbol{\sigma}))$ representing the task embeddings. These task embeddings are given by a diagonal Gaussian distribution with mean $\boldsymbol{\mu} \in \mathbb{R}^d$ and variance $\boldsymbol{\sigma} \in \mathbb{R}^d_+$. We concisely denote this distribution over task embeddings as $\mathcal{N}(\mathbf{z})$. The encoder is represented as a recurrent neural network which outputs $\boldsymbol{\mu}$ and $\log(\text{diag}(\boldsymbol{\sigma}))$. At each time step, information about the trajectory consisting of observations, actions and rewards is fed into the encoder together with the previous hidden state to update the task embeddings. The hidden state is reset after each episode to enable training on multiple tasks sampled for each episode. In order to train the encoder, we define a decoder p to reconstruct the transition and reward functions across all tasks conditioned on task embedding samples $\mathbf{z} \sim \mathcal{N}(\mathbf{z})$. The encoder and decoder, parameterised by $\boldsymbol{\phi}$ and $\boldsymbol{\psi}$ respectively, are jointly optimised to maximise

the following evidence lower bound (ELBO) given trajectory $\boldsymbol{\tau}_{1:t}$

$$\text{ELBO}(\boldsymbol{\phi}, \boldsymbol{\psi}|\boldsymbol{\tau}_{1:t}) = \mathbb{E}_{q(\mathbf{z}_t|\boldsymbol{\tau}_{1:t};\boldsymbol{\phi})}\left[\log p(\mathbf{o}_{t+1}, \mathbf{r}_t|\mathbf{o}_t, \mathbf{a}_t, \mathbf{z}; \boldsymbol{\psi})\right] - \beta\text{KL}\left(q\left(\mathbf{z}|\boldsymbol{\tau}_{1:t}; \boldsymbol{\phi}\right)||p(\mathbf{z})\right) \quad (1)$$

with additional hyperparameter β to control the regularisation of the KL prior [Higgins et al., 2017]. This objective is motivated by the observation that each task can be identified by its unique transition and reward functions. Modelling the decoder as a multivariate Gaussian model over observations and rewards with constant diagonal covariance matrix and assuming a standard Gaussian prior allows us to minimise the following loss, equivalent to maximising the ELBO (see Appendix A for derivation):

$$\mathbb{L}(\boldsymbol{\phi}, \boldsymbol{\psi}|\boldsymbol{\tau}_{1:t}) = \mathbb{E}_{q(\mathbf{z}_t|\boldsymbol{\tau}_{1:t};\boldsymbol{\phi})}\Big[\left(p(\mathbf{o}_{t+1}|\mathbf{o}_t, \mathbf{a}_t, \mathbf{z}; \boldsymbol{\psi}) - \mathbf{o}_{t+1}\right)^2 + \left(p(\mathbf{r}_t|\mathbf{o}_t, \mathbf{a}_t, \mathbf{z}; \boldsymbol{\psi}) - \mathbf{r}_t\right)^2\Big] - \beta\frac{1}{2}\sum_{j=1}^{d}\left(1 + \log\left(\sigma_j^2\right) - \mu_j^2 - \sigma_j^2\right) \quad (2)$$

The intuition of this loss is that the decoder p is optimised to model the mean of the generative multivariate Gaussian distribution over reconstructed observations and rewards. We consider three paradigms of learning MATE: (1) independent MATE, (2) centralised MATE and (3) mixed MATE which vary in the information used to encode task embeddings.

Independent Multi-Agent Task Embeddings (Ind-MATE) independently trains separate encoders $q^i(z^i|\tau_{1:t}^i; \phi^i)$ with $\tau_{1:t}^i = \{(o_u^i, a_u^i, r_u^i)\}_{u=1}^t$ for each agent i conditioned only on its individual trajectory. The centralised decoder is shared across all agents and used to decode individual task embeddings of all agents, and the policy $\pi_i(a_t^i|o_t^i, \mathcal{N}(z^i))$ of agent i is conditioned only on its individual task embedding. We hypothesise that such task embeddings are limited in the encoded information which cannot fully represent the task under partial observability.

Centralised Multi-Agent Task Embeddings (Cen-MATE) instead shares a single encoder $q(\mathbf{z}|\boldsymbol{\tau}_{1:t}; \boldsymbol{\phi})$ with $\boldsymbol{\tau}_{1:t} = \{(\mathbf{o}_u, \mathbf{a}_u, \mathbf{r}_u)\}_{u=1}^t$ and decoder conditioned on the joint information across all agents. The policy of each agent i is conditioned on the joint, shared task embedding $\mathcal{N}(\mathbf{z})$. Such a shared, centralised task embedding has access to information from all agents and can therefore encode more information about the task than Ind-MATE. However, policies depend on the computation of the task embedding which requires access to the joint information across all agents. The access to this privileged information prevents decentralised execution of agents.

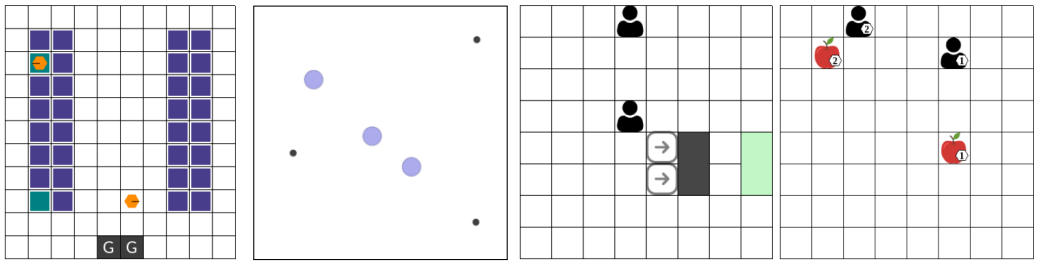
Mixed Multi-Agent Task Embeddings (Mix-MATE) is similar to Ind-MATE in that each agent trains an individual encoder conditioned only on its local trajectory to ensure decentralised execution. However, instead of decoding each variational task embedding independently, a mixture of task embeddings is computed $q(\mathbf{z}|\boldsymbol{\tau}_{1:t}; \boldsymbol{\phi}) = \sum_{i=1}^N w_i q^i(z^i|\tau_{1:t}^i; \phi^i)$ and sampled from. Mixture weights $\mathbf{w} = f_m(\mathbf{o}_t)$ are computed using a single-layer network with softmax output conditioned on the joint observations. We hypothesise that such mixing allows individual task embeddings to be more representative of the full task while preserving decentralised execution. We also note that the mixture distribution provides insight into which agent's task embedding is considered most important in a given state.

5 Experimental Evaluation

In this section, we evaluate our proposed approach of learning multi-agent task embeddings. In particular, we will investigate (1) whether MATE improves teamwork adaptation given by returns in testing tasks after limited fine-tuning, (2) how the three paradigms of MATE compare to each other, and (3) what information is encoded by MATE. To answer these questions, we conduct an evaluation in four multi-agent environments, visualised in Figure 3. In all experiments, we train agents using the multi-agent synchronous advantage actor-critic (MAA2C) [Papoudakis et al., 2021b, Mnih et al., 2016] algorithm with recurrent policies containing gated recurrent units (GRU) [Cho et al., 2014]. During fine-tuning, we freeze MATE encoders and only fine-tune policies and critics. For further implementation details, see Appendix C, and details for each task can be found in Appendix B.

5.1 Multi-Agent Environments

Multi-Robot Warehouse: In our motivational example of the multi-robot warehouse (RWARE) [Papoudakis et al., 2021b, Christianos et al., 2020] agents need to collect and deliver requested items



(a) Multi-robot warehouse (RWARE) (b) Multi-agent particle (MPE) (c) Boulder-push (BPUSH) (d) Level-based foraging (LBF)

Figure 3: Visualisations of all four multi-agent environments

from shelves within warehouses. Each agent observes a 5×5 grid centred around the agent containing information about nearby shelves, agents and delivery zones. Agents can move forward, pick-up shelves, and rotate which also rotates their observation. Tasks include warehouses with two or four agents, and varying layouts which lead to static, but significant variation requiring adaptation in coordination behaviour outlined in Figure 1. Training and testing sets include multiple warehouses of similar layout but a range of sizes. Each episode consists of 500 time steps.

Multi-Agent Particle Environment: In the cooperative navigation task of the multi-agent particle environment (MPE) [Mordatch and Abbeel, 2018, Lowe et al., 2017], three agents need to navigate a continuous two-dimensional, fully-observable world to cover all three landmarks which are initialised in random locations while avoiding collisions with each other. In the training task, agents only receive a small punishment of -1 for collisions with each other whereas testing tasks punish agents with significantly higher negative rewards of -5 and -50, respectively. Episodes terminate after 25 time steps.

Boulder-Push: In this new environment (BPUSH), agents navigate a gridworld and need to push a box towards a goal location. Agents observe the box and agents in a 9×9 grid and the direction the box is required to be pushed in. These tasks require significant coordination with agents only receiving rewards for successfully pushing the box forward which requires cooperation of all agents. Episodes terminate after the box has been pushed to its goal location or after at most 50 time steps. In the training task, two agents need to push a box in a small 8×8 gridworld. Testing tasks include tasks with gridworld sizes of 12×12 (medium) and 20×20 (large) as well as tasks with a small negative penalty of -0.01 for unsuccessful pushing attempts of agents. Tasks with penalty terms make the exploration of the optimal behaviour of only pushing the box together with the other agent significantly harder.

Level-Based Foraging: In the level-based foraging (LBF) environment [Albrecht and Ramamoorthy, 2013, Albrecht and Stone, 2018], multiple agents need to coordinate in a gridworld to pick-up food. Food and agents are assigned levels with agents only being able to pick-up adjacent food if the sum of levels of all agents cooperating to pick-up is greater or equal to the level of the food. Agents observe a 5×5 grid centred on themselves and are rewarded for picking-up food depending on its level and their contribution. Episodes terminate after all food has been collected or after at most 50 time steps. We train in two comparably simple LBF tasks with gridworld sizes of 8×8 and two and four agents, respectively. Testing tasks for two and four agents contain a task with gridworld size 10×10 in which each food will require at least two agents to coordinate to pick it up, tasks with larger gridworld size of 15×15 and tasks with a small penalty of -0.1 for unsuccessful picking attempts of food.

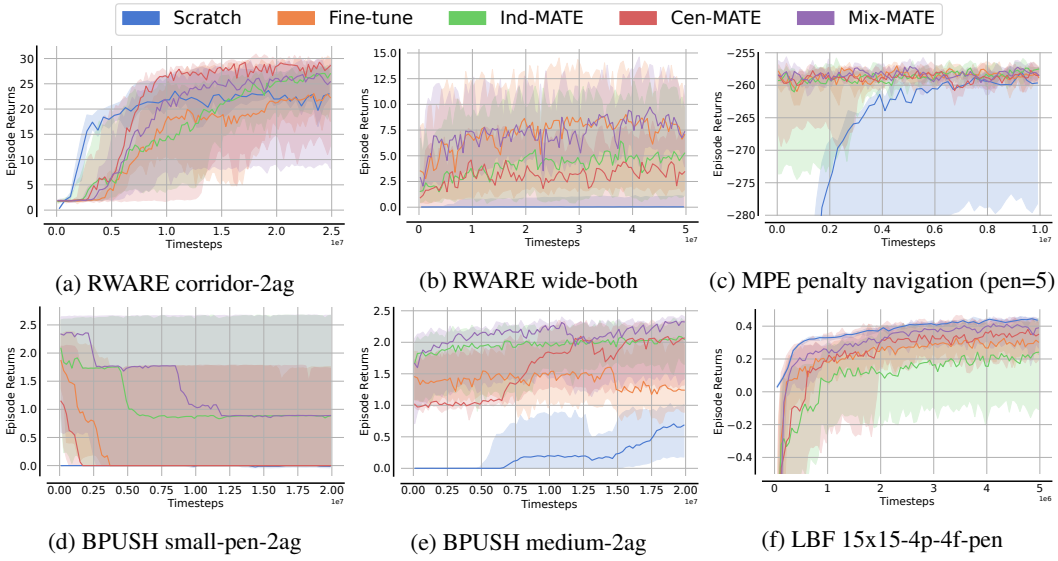
5.2 Baselines

No MATE (fine-tune): For the fine-tune baseline, we train MARL agents without any task embeddings in the training task and fine-tune in testing tasks for further N_{test} time steps. This baseline allows us to evaluate the benefits of task embeddings for teamwork adaptation.

No MATE (scratch): For the scratch baseline, we train MARL agents without any task embeddings only in the testing tasks for N_{test} time steps. This baseline allows us to distinguish adaptation settings where prior training in potentially simpler tasks can improve returns in testing tasks from cases of negative transfer where prior training hurts the performance of agents in the testing tasks.

Table 1: Fine-tuning performance given by IQM and standard deviation across final returns over five random seeds. Highest IQM per testing task (within one standard deviation) are shown in bold.

	$\mathcal{T}_{\text{train}}$ (N_{train})	$\mathcal{T}_{\text{test}}$ (N_{test})	Scratch	Fine-tune	Ind-MATE	Cen-MATE	Mix-MATE
RWARE	tiny-2ag (25M)	small-2ag (25M)	0.02	$\mathbf{8.54 \pm 2.29}$	5.97 ± 3.41	$\mathbf{10.90 \pm 2.97}$	$\mathbf{7.96 \pm 4.11}$
		corridor-2ag (25M)	$\mathbf{23.00 \pm 2.91}$	$\mathbf{22.26 \pm 5.58}$	$\mathbf{26.98 \pm 4.06}$	$\mathbf{28.59 \pm 7.10}$	$\mathbf{25.39 \pm 10.17}$
	tiny-4ag (25M)	small-4ag (25M)	0.12 ± 0.87	$\mathbf{28.48 \pm 0.92}$	26.61 ± 1.14	26.58 ± 1.32	$\mathbf{27.69 \pm 1.21}$
		corridor-4ag (25M)	$\mathbf{50.13 \pm 1.84}$	42.75 ± 3.70	43.12 ± 2.31	45.53 ± 2.42	43.96 ± 3.30
	wide-both (50M)	wide-one-sided (50M)	$\mathbf{12.39 \pm 6.75}$	0.03 ± 3.49	0.02	3.08 ± 4.50	0.02
	wide-one-sided (50M)	wide-both (50M)	0.04 ± 0.62	$\mathbf{7.06 \pm 3.95}$	$\mathbf{5.26 \pm 3.69}$	3.45 ± 3.11	$\mathbf{7.34 \pm 3.39}$
MPE	cooperative navigation (10M)	penalty navigation, pen=5 (10M)	$\mathbf{-259.65 \pm 11.09}$	-258.81 ± 1.13	$\mathbf{-257.50 \pm 0.31}$	-258.47 ± 0.75	-258.45 ± 0.68
		penalty navigation, pen=50 (10M)	-2019.35 ± 3.95	$\mathbf{-2016.83 \pm 3.54}$	-2019.20 ± 3.79	-2020.60 ± 4.39	$\mathbf{-2015.15 \pm 2.01}$
BPUSH	small-2ag (5M)	small-pen-2ag (20M)	$\mathbf{-0.01 \pm 0.01}$	$\mathbf{-0.00 \pm 1.05}$	$\mathbf{0.88 \pm 1.30}$	$\mathbf{-0.00 \pm 1.03}$	$\mathbf{0.89 \pm 1.30}$
		medium-pen-2ag (20M)	$\mathbf{-0.01 \pm 0.01}$	$\mathbf{-0.00}$	$\mathbf{-0.00}$	$\mathbf{0.00}$	$\mathbf{0.00}$
		medium-2ag (20M)	0.69 ± 0.36	1.24 ± 0.47	$\mathbf{2.04 \pm 0.29}$	$\mathbf{2.07 \pm 0.46}$	$\mathbf{2.32 \pm 0.52}$
		large-2ag (20M)	0.00	0.39 ± 0.10	$\mathbf{0.60 \pm 0.16}$	0.43 ± 0.14	$\mathbf{0.97 \pm 0.40}$
LBF	8x8-2p-2f (5M)	10x10-2p-2f-coop (5M)	$\mathbf{0.88 \pm 0.02}$	0.80 ± 0.03	$\mathbf{0.84 \pm 0.04}$	0.81 ± 0.03	$\mathbf{0.84 \pm 0.04}$
		15x15-2p-4f (5M)	$\mathbf{0.71 \pm 0.03}$	$\mathbf{0.67 \pm 0.04}$	$\mathbf{0.73 \pm 0.08}$	$\mathbf{0.71 \pm 0.01}$	$\mathbf{0.71 \pm 0.02}$
		15x15-2p-2f-pen (5M)	$\mathbf{0.70}$	$\mathbf{0.64 \pm 0.08}$	0.55 ± 0.06	0.65 ± 0.04	0.56 ± 0.12
	8x8-4p-4f (5M)	10x10-4p-2f-coop (5M)	$\mathbf{0.78 \pm 0.01}$	$\mathbf{0.73 \pm 0.05}$	0.74 ± 0.03	0.72 ± 0.02	0.74 ± 0.02
		15x15-4p-6f (5M)	$\mathbf{0.73}$	0.66 ± 0.03	$\mathbf{0.71 \pm 0.02}$	0.69 ± 0.02	$\mathbf{0.71 \pm 0.03}$
		15x15-4p-4f-pen (5M)	$\mathbf{0.44}$	0.30 ± 0.08	$\mathbf{0.24 \pm 0.24}$	$\mathbf{0.35 \pm 0.10}$	$\mathbf{0.39 \pm 0.07}$



(a) RWARE corridor-2ag (b) RWARE wide-both (c) MPE penalty navigation (pen=5)

(d) BPUSH small-pen-2ag (e) BPUSH medium-2ag (f) LBF 15x15-4p-4f-pen

Figure 4: IQM and 95% confidence intervals for fine-tuning returns in selected testing tasks.

We further highlight that the policies of all agents, including both baselines, contain GRU networks such that an implicit task embedding can be learned through the hidden state of the recurrent network.

5.3 Evaluation Results

We report episodic returns as the sum of episodic returns of all agents. We train and fine-tune each algorithm for five random seeds and report the interquartile mean (IQM) [Agarwal et al., 2021] and standard deviation across final fine-tuning returns over all seeds in Table 1. Learning curves with IQM and stratified bootstrap 95% confidence intervals computed over 100,000 samples are shown in Figure 4 for some testing tasks. Learning curves for all training and testing scenarios can be found in Appendices D and E.

In RWARE (Figure 14), agents need to heavily explore in order to receive positive rewards as they need to execute long sequences of actions to deliver requested items. Therefore, agents significantly benefit from the initial training phase during fine-tuning for testing tasks with larger or more complicated environments. In particular, in small warehouse instances and wide warehouses with shelves on both sides (Figure 4b) we find the fine-tuning of agents with and without MATE improves returns significantly compared to training from scratch. In contrast, in simpler environments such as the corridor warehouses (Figure 4a) agents appear to suffer from the initial training phase starting with policies which exhibit less exploration than randomly initialised policies with entropy regularisation. For MATE, we find that Cen-MATE with access to observations of all agents performs the best in most tasks due to significant partial observability in these tasks.

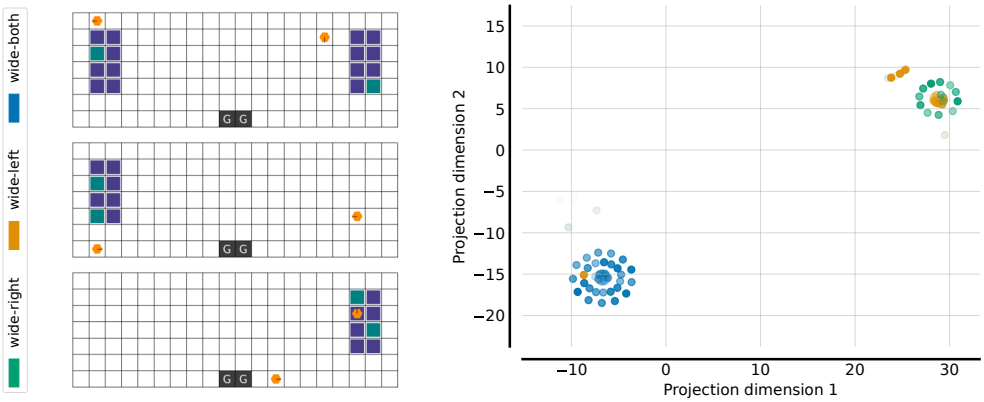


Figure 5: We collect task embeddings across multiple warehouse environments for a single episode using Cen-MATE trained in one-sided warehouses. We project the task embeddings with t-SNE into two dimensions and visualise embeddings across episodes with colours representing warehouse layouts and opacity indicating time steps within the episodes (more opaque for later time steps).

Fine-tuning agents in MPE (Figure 17) from simple spread to tasks with larger collision penalties appears to lead to no significant benefits over training agents from scratch. Agent policies remain almost identical throughout fine-tuning for both testing tasks, exhibiting almost identical returns at the beginning and end of fine-tuning.

For BPUSH (Figure 16), we find MATE significantly outperforms all baselines in three out of four fine-tuning scenarios. In particular Mix-MATE outperforms all other approaches. Adding a penalty to unsuccessful pushing attempts of agents makes the exploration of the optimal behaviour significantly more challenging, so training agents from scratch is unsuccessful. Agents previously trained on the simpler task without such a penalty perform well early during fine-tuning, but throughout fine-tuning agents exhibit more defensive behaviour and refrain from pushing the box to avoid penalties leading to decreasing overall returns. We verified that this behaviour was consistent even for lower entropy regularisation coefficient 10^{-4} for reduced penalty-inducing exploration. We find Ind-MATE and Mix-MATE are significantly more robust at informing agents about the task and allow them to adapt their behaviour by limiting unsuccessful pushing attempts. We hypothesise that Cen-MATE is less successful in these tasks due to its worse sample efficiency exhibiting lower returns at the end of the initial training phase. Agents fine-tuned with MATE are also better at adapting to tasks with larger gridworlds, reaching higher returns in particular for Mix-MATE and Ind-MATE.

In LBF (Figure 15), we find that the majority of tasks require comparably little adaptation with agents reaching similar returns at the beginning and end of fine-tuning. We hypothesise that the high degree of variability within individual LBF tasks, through agent and food placement as well as level allocation, allows agents to already learn most capabilities required for testing tasks during the initial training phase. In tasks with penalty for unsuccessful picking-up of food, significant adaptation of behaviour is required but policies obtained after the initial training phase tend to frequently attempt to pick-up food. This behaviour results in lower returns and consequentially negative initial transfer performance.

Overall, we find MATE to improve teamwork adaptation in several cases compared to both baselines. In particular in the BPUSH environment where all agents need to coordinate, we find MATE consistently improves returns during fine-tuning. Across all three paradigms of MATE, we find Cen-MATE to perform the best in environments with significant partial observability such as most RWARE tasks. However, training the shared encoder on the larger joint observations of all agents makes training with Cen-MATE less sample efficient leading to worse returns at the end of fine-tuning in several BPUSH tasks and the corridor task with four agents (Figure 14d). Lastly, we find the mixture of task embeddings appears to improve adaptation with Mix-MATE outperforming Ind-MATE in almost all fine-tuning tasks. Due to these benefits, we recommend to learn task embeddings using Mix-MATE whenever decentralised execution is of importance. For testing tasks with significant partial observability, Cen-MATE should be considered for its privileged encoded information but prevents decentralised deployment of agents.

Figure 6: Visualisation of Mix-MATE mixing weights for 25 time steps in the wide-left warehouse environment (left). At time step 10 (right top), agent 1 is observing the shelves in the warehouse while agent 2 does not observe any shelves so higher weighting is assigned to its respective embedding. At time step 17 (right bottom), agent 2 now observes a larger part of the shelves which results in higher weights on the embedding of agent 2.

5.4 Learned Multi-Agent Task Embeddings

After evaluating the benefits of MATE for teamwork adaptation, we analyse the task embeddings learned by MATE to gain a deeper understanding for the proposed method. First, we address the questing whether task embeddings encode meaningful information to identify tasks. Figure 5 visualises the t-SNE [Van der Maaten and Hinton, 2008] projection of task embeddings obtained using Cen-MATE for both one-sided wide warehouses as well as the wide warehouse with shelves on both sides. More specifically, we execute a policy trained without MATE in one-sided warehouses and sequentially encode the trajectory using Cen-MATE trained in the same one-sided warehouses. Figure 5 clearly illustrates separate clusters of embeddings for different tasks. Whereas the warehouse with shelves on both sides (blue) is clearly separated from one-sided warehouses, embeddings for both symmetric one-sided warehouses are close together. However, embeddings of one-sided warehouses are also separated with embeddings for the wide-left (orange) warehouse concentrated within a ring of embeddings for the wide-right (green) warehouse. Note that these embeddings were not trained in the wide-both warehouse task, demonstrating that MATE is able to generalise to novel tasks and produce task embeddings which identify tasks not encountered during training.

In Section 5, we find Mix-MATE to consistently outperform Ind-MATE for teamwork adaptation. It appears the mixture model significantly improves the impact of learned task embeddings on teamwork adaptation. Figure 6 visualises mixture weights for Mix-MATE in the wide-left warehouse environment. The mixing has no significant emphasis on the embedding of either agent until the first agent reaches the block of shelves on the left. The discovery and encoding of this task-relevant information leads to a rise in weighting on task embedding of the first agent. This importance ends after the second agent also reaches the block of shelves. After both agents discovered the shelves identifying the task, the mixing appears to again focus on the first agent which has loaded a shelf with requested items.

6 Conclusion

In this work, we discussed the problem of teamwork adaptation in which a team of agents needs to adapt their policies to solve novel tasks with limited fine-tuning. We presented three paradigms to learn multi-agent task embeddings (MATE), which recurrently encode information about the current task identified by its unique transition and reward functions. Our experiments in four multi-agent environments demonstrated that MATE provides useful information for teamwork adaptation, with most significant improvements in tasks where all agents need to coordinate. Our analysis of learned task embeddings shows that they form clusters which identify tasks. Among the three paradigms proposed to learn MATE, we find mixed MATE to perform the best across a wide range of testing tasks. Learned mixtures of task embeddings provide interpretable insight into the training of these

embeddings, with higher weights being put on agents which currently observe more information about the task.

Future work could aim to relax the assumption that training and testing tasks have identical observation dimensionality and number of agents. This assumption limits the settings in which MATE can be applied, and could be addressed using network architectures leveraging attention [Vaswani et al., 2017]. Lastly, future work should evaluate the feasibility of learning a mixture of task embeddings and centrally encoding joint information for centralised embeddings in tasks with many agents to investigate the scalability of our proposed approach.

References

Rishabh Agarwal, Max Schwarzer, Pablo Samuel Castro, Aaron Courville, and Marc G Bellemare. Deep reinforcement learning at the edge of the statistical precipice. In *Advances in Neural Information Processing Systems*, 2021.

Ilge Akkaya, Marcin Andrychowicz, Maciek Chociej, Mateusz Litwin, Bob McGrew, Arthur Petron, Alex Paino, Matthias Plappert, Glenn Powell, Raphael Ribas, Jonas Schneider, Nikolas A. Tezak, Jerry Tworek, Peter Welinder, Lilian Weng, Qiming Yuan, Wojciech Zaremba, and Lei M. Zhang. Solving rubik's cube with a robot hand. *ArXiv*, abs/1910.07113, 2019.

Stefano V. Albrecht and Subramanian Ramamoorthy. A Game-Theoretic Model and Best-Response Learning Method for Ad Hoc Coordination in Multiagent Systems. In *International Conference on Autonomous Agents and Multi-Agent Systems*, volume 2, pages 1155–1156, 2013. URL `http://dl.acm.org/citation.cfm?id=2485118`.

Stefano V Albrecht and Peter Stone. Autonomous agents modelling other agents: A comprehensive survey and open problems. *Artificial Intelligence*, 258:66–95, 2018.

Samuel Barrett and Peter Stone. Cooperating with unknown teammates in complex domains: A robot soccer case study of ad hoc teamwork. In *AAAI Conference on Artificial Intelligence*, 2015.

Christopher Berner, Greg Brockman, Brooke Chan, Vicki Cheung, Przemyslaw Debiak, Christy Dennison, David Farhi, Quirin Fischer, Shariq Hashme, Christopher Hesse, Rafal Józefowicz, Scott Gray, Catherine Olsson, Jakub W. Pachocki, Michael Petrov, Henrique Pondé de Oliveira Pinto, Jonathan Raiman, Tim Salimans, Jeremy Schlatter, Jonas Schneider, Szymon Sidor, Ilya Sutskever, Jie Tang, Filip Wolski, and Susan Zhang. Dota 2 with large scale deep reinforcement learning. *ArXiv*, abs/1912.06680, 2019.

Rich Caruana. Multitask learning. *Machine Learning*, 28(1):41–75, 1997.

Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using RNN encoder-decoder for statistical machine translation. *ArXiv*, abs/1406.1078, 2014.

Filippos Christianos, Lukas Schäfer, and Stefano V. Albrecht. Shared experience actor-critic for multi-agent reinforcement learning. In *Advances in Neural Information Processing Systems*, 2020.

Felipe Leno Da Silva and Anna Helena Reali Costa. A survey on transfer learning for multiagent reinforcement learning systems. *Journal of Artificial Intelligence Research*, 64:645–703, 2019.

Felipe Leno Da Silva, Ruben Glatt, and Anna Helena Reali Costa. Simultaneously learning and advising in multiagent reinforcement learning. In *International Conference on Autonomous Agents and Multi-Agent Systems*, 2017.

Felipe Leno Da Silva, Matthew E. Taylor, and Anna Helena Reali Costa. Autonomously reusing knowledge in multiagent reinforcement learning. In *International Joint Conference on Artificial Intelligence*, 2018.

Yan Duan, John Schulman, Xi Chen, Peter L. Bartlett, Ilya Sutskever, and Pieter Abbeel. Rl^2: Fast reinforcement learning via slow reinforcement learning. *ArXiv*, abs/1611.02779, 2016.

Gabriel Dulac-Arnold, Nir Levine, Daniel J Mankowitz, Jerry Li, Cosmin Paduraru, Sven Gowal, and Todd Hester. Challenges of real-world reinforcement learning: definitions, benchmarks and analysis. *Machine Learning*, 110(9):2419–2468, 2021.

Benjamin Eysenbach, Swapnil Asawa, Shreyas Chaudhari, Sergey Levine, and Ruslan Salakhutdinov. Off-dynamics reinforcement learning: Training for transfer with domain classifiers. In *International Conference on Learning Representations*, 2021.

Rasool Fakoor, Pratik Chaudhari, Stefano Soatto, and Alexander J Smola. Meta-Q-Learning. In *International Conference on Learning Representations*, 2020.

Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *International Conference on Machine Learning*. PMLR, 2017.

Jakob Foerster, Gregory Farquhar, Triantafyllos Afouras, Nantas Nardelli, and Shimon Whiteson. Counterfactual multi-agent policy gradients. In *AAAI Conference on Artificial Intelligence*, 2018.

Eric A Hansen, Daniel S Bernstein, and Shlomo Zilberstein. Dynamic programming for partially observable stochastic games. In *AAAI*, volume 4, pages 709–715, 2004.

Irina Higgins, Loic Matthey, Arka Pal, Christopher Burgess, Xavier Glorot, Matthew Botvinick, Shakir Mohamed, and Alexander Lerchner. β-vae: Learning basic visual concepts with a constrained variational framework. In *International Conference on Learning Representations*, 2017.

Timothy Hospedales, Antreas Antoniou, Paul Micaelli, and Amos Storkey. Meta-learning in neural networks: A survey. *ArXiv*, abs/2004.05439, 2020.

Hengyuan Hu, Adam Lerer, Alex Peysakhovich, and Jakob Foerster. "Other-Play" for zero-shot coordination. In *International Conference on Machine Learning*. PMLR, 2020.

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

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *ArXiv*, abs/1412.6980, 2014.

Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *ArXiv*, abs/1312.6114, 2013.

Wenqian Liang, Ji Wang, Weidong Bao, Xiaomin Zhu, Qingyong Wang, and Beibei Han. Continuous self-adaptive optimization to learn multi-task multi-agent. *Complex & Intelligent Systems*, 8(2): 1355–1367, 2022.

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

Andrei Lupu, Brandon Cui, Hengyuan Hu, and Jakob Foerster. Trajectory diversity for zero-shot coordination. In *International Conference on Machine Learning*. PMLR, 2021.

Anuj Mahajan, Mikayel Samvelyan, Tarun Gupta, Benjamin Ellis, Mingfei Sun, Tim Rocktäschel, and Shimon Whiteson. Generalization in cooperative multi-agent systems. *ArXiv*, abs/2202.00104, 2022.

Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, 2016.

Igor Mordatch and Pieter Abbeel. Emergence of grounded compositional language in multi-agent populations. In *AAAI Conference on Artificial Intelligence*, 2018.

Alex Nichol, Joshua Achiam, and John Schulman. On first-order meta-learning algorithms. *arXiv preprint arXiv:1803.02999*, 2018.

Shayegan Omidshafiei, Jason Pazis, Christopher Amato, Jonathan P How, and John Vian. Deep decentralized multi-task multi-agent reinforcement learning under partial observability. In *International Conference on Machine Learning*. PMLR, 2017.

Georgios Papoudakis, Filippos Christianos, and Stefano Albrecht. Agent modelling under partial observability for deep reinforcement learning. *Advances in Neural Information Processing Systems*, 2021a.

Georgios Papoudakis, Filippos Christianos, Lukas Schäfer, and Stefano V. Albrecht. Benchmarking multi-agent deep reinforcement learning algorithms in cooperative tasks. In *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks*, 2021b.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019. URL `http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf`.

Wei Qiu, Xiao Ma, Bo An, Svetlana Obraztsova, YAN Shuicheng, and Zhongwen Xu. Rpm: Generalizable multi-agent policies for multi-agent reinforcement learning. In *International Conference on Learning Representations*, 2023.

Muhammad A. Rahman, Niklas Hopner, Filippos Christianos, and Stefano V. Albrecht. Towards open ad hoc teamwork using graph-based policy learning. In *International Conference on Machine Learning*. PMLR, 2021.

Kate Rakelly, Aurick Zhou, Chelsea Finn, Sergey Levine, and Deirdre Quillen. Efficient off-policy meta-reinforcement learning via probabilistic context variables. In *International conference on machine learning*. PMLR, 2019.

Tabish Rashid, Mikayel Samvelyan, Christian Schroeder de Witt, Gregory Farquhar, Jakob Foerster, and Shimon Whiteson. QMIX: Monotonic value function factorisation for deep multi-agent reinforcement learning. In *International Conference on Machine Learning*. PMLR, 2018.

Lukas Schäfer. Task generalisation in multi-agent reinforcement learning. In *Doctoral Consortium at the International Conference on Autonomous Agents and Multiagent Systems*, 2022.

Peter Stone, Gal Kaminka, Sarit Kraus, and Jeffrey Rosenschein. Ad hoc autonomous agent teams: Collaboration without pre-coordination. In *AAAI Conference on Artificial Intelligence*, 2010.

Peter Sunehag, Guy Lever, Audrunas Gruslys, Wojciech M. Czarnecki, Vinícius Flores Zambaldi, Max Jaderberg, Marc Lanctot, Nicolas Sonnerat, Joel Z. Leibo, Karl Tuyls, and Thore Graepel. Value-decomposition networks for cooperative multi-agent learning. *International Conference on Autonomous Agents and Multi-Agent Systems*, 2018.

Matthew E. Taylor and Peter Stone. Transfer learning for reinforcement learning domains: A survey. *Journal of Machine Learning Research*, 10(7), 2009.

Laurens Van der Maaten and Geoffrey Hinton. Visualizing data using t-SNE. *Journal of machine learning research*, 9(11), 2008.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *International Conference on Neural Information Processing Systems*, 2017.

Alexander Sasha Vezhnevets, Yuhuai Wu, Remi Leblond, and Joel Z. Leibo. Options as responses: Grounding behavioural hierarchies in multi-agent RL. In *International Conference on Machine Learning*. PMLR, 2020.

Oriol Vinyals, Igor Babuschkin, Wojciech M Czarnecki, Michaël Mathieu, Andrew Dudzik, Junyoung Chung, David H Choi, Richard Powell, Timo Ewalds, Petko Georgiev, et al. Grandmaster level in starcraft ii using multi-agent reinforcement learning. *Nature*, 575(7782):350–354, 2019.

Nelson Vithayathil Varghese and Qusay H. Mahmoud. A survey of multi-task deep reinforcement learning. *Electronics*, 9(9):1363, 2020.

Jane X Wang, Zeb Kurth-Nelson, Hubert Soyer, Joel Z. Leibo, Dhruva Tirumala, Rémi Munos, Charles Blundell, Dharshan Kumaran, and Matt M Botvinick. Learning to reinforcement learn. In *Annual Meeting of the Cognitive Science Community (CogSci)*, 2016.

Shenao Zhang, Li Shen, and Lei Han. Learning meta representations for agents in multi-agent reinforcement learning. *ArXiv*, abs/2108.12988, 2021.

Brian D. Ziebart. Modeling purposeful adaptive behavior with the principle of maximum causal entropy. 2010.

Luisa Zintgraf, Kyriacos Shiarlis, Maximilian Igl, Sebastian Schulze, Yarin Gal, Katja Hofmann, and Shimon Whiteson. Varibad: A very good method for bayes-adaptive deep rl via meta-learning. In *International Conference on Learning Representations*, 2020.

Luisa Zintgraf, Sam Devlin, Kamil Ciosek, Shimon Whiteson, and Katja Hofmann. Deep interactive bayesian reinforcement learning via meta-learning. *ArXiv*, abs/2101.03864, 2021.

A MATE Loss Derivation

The standard evidence lower bound (ELBO) to maximise for variational encoder-decoder networks [Kingma and Welling, 2013] with additional hyperparameter β proposed to control the KL prior regularisation [Higgins et al., 2017] is given by

$$\text{ELBO}(\boldsymbol{\phi}, \boldsymbol{\psi}|\boldsymbol{\tau}_{1:t}) = \mathbb{E}_{q(\mathbf{z}_t|\boldsymbol{\tau}_{1:t};\boldsymbol{\phi})}\left[\log p(\mathbf{o}_{t+1}, \mathbf{r}_t|\mathbf{o}_t, \mathbf{a}_t, \mathbf{z}; \boldsymbol{\psi})\right] - \beta \text{KL}\left(q\left(\mathbf{z}|\boldsymbol{\tau}_{1:t}; \boldsymbol{\phi}\right)||p(\mathbf{z})\right) \quad (3)$$

For a standard Gaussian prior $p(\mathbf{z}) = \mathcal{N}(\mathbf{z}; \mathbf{0}, \text{diag}(\mathbf{1}))$ and Gaussian variational distribution $q(\mathbf{z}|\tau_{1:t}) = \mathcal{N}(\mathbf{z}; \boldsymbol{\mu}, \text{diag}(\boldsymbol{\sigma}))$ ($\boldsymbol{\mu}, \text{diag}(\boldsymbol{\sigma}) \in \mathbb{R}^d$), we can write a closed-form solution for the KL divergence term:

$$\text{KL}\left(q\left(\mathbf{z}|\tau_{1:t}; \phi\right)||p(\mathbf{z})\right) = -\frac{1}{2}\sum_{j=1}^{d}\left(1 + \log\left(\sigma_j^2\right) - \mu_j^2 - \sigma_j^2\right) \quad (4)$$

Next, we will derive that maximising the log-likelihood of a random variable, such as the decoded observations and rewards, of dimensionality d_p under a generative multivariate Gaussian distribution $p(\mathbf{x}|\mathbf{z}) = \mathcal{N}(\boldsymbol{\mu}_p, \text{diag}(\boldsymbol{\sigma}_p))$ with constant diagonal covariance matrix $\text{diag}(\boldsymbol{\sigma}_p)$ (1) is equivalent to minimising the Euclidean distance between the random variable and the mean of the multivariate generative distribution. Therefore, minimising the Euclidean distance between samples of the observations and rewards and a deterministic output of the decoder is equivalent to maximising the log-likelihood with the generator learning to predict the mean of the modelled variables. We make use of the fact that we can ignore any constants for the maximisation objective (2):

$$\max \log p(\mathbf{x}|\mathbf{z}) = \max \log \left[\frac{1}{\sqrt{(2\pi)^{d_p}|\text{diag}(\boldsymbol{\sigma}_p)|}} \exp\left(-\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu}_p)^T \text{diag}(\boldsymbol{\sigma}_p)^{-1}(\mathbf{x} - \boldsymbol{\mu}_p)\right)\right] \quad (5)$$

$$= \max \log \left[\frac{1}{\sqrt{(2\pi)^{d_p}|\text{diag}(\boldsymbol{\sigma}_p)|}}\right] + \log \left[\exp\left(-\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu}_p)^T \text{diag}(\boldsymbol{\sigma}_p)^{-1}(\mathbf{x} - \boldsymbol{\mu}_p)\right)\right] \quad (6)$$

$$\overset{(1,2)}{=} \max \log \left[\exp\left(-\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu}_p)^T \text{diag}(\boldsymbol{\sigma}_p)^{-1}(\mathbf{x} - \boldsymbol{\mu}_p)\right)\right] \quad (7)$$

$$= \max -\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu}_p)^T \text{diag}(\boldsymbol{\sigma}_p)^{-1}(\mathbf{x} - \boldsymbol{\mu}_p) \quad (8)$$

$$\overset{(1,2)}{=} \max -(\mathbf{x} - \boldsymbol{\mu}_p)^T(\mathbf{x} - \boldsymbol{\mu}_p) \quad (9)$$

$$= \min \left(\mathbf{x} - \boldsymbol{\mu}_p\right)^2 \quad (10)$$

Given this derivation (3), we will rewrite the log-likelihood term in the ELBO assuming that the generator, or decoder, is parameterised as a multivariate Gaussian model over observations $p(\mathbf{o}_{t+1}|\mathbf{o}_t, \mathbf{a}_t, \mathbf{z}) = \mathcal{N}(\boldsymbol{\mu}_o, \text{diag}(\boldsymbol{\sigma}_o))$ and rewards $p(\mathbf{r}_t|\mathbf{o}_t, \mathbf{a}_t, \mathbf{z}) = \mathcal{N}(\boldsymbol{\mu}_r, \text{diag}(\boldsymbol{\sigma}_r))$ with constant diagonal covariance matrices $\text{diag}(\boldsymbol{\sigma}_o)$ and $\text{diag}(\boldsymbol{\sigma}_r)$, respectively. Following the assumption of a constant diagonal covariance matrix, we can assume that the multivariate Gaussian models for observations and rewards are independent of each other (4).

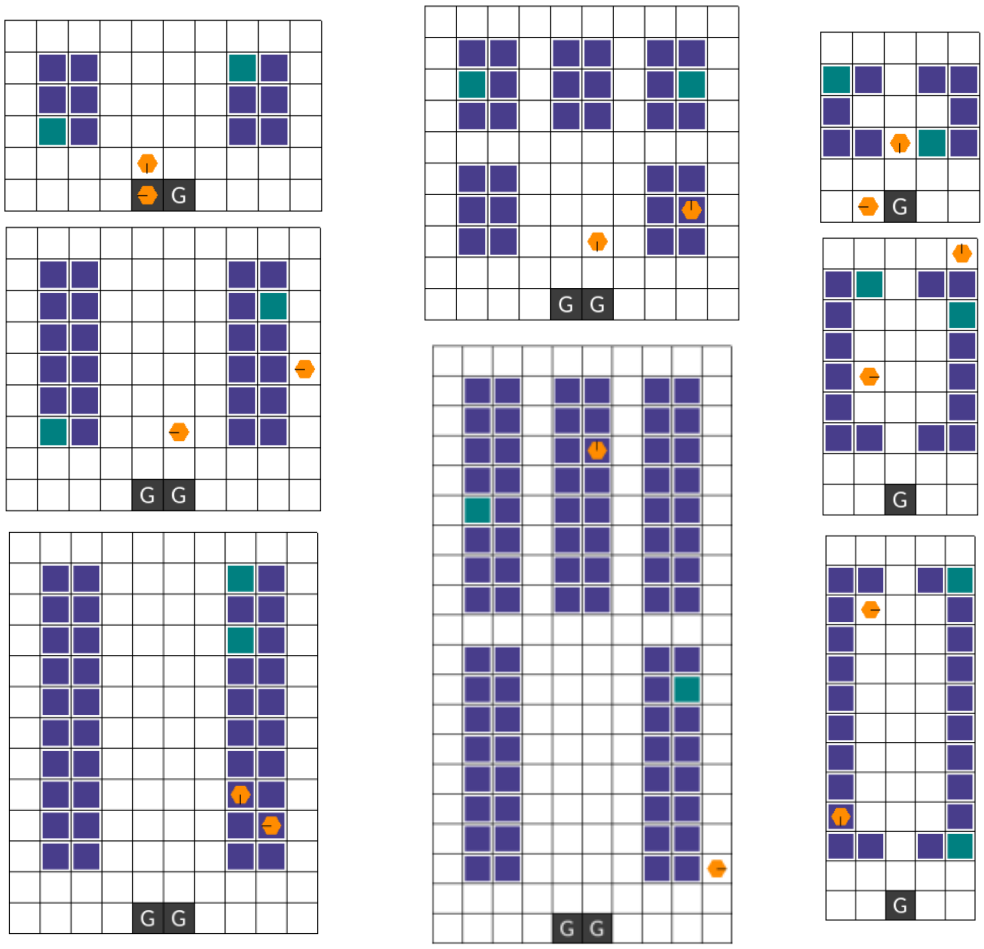
$$\max \log p(\mathbf{o}_{t+1}, \mathbf{r}_t|\mathbf{o}_t, \mathbf{a}_t, \mathbf{z}; \psi) = \max \log \left[p(\mathbf{o}_{t+1}|\mathbf{o}_t, \mathbf{a}_t, \mathbf{z}; \psi) \cdot p(\mathbf{o}_{t+1}|\mathbf{o}_t, \mathbf{a}_t, \mathbf{z}; \psi)\right] \quad (11)$$

$$\overset{(4)}{=} \max \log p(\mathbf{o}_{t+1}|\mathbf{o}_t, \mathbf{a}_t, \mathbf{z}; \psi) + \log p(\mathbf{o}_{t+1}|\mathbf{o}_t, \mathbf{a}_t, \mathbf{z}; \psi) \quad (12)$$

$$\overset{(3)}{=} \min \left(\mathbf{o}_{t+1} - \boldsymbol{\mu}_o\right)^2 + \left(\mathbf{r}_t - \boldsymbol{\mu}_r\right)^2 \quad (13)$$

Following the derivation of the KL regularisation term as well as showing the equivalence of maximising the log-likelihood of observations and rewards to minimising the Euclidean distance between generative Gaussian distribution mean and samples of observations and rewards, we obtain the MATE loss presented in Equation (2):

$$\mathbb{L}(\boldsymbol{\phi}, \boldsymbol{\psi}|\boldsymbol{\tau}_{1:t}) = \mathbb{E}_{q(\mathbf{z}_t|\boldsymbol{\tau}_{1:t};\boldsymbol{\phi})}\Big[\left(p(\mathbf{o}_{t+1}|\mathbf{o}_t, \mathbf{a}_t, \mathbf{z}; \boldsymbol{\psi}) - \mathbf{o}_{t+1}\right)^2 + \left(p(\mathbf{r}_t|\mathbf{o}_t, \mathbf{a}_t, \mathbf{z}; \boldsymbol{\psi}) - \mathbf{r}_t\right)^2\Big] - \beta\frac{1}{2}\sum_{j=1}^{d}\left(1 + \log\left(\sigma_j^2\right) - \mu_j^2 - \sigma_j^2\right) \quad (14)$$



(a) Tiny (b) Small (c) Corridor

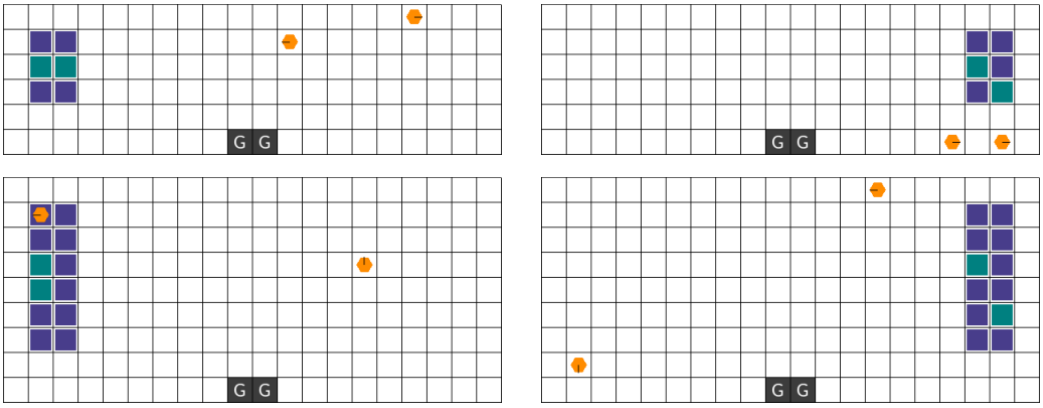
Figure 7: Tiny, small and corridor warehouses with 2 agents and column heights of 3, 6, 10 for tiny and corridor and colum heights 3 and 8 for small.

B Multi-Agent Environments

B.1 Multi-Robot Warehouse

The multi-robot warehouse (RWARE) [Papoudakis et al., 2021b, Christianos et al., 2020] environment[1] simulates a logistics task in which agents control robots needing to collect and deliver requested items from shelves within warehouses. The number of requested items at each time step is identical to the number of agents N. For this work, we use a modified observation space with a 5×5 grid centred around the agent containing information about shelves, requested items, agents, delivery zones and boundaries of the warehouse. All information is encoded as a binary features for $5 \times 5 \times 5$ values. Additionally, each agent observes its own rotation within the warehouse as a onehot vector and two binary features indicating if the agent currently carries a shelf and is located on shelves, respectively. All this information is flattened to observations with 131 values. This observation space was chosen to simplify generalisation by encoding all information relative to the agent [Schäfer, 2022] which is further ensured by rotating observations whenever the agent rotates within the warehouse. Agents can choose between five actions: stay, move forward, rotate left, rotate right and toggle load. The last action corresponds to picking-up as well as putting-down shelves depending on the current load of the agent. Agents receive individual rewards of $+1$ for successfully delivering a shelf with a requested item to the delivery zone and 0 otherwise. Upon successful delivery of a requested item, a

[1] RWARE is licensed under the MIT license.



(a) Wide-one-sided left

(b) Wide-one-sided right

(c) Wide-both

(d) Wide-both

Figure 8: Wide-one-sided and wide-both warehouses with 2 agents and column heights of 3 and 6.

new, currently unrequested item is randomly sampled and added to the list of requests to ensure that the number of total requests at each time step is constant and equal to the number of agents N.

In this work, we consider tasks with varying warehouses with two or four agents. Each set of training and testing tasks contains multiple warehouse tasks with varying heights of blocks of shelves between 3 and 10. For visualisations of all warehouse layouts, see Figures 7 and 8.

B.2 Multi-Agent Particle Environment

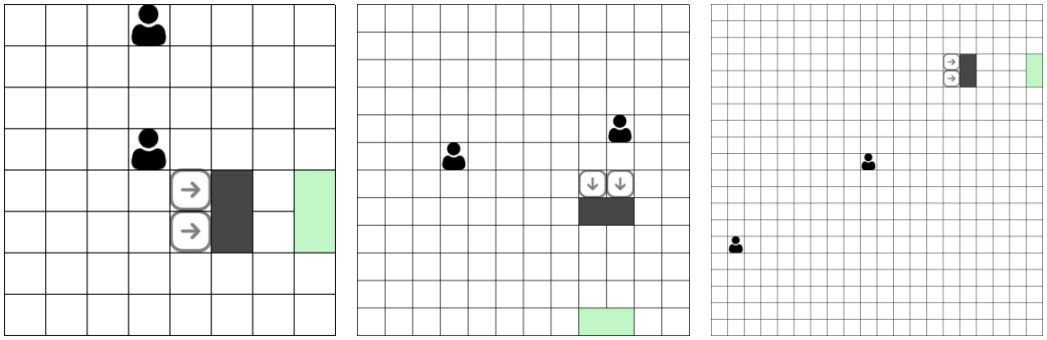
The multi-agent particle environment[2] (MPE) [Mordatch and Abbeel, 2018, Lowe et al., 2017] contains several cooperative and competitive multi-agent tasks. Agents control particles in a two-dimensional continuous environment to move with actions being: stay, move right, move left, move up, and move down. In this work, we consider the cooperative navigation task in which three agents need to move to three landmarks within the environment and avoid collisions with other agents. For each episode, landmarks are randomly placed in the environment but landmarks remain fixed throughout the episode. Agents observe their velocity, position as well as their relative positions to other agents and all landmarks. All agents receive the same rewards defined as the sum of negative minimal distances from each landmark to any agent and an additional penalty reward of -1 for collisions of agents. Episodes end after 25 time steps.

Based on this environment, we further define two new tasks as testing tasks in which agents receive larger negative rewards for collisions with -5 and -50 penalty rewards for each collision.

B.3 Boulder-Push

The Boulder-push environment (BPUSH) is a new environment proposed as part of this work. In this environment, agents navigate a gridworld and are tasked to push a box in a fixed direction towards a given goal location. At the beginning of each episode, agents and the box are randomly placed in the environment and the direction and distance in which the box needs to be pushed are randomly

[2]MPE environment is licensed under the MIT license.



(a) Small (b) Medium (c) Large

Figure 9: BPUSH environments with varying sizes and two agents.

determined. Agents observe a flattened 9×9 grid centred on themselves containing the locations of agents and the box as binary features. Additionally, agents globally observe the direction in which the box has to be pushed. Agent actions include movement in each cardinal direction and agents receive a large positive reward of $+1$ for pushing the box to its goal location. Agents also receive a smaller reward of $+0.1$ for pushing the box forward at all. This additional reward is intended to simplify the exploration of agents to discover the optimal policy. The box is only successfully pushed if all agents walk against the box from the opposite side and the box can only be moved in its pre-determined direction. Episodes terminate after the box has reached its goal location or after at most 50 time steps.

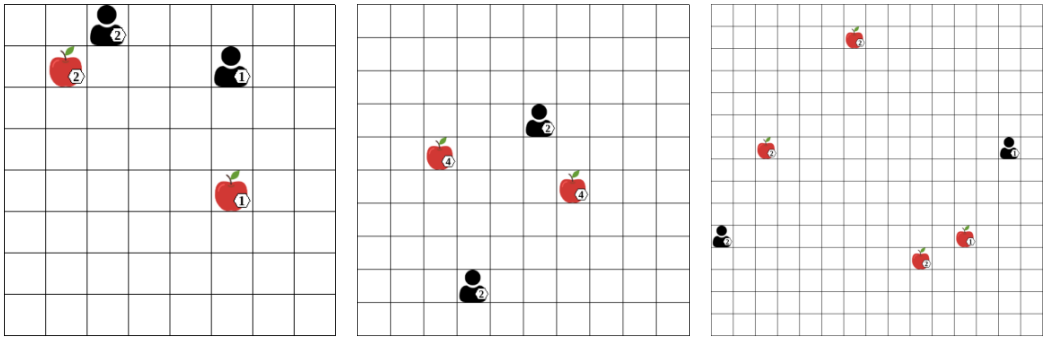
We initially train two agents in a task with a small 8×8 gridworld. Testing tasks include tasks with larger gridworld sizes medium (12×12) and large (20×20) and tasks in which agents receive a small negative reward of -0.01 for unsuccessful pushing attempts. Latter occurs whenever a single agent attempts to push the box but the other agent is not cooperating such that the box is not moved. For visualisations of all considered gridworld sizes with two agents, see Figure 9.

B.4 Level-Based Foraging

The level-based foraging (LBF) [Albrecht and Ramamoorthy, 2013, Albrecht and Stone, 2018] environment[3] contains tasks in which agents need to navigate a gridworld to collect food. For each episode, food and agents are randomly placed in the gridworld and assigned levels. Agents are only able to pick-up adjacent food items if the sum of the levels of all agents currently picking-up the food is greater or equal to the level of the food. Each agent observes a 5×5 grid centred on itself containing information about nearby agents and food given by their level as well as boundaries of the gridworld. Agents choose one of six discrete actions: stay, move north, move south, move west, move east and pick-up. Agents receive normalised rewards for successful picking-up of food equal to the level of the collected food. It should be noted that agents receive individual rewards rather than sharing a single reward across all agents.

We initially train two or four agents in tasks with comparably small gridworlds with sizes 8×8 containing as many food items as agents. Agent and food levels are randomly assigned for each episode while ensuring that each food can be collected. Therefore, agents will need to cooperate to pick-up some food while others can be collected by individual agents. Testing tasks contain tasks with larger gridworld environments, more food as well as tasks with enforced cooperation or penalty. In tasks with enforced cooperation (denoted by "-coop"), each food has a high level such that it can only be collected if all agents in the environment cooperate. In testing tasks with penalty (denoted by "-pen"), agents receive a negative reward of -0.1 for unsuccessful picking attempts. A picking attempt is unsuccessful whenever an agent chose the pick-up action but did not succeed at collecting any food. For visualisations of all considered gridworld sizes with two agents as well as visualisation of a task with forced cooperation, see Figure 10.

[3]LBF is licensed under the MIT license.



(a) 8x8-2p-2f (b) 10x10-2p-2f-coop (c) 15x15-2p-4f

Figure 10: LBF tasks with varying sizes and two agents.

C Experimental Details

C.1 Hyperparameters

For all algorithms, we train agent policies using the MAA2C algorithm with each agent optimising its own policy and critic. The critic computes a state value function and is conditioned on the joint observations of all agents. Agents are trained in ten synchronous environments running in parallel [Mnih et al., 2016] with batches of experience from all environments across 5 time steps used to compute policy, value and MATE loss and optimise the networks. We compute 5-step return estimates for each of the environments and a entropy regularisation term was added to the loss to incentivise exploration [Mnih et al., 2016, Ziebart, 2010]. Policy networks consist of a single fully-connected layer followed by a gated recurrent unit (GRU) [Cho et al., 2014] and a last fully-connected layer. Critic networks consist of two fully-connected layers and the ReLU activation function is applied after each layer. Networks are optimised using the Adam [Kingma and Ba, 2014] optimiser and target critic networks were used to compute target return estimates and updated using soft updates with $\tau = 0.01$.

MATE encoders consist of a fully-connected layer, followed by a GRU and a final fully-connected layer and decoders consist of two fully-connected layers. The encoder output $\mathcal{N}(\mathbf{z})$ has dimensionality of 6 with $|\mu| = |\sigma| = 3$. For MATE hyperparameters, a small gridsearch over β, task embedding size and learning rate was conducted in the RWARE wide fine-tuning scenarios. We also find that backpropagating gradients from the MARL training to the MATE encoders is possible but does not lead to improved adaptation performance so we stop gradients to flow into the MATE encoders from the MARL training.

For all hyperparameters for MAA2C and MATE, see Table 2.

C.2 Computational Resources

Experiments were run on CPUs only across three research clusters being equipped with (1) a AMD EPYC 7502 CPU with 32 cores @2.5Ghz running Ubuntu 20.04.3 LTS, (2) a AMD EPYC 7502 CPU running Scientific Linux 7.9, and (3) a AMD EPYC 7H12 CPU with 64 cores @2.6Ghz running Ubuntu 20.04.3 LTS. We used Python 3.9.7 with PyTorch 1.10 [Paszke et al., 2019].

We conduct a runtime comparison of training agents with and without MATE and present the runtime together with percentage increase of using MATE in comparison to using no MATE in two tasks in Table 3. For this comparison, we train each of the algorithms for 100,000 time steps in the RWARE tiny-2ag and RWARE tiny-4ag tasks to also indicate the scalability with varying number of agents. Timings were measured on a personal computer with a Intel i7-10750H CPU with 12 cores @2.60GHz running Ubuntu 20.04.4 LTS, Python 3.9.7 and PyTorch 1.10. For all algorithms, we used the hyperparameters specified in Table 2.

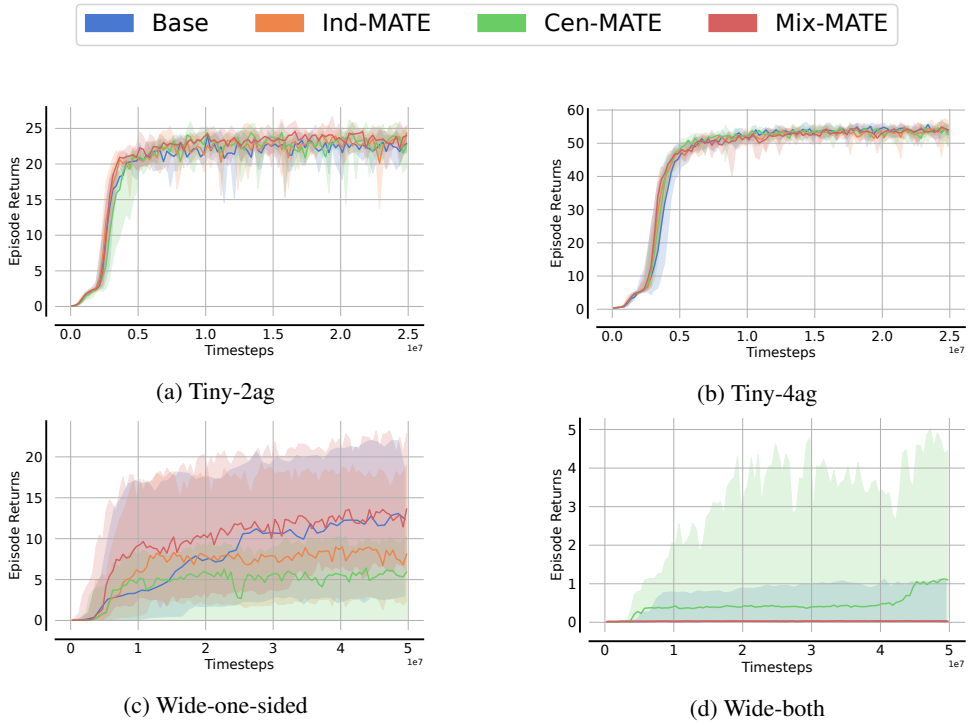
Table 2: Hyperparameters for MAA2C and MATE

Algorithm	Hyperparameter	Value
MAA2C	Policy network	FC(128) $\times$ GRU(128) $\times$ FC(128)
	Critic network	FC(128) $\times$ FC(128)
	Activation function	ReLU
	Optimiser	Adam
	Learning rate	$5e^{-4}$
	Adam epsilon	$1e^{-3}$
	Entropy loss coefficient	0.01
	Value loss coefficient	0.5
	Target critic τ	0.01
	γ	0.99
	N-step returns	5
	Parallel environments	10
	Max grad norm	/ (no gradient normalisation)
MATE	Encoder q network	FC(64) $\times$ GRU(64) $\times$ FC(64)
	Decoder p network	FC(64) $\times$ FC(64)
	Mixing f_m network	FC(64)
	Activation function	ReLU
	Optimiser	Adam
	Learning rate	$1e^{-4}$
	Adam epsilon	$1e^{-3}$
	β	0.1
	Task embedding size	3
	Max grad norm	0.5

Table 3: Runtime (in seconds) for agents trained using MAA2C with and without MATE for 100,000 time steps.

Task	Algorithm	Runtime	% Increase
RWARE tiny-2ag	MAA2C No MATE	144.32s	/
	MAA2C Ind-MATE	170.02s	17.81%
	MAA2C Cen-MATE	161.06s	11.60%
	MAA2C Mix-MATE	168.38s	16.67%
RWARE tiny-4ag	MAA2C No MATE	204.00s	/
	MAA2C Ind-MATE	243.69s	19.46%
	MAA2C Cen-MATE	223.80s	9.71%
	MAA2C Mix-MATE	250.06s	22.58%

D Training Returns



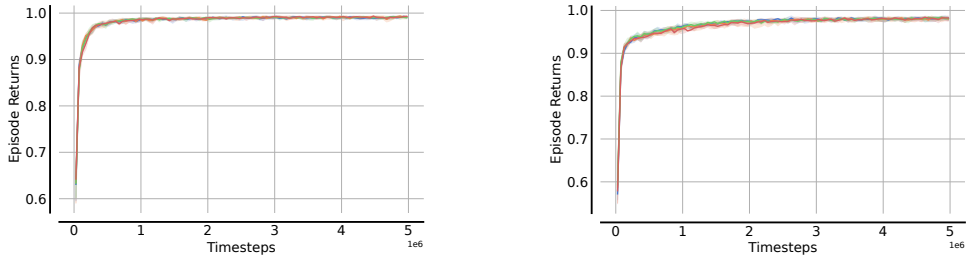
(a) Tiny-2ag

(b) Tiny-4ag

(c) Wide-one-sided

(d) Wide-both

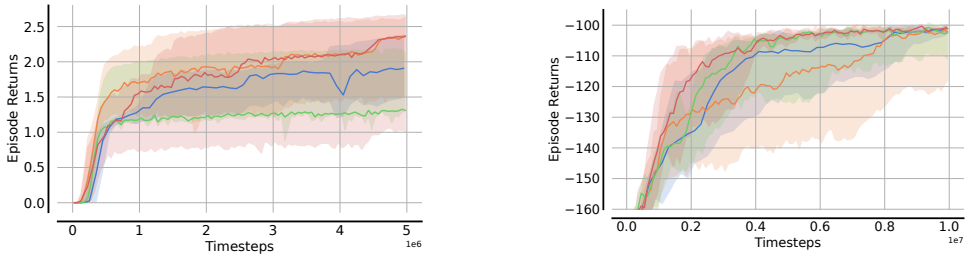
Figure 11: IQM and 95% confidence intervals for RWARE training



(a) 8x8-2p-2f

(b) 8x8-4p-4f

Figure 12: IQM and 95% confidence intervals for LBF training

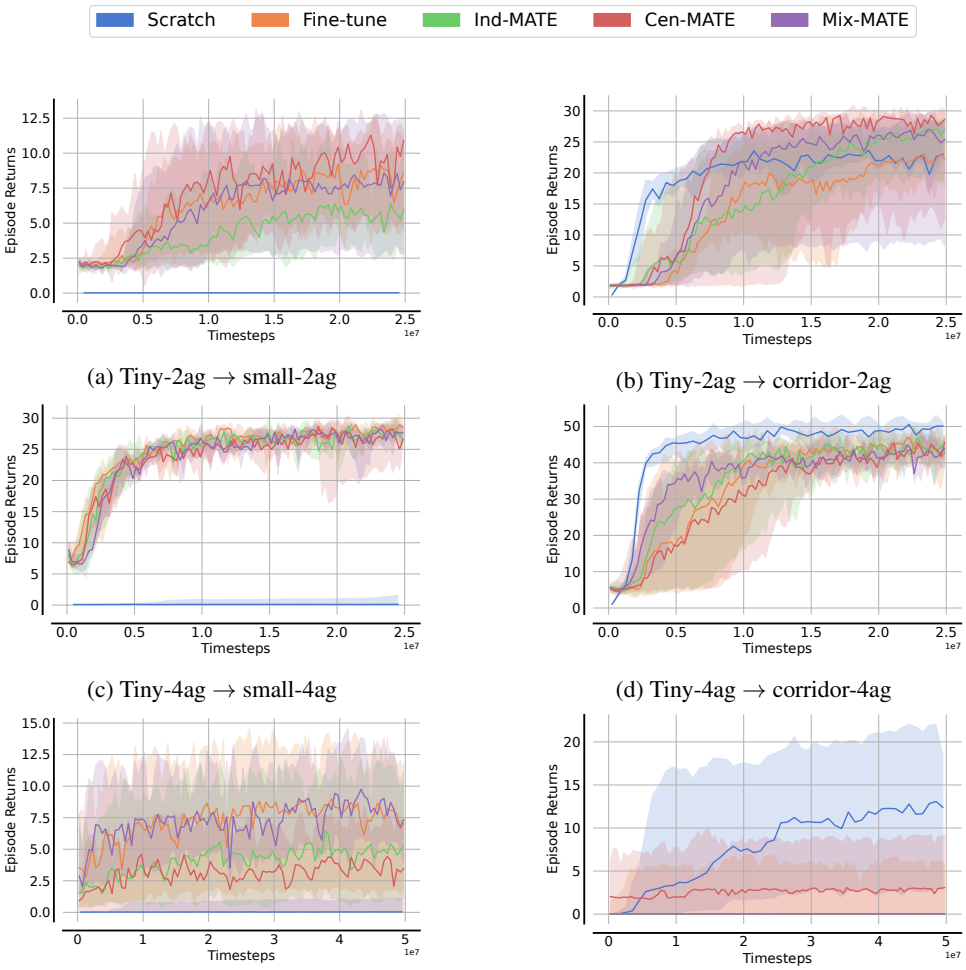


(a) BPUSH small-2ag

(b) MPE cooperative navigation

Figure 13: IQM and 95% confidence intervals for BPUSH and MPE training

E Fine-Tuning Returns



(a) Tiny-2ag → small-2ag

(b) Tiny-2ag → corridor-2ag

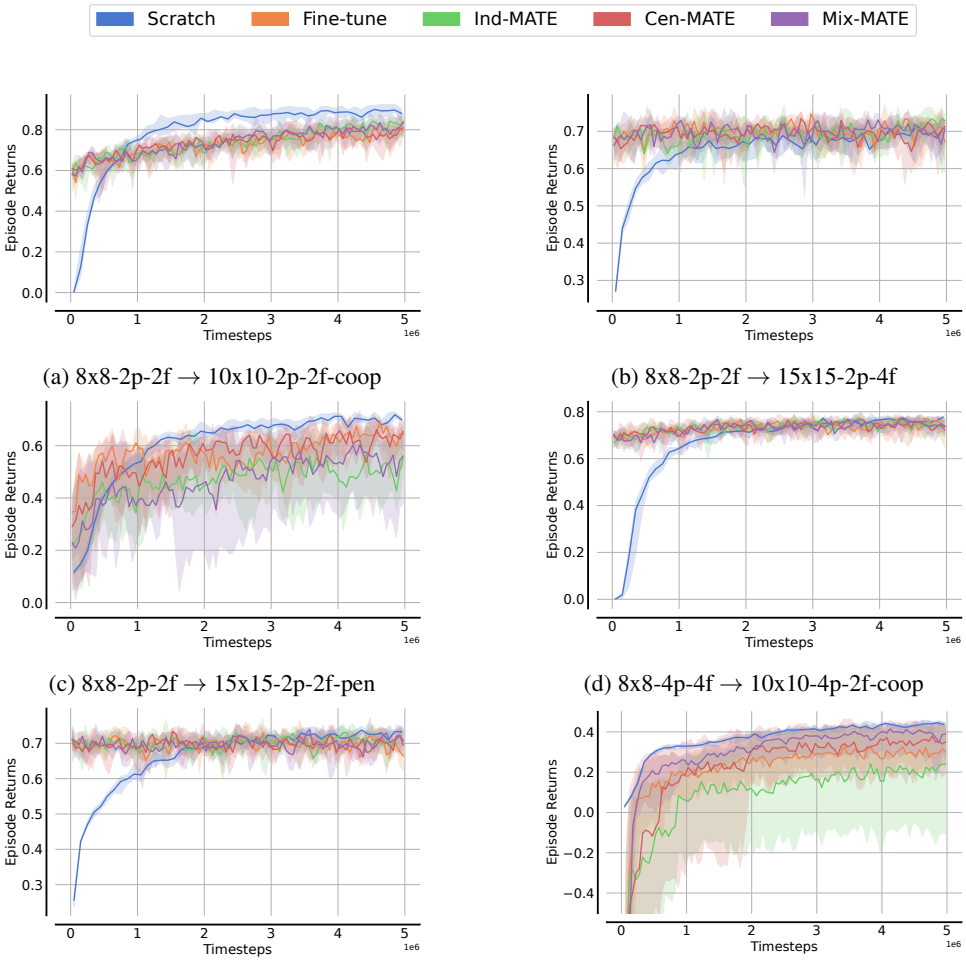
(c) Tiny-4ag → small-4ag

(d) Tiny-4ag → corridor-4ag

(e) Wide-one-sided → wide-both

(f) Wide-both → wide-one-sided

Figure 14: IQM and 95% confidence intervals for RWARE fine-tuning



(a) 8x8-2p-2f → 10x10-2p-2f-coop

(b) 8x8-2p-2f → 15x15-2p-4f

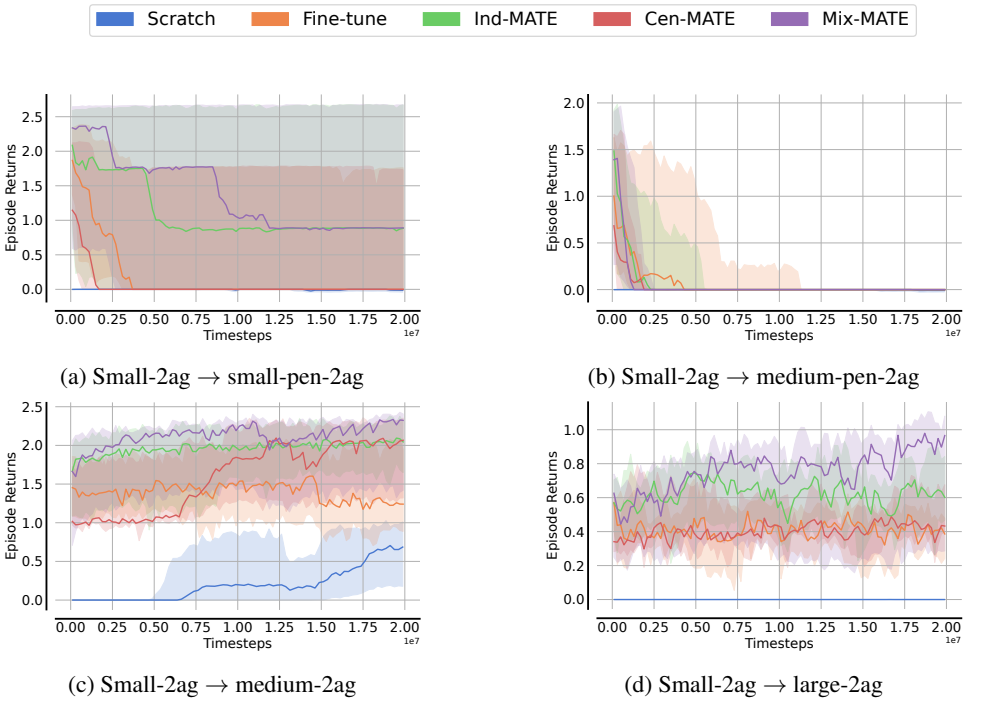
(c) 8x8-2p-2f → 15x15-2p-2f-pen

(d) 8x8-4p-4f → 10x10-4p-2f-coop

(e) 8x8-4p-4f → 15x15-4p-6f

(f) 8x8-4p-4f → 15x15-4p-4f-pen

Figure 15: IQM and 95% confidence intervals for LBF fine-tuning



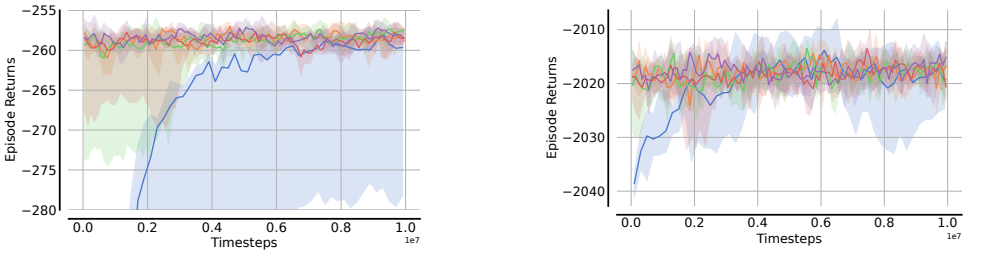
(a) Small-2ag → small-pen-2ag

(b) Small-2ag → medium-pen-2ag

(c) Small-2ag → medium-2ag

(d) Small-2ag → large-2ag

Figure 16: IQM and 95% confidence intervals for BPUSH fine-tuning



(a) Cooperative navigation (pen=1) → penalty navigation (pen=5)

(b) Cooperative navigation → penalty navigation (pen=50)

Figure 17: IQM and 95% confidence intervals for MPE fine-tuning