Dynamic Coordination Graph for Cooperative Multi-Agent Reinforcement Learning

Chapman Siu
Faculty of Engineering and Information Technology
University of Technology Sydney, Australia
chpmn.siu@gmail.com

Jason Traish
Faculty of Engineering and Information Technology
University of Technology Sydney, Australia
Jason.Traish@uts.edu.au

Richard Yi Da Xu
Faculty of Engineering and Information Technology
University of Technology Sydney, Australia
YiDa.Xu@uts.edu.au

Abstract

This paper introduces Dynamic Q-value Coordination Graph (QCGraph) for cooperative multi-agent reinforcement learning. QCGraph aims to dynamically represent and generalize through factorizing the joint value function of all agents according to dynamically created coordination graph based on subsets of agents. The value can be maximized by message passing at both a local and global level along the graph which allows training the value function end-to-end. The coordination graph is dynamically generated and used to generate the payoff functions which are approximated using graph neural networks and parameter sharing to improve generalization over the state-action space. We show that QCGraph can solve a variety of challenging multi-agent tasks being superior to other value factorization approaches.

1 Introduction

One of the central challenges in cooperative *multi-agent reinforcement learning* (MARL) is coping with the size of the joint action space, which grows exponentially in the number of agents. For example, some of the environments used in this paper have over eight agents with nine actions to choose from, yielding a joint action space which exceeds over a million actions. Efficient MARL methods must then be able to generalize over a large joint action space, particularly when the application of traditional single-agent reinforcement learning methods are impractical.

A common approach used to cope with large action spaces is to decentralize the decision policy and/or value function, whereby an agent policy is conditioned on only their own local action-observation history. In order to facilitate training and communication amongst agents, learning may occur in a centralized fashion to provide additional state information and to remove inter-agent communication constraints. This paradigm is known as *centralized training with decentralized execution* [16, 10]. Recently, these approaches have generally focused on *factored Q-value functions*, which focus on the value function factorization with some success [17, 19, 18].

Preprint - to appear on the 13th Asian Conference on Machine Learning 2021 (ACML).

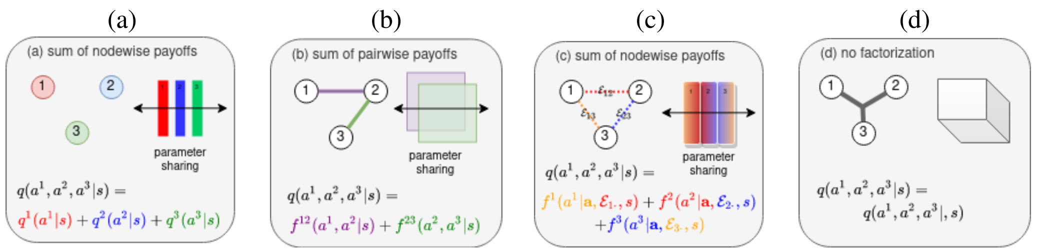
(a)
(a) sum of nodewise payoffs
1
2
3
parameter sharing
$q(a^1, a^2, a^3|s) =$
$q^1(a^1|s) + q^2(a^2|s) + q^3(a^3|s)$
(b)
(b) sum of pairwise payoffs
parameter sharing
$q(a^1, a^2, a^3|s) =$
$f^{12}(a^1, a^2|s) + f^{23}(a^2, a^3|s)$
(c)
(c) sum of nodewise payoffs
$\mathcal{E}_{12}$
$\mathcal{E}_{13}$
$\mathcal{E}_{23}$
parameter sharing
$q(a^1, a^2, a^3|s) =$
$f^1(a^1|\mathbf{a}, \mathcal{E}_{1\cdot}, s) + f^2(a^2|\mathbf{a}, \mathcal{E}_{2\cdot}, s)$
$+f^3(a^3|\mathbf{a}, \mathcal{E}_{3\cdot}, s)$
(d)
(d) no factorization
$q(a^1, a^2, a^3|s) =$
$q(a^1, a^2, a^3|, s)$

Figure 1: Examples of value factorization are shown in (a, b, c, d), examples of coordination graph factorization shown in items (b, c, d): (a) sum of independent utilities as used in VDN, QMIX; (b) sum of pairwise payoffs as used in DCG; (c) sum of node-wise payoffs based on dynamically generated coordination graph (our approach); (d) single hyper-edge with no factorization as used in QTRAN.

There has also been a focus on representational ability when these factorization techniques suffer from structural constraints. This has been partially addressed through considering the joint action-value space within the centralized training step [18, 1, 2]. One approach is to construct either new networks or models for every pair of agents[2]. While this approach appears intractable for large number of agents, the complexity can be resolved through directly factorizing the full joint action-value function [18] or using a static predetermined coordination graph with parameter sharing [1]. However, both approaches presuppose a fixed, immutable coordination graph which may not necessarily be appropriate depending on the MARL problem.

Contribution To address these issues, this paper proposes Dynamic Q-value Coordination Graph (QCGraph). QCGraph represents the value function as a coordination graph (CG) with payoffs determined by dynamically generating coordination structures among subsets of agents. To achieve scalability, QCGraph employs graph neural network techniques to enable parameter sharing among arbitrary dynamically generated graph structures.

We compare QCGraph's performance with that of other MARL Q-learning algorithms in a variety of tasks including predator-prey that requires coordinated actions. This nuance is important if we take the example of a pack of predators aiming at prey who become separated during the journey. The actions of predators who are close by ought to be more *relevant* than predators who are farther away, which may break down in the scenario where we have a fixed or predetermined coordination graph.

2 Background

Dec-POMDP. We assume a Dec-POMDP for n agents $\langle S, U, P, r, Z, O, n, \gamma \rangle$, where S is the state space of the environment [15]. At each timestep, $t = 1, 2, 3, \ldots$, every agent $i \in \mathcal{A} \equiv \{1, \ldots, n\}$, which chooses an action $u^i \in U$ which forms the joint action $\mathbf{u} \in \mathbf{U} \equiv U^n$. At each time step t, the next state $s \in S$ is drawn from state transition function $P(.|s, \mathbf{u})$. A transition yields a collaborative reward $r(s, \mathbf{u})$, and $\gamma \in [0, 1)$ denotes the discount factor. We consider *partially observable* settings, where each agent does not have full state, but samples observations $z \in Z$ according to observation function $O(s, i) : S \times \mathcal{A} \to Z$. We denote the action observation history for an agent i to be $\tau^i \in T \equiv (Z \times U)^*$, on which it conditions its stochastic policy $\pi^i(u^i|\tau^i) : T \times U \to [0, 1]$. The joint policy π is based on the *action-value function*: $Q^\pi(s_t, \mathbf{u}_t) = \mathbb{E}_{s_{t+1:\infty}, \mathbf{u}_{t+1:\infty}}[R_t|s_t, \mathbf{u}_t]$, where $R_t = \sum_{i=0}^{\infty} \gamma^i r_{t+i}$ which is the *discounted reward*. The goal of the problem is to find the optimal action value function Q^*

Centralized Training Decentralized Execution (CTDE). In this setting we further consider the scenario where training is centralised, and execution is decentralized, that is during training, the learning algorithm has access to the action-observation histories of all agents and the full state, however each agent can only condition on its own local action-observation history τ^i during decentralized execution.

Model-Free Q Learning. Prior work on the decentralized agent execution using Q-learning include Independent Q-learning (IQL) [20], where the algorithm models each agent as an independent Q-learner. This yields challenges as the perspective of a single agent becomes non-stationary as other agents change their policies. In order to resolve this some centralized entity is used [16, 10], which typically employ some form of parameter sharing between agents. Value Decomposition Networks (VDN) [19], perform central Q-learning with a value function that is the sum of independent utility functions for each agent. QMIX [17] extends VDN through employing a mixing network for the agent utilities with monotonic constraints, which allows different mixtures in different states so that the central value can be maximized independently. These two methods, condition on an agent's utility on its history, i.e. the past observations and actions and share the parameters of all utility functions as shown in Figure 1(a). DCG [1] is another decentralized agent execution approach which uses the addition of one extra linear layer which is used to approximate a coordination graphs (CG) *pairwise payoffs*. For DCG, parameter sharing is employed to approximate *all* payoff functions in the same neural network as shown in 1(b). QTRAN [18] is another approach which attempts to calculate the central Q-learning through applying linear constraints between agent utilities and joint action values, which removes structural constraints of factorizable MARL tasks.

Other Related Multi-Agent Approaches. Our work primarily examines cooperative multi-agent reinforcement learning using Q-learning, however there are other multi-agent which leverage actor-critic and deterministic policy gradient approaches which operate under the CTDE framework. Multi-Agent DDPG (MADDPG) [11] is a variation of the deep deterministic policy gradient approaches for MARL. In this setting, each agent has its own actor and critic. However the critic is trained on the joint observation and action of all agents to approximate the joint state-action value function. Another approach is Shared Experience Actor Critic (SEAC), where each agent again has its own actor and critic, however it is assumed that local policy gradients of agents provide useful learning directions for all agents, allowing agents to learn from experiences of other agents [3]. In order to determine the ideal manner to have a centralised critic, a QMIX-inspired *mixing critic* approach has been proposed called Learning Implicit Credit Assignment (LICA), which uses *adaptive entropy regularization* in order to maximize joint value through cooperation [23].

Individual-Global-Max (IGM) Condition and Factorizable Tasks. For CTDE methods, we define the individual agent utilities by $q^i, i \in \mathcal{A}$. An important concept for these methods is *decentralizability* (see IGM in [18]) which asserts that $\exists q^i$, such that $\forall s, \mathbf{u}$, $\arg\max_{\mathbf{u}} Q^*(s, \mathbf{u}) = (\arg\max_{u^1} q^1(\tau^1, u^1), \ldots, \arg\max_{u^n} q^n(\tau^n, u^n))$. VDN and QMIX demonstrate two sufficient methods for factorizing Q^* for IGM can be accomplished through additivity and monotonicity respectively. QTRAN has demonstrated that any joint action-value function $Q^*(s, \mathbf{u})$ satisfies IGM condition if

$$\mathscr{Q}(s, \mathbf{u}) - Q^*(s, \mathbf{u}) + \mathscr{V}(s) = \begin{cases} 0, & \text{if } \mathbf{u} = \bar{\mathbf{u}} \\ \geq 0, & \text{if } \mathbf{u} \neq \bar{\mathbf{u}} \end{cases} \tag{1}$$

where $\bar{u}^i$ is the optimal local action $\arg\max_{u^i} q^i(\tau^i, u^i)$ and $\bar{\mathbf{u}} = \{\bar{u}^1, \ldots, \bar{u}^n\}$, $\mathscr{Q}(s, \mathbf{u} = \sum_{i=1}^n q^i(\tau^i, u^i)$ [1], $\mathscr{V}$ is the state-value network defined as $\mathscr{V}(s) = \max_{\mathbf{u}} Q^*(s, \mathbf{u}) - \sum_{i=1}^n q^i(\tau^i, \bar{u}^i)$. QTRAN uses a similar approach to *Deep Q-Networks* DQNs [14], which uses a *replay memory* to store the transition tuple $\langle s, u, r, s' \rangle$, where state s' is observed after action u in state s with reward r. The parameters of the action-value function, θ is learned by sampling batches from the replay memory and minimising the following:

$$\begin{aligned} \mathcal{L}(\theta) :=& (Q^*(s, \mathbf{u}) - y^{\text{DQN}})^2 + \left(\mathscr{Q}(s, \mathbf{u}) - \perp Q^*(s, \bar{\mathbf{u}}) + \mathscr{V}(s)\right)^2 \\ &+ \Big(\min\left(\mathscr{Q}(s, \mathbf{u}) - \perp Q^*(s, \mathbf{u}) + \mathscr{V}(s), 0\right)\Big)^2 \end{aligned} \tag{2}$$

where $y^{\text{DQN}} = r + \gamma \max_{u'} Q^*(s', u'; \theta^-)$. θ^- is the parameter of a *target network* that are periodically copied from θ and kept constant for a number of iterations, and $\perp$ is the 'detach' operator which stops the gradient flow through to Q^*, which ensures Q^* is fixed.

2.1 Coordination Graph

In our framework, each agent i has an associated *payoff* f^i. An agent's local utility may be influenced by other agent's actions and state, if we assume that the payoffs are independent utilities then $f^i \equiv q^i$

[1] noting that s is the global state, and τ^i is the partial observable state for the agent i

as shown in Figure 1(a) for QMIX, VDN, LICA models. We define $Scope(f^i) \subset S \cup \mathbf{U}$ to be set of state variables and action variables that influence an agent's local utility q^i. We further define the relevant agent decision variables to be $Relevant(f^i) = \{a^j \in \mathcal{A} | a^j \in Scope(f^i)\}$. Intuitively, a coordination graph connects agents whose local utility functions interact with each other. Then extending the definition of coordination graph to the undirected counterpart proposed by Guestrin et al.[5]:

Definition 2.1 *A coordination graph for a set of agents with local payoff* $\{f^i, \ldots, f^n\}$ *is an undirected graph whose nodes are* $\{a^1, \ldots, a^n\}$ *which contains an edge* $\{a^i, a^j\}$ *if and only if* $a^i \in Relevant(f^j)$ *or* $a^j \in Relevant(f^i)$.

Coordination graphs depend only on state and thus every state can have its own unique CG. DCG constructs the coordination graph through consideration pairwise payoffs Figure 1(b), whilst QTRAN constructs a coordination graph through a single hyper-edge with no factorisation Figure 1(d). Our approach as shown in Figure 1(c), where the nodewise (or agent) payoff is constructed through consideration the whole state space and all agent actions. Then the total utility will depend on the setup of the respective coordination graph, as shown in Figure 1(b, c, d).

2.2 Graph Neural Networks

In applying graph neural networks to the centralized training phase, in a manner which is amenable to factorization, we need to ensure that all operations are permutation invariant. In our paper, we represent the undirected graph (as defined by Definition 2.1) G as $\langle A, F \rangle$, where $A \in \{0,1\}^{n \times n}$ is a symmetric adjacency matrix, and $F \in \mathbb{R}^{n \times d}$ is the node feature matrix assuming each node has d features.

Graph Convolution Network. A graph convolution network [9] (GCN) is a network which takes as input the feature matrix and the adjacency information in order to summarize the attributes of each node and output a node-level feature matrix. This operates in a similar way to how convolution operations are used where kernels are convolved across local regions of the input to produce feature maps. GCNs use layer-wise forward-propagation operation defined by $X_{(\ell+1)} = \sigma(\hat{A} X_{(\ell)} W_{(\ell)}) \in \mathbb{R}^{n \times d_{\ell+1}}$. Where $X_{(\ell)} \in \mathbb{R}^{n \times d_\ell}$ represents the feature matrix of layer ℓ with d_ℓ features, where the feature matrix at layer 0 is initialized by $X_{(0)} = F$, and where $\hat{A} = D^{-\frac{1}{2}}(A + I)D^{-\frac{1}{2}}, D_{ii} = \sum_j (A + I)_{ij}$. We denote the trainable weight matrix as $W_\ell \in \mathbb{R}^{d_\ell \times d_{\ell+1}}$ which applies a linear transformation to the feature vectors which is a weight matrix, and σ is the activation function. GCN is naturally permutation invariant and can be used to addressed the permutation invariance challenge in CG.

Differentiable Pooling. Whilst GCN is used for a *node level* prediction, our challenge involved generating a *graph level* prediction for the total payoff. One naive approach, is to approximate the total payoff by taking a sum, i.e. $\sum_i q^i$ [6]. Differentiable Pooling (DiffPool) [22] allow for graph coarsening through a learnable pooling strategy. DiffPool is a module which pools nodes given a (learned) assignment matrix which uses a layer-wise forward-propagation operation. Let the number of nodes at layer ℓ be n_ℓ, then DiffPool generates a new graph $\langle A_{(\ell+1)}, Z_{(\ell+1)} \rangle$ at layer $(\ell + 1)$, defined by $A_{(\ell+1)} = V_{(\ell)}^\top A_{(\ell)} V_{(\ell)} \in \mathbb{R}^{n_{\ell+1} \times n_{\ell+1}}$ and $Z_{(\ell+1)} = V_{(\ell)}^\top X_{(\ell+1)} \in \mathbb{R}^{n_{\ell+1} \times d_\ell}$. Where $V_{(\ell)} \in \mathbb{R}^{n_\ell \times n_{\ell+1}}$ is the learned cluster assignment matrix at layer ℓ. Then $\langle A_{(\ell+1)}, Z_{(\ell+1)} \rangle$ fully defines the new coarsened graph with $n_{\ell+1}$ nodes. $A_{(i)}$ is a real matrix, which represents the strength of fully connected edge-weighted graph. To enforce $V_{(\ell)}$ to be a proper assignment matrix, softmax function can be applied in a row-wise fashion to ensure that the assignments generated are probabilistic in nature. The learned cluster assignment matrix can be generated using a GCN layer, $V_{(\ell)} = \bar{\sigma}(A X_{(\ell)} \bar{W}_{(\ell)})$ where the learnable weight matrix $\bar{W}_{(\ell)} \in \mathbb{R}^{n_\ell \times n_{\ell+1}}$, and $\bar{\sigma}$ is the softmax activation function (i.e. the assignments are probabilistic). This approach has been demonstrated to be permutation invariant[22].

3 Proposed Method

The key idea behind QCGraph is to generate dynamic coordination graphs as part of the mixing network in the centralized training phase of the CTDE problem. In this section we explore how dynamic adjacency matrices can be generated and demonstrate how it is applied to cooperative Dec-POMDP problems.

3.1 Dynamic Adjacency Matrix Generation

In the previous section, both GCN and DiffPool assume a provided adjacency matrix as part of the input to the graph neural network. In the setting of coordination graphs, the adjacency matrix is defined if and only if the agents are *relevant* to each other. In this instance, we aim to have a different and dynamic adjacency matrix depending on how agents and state are reflected in the game.

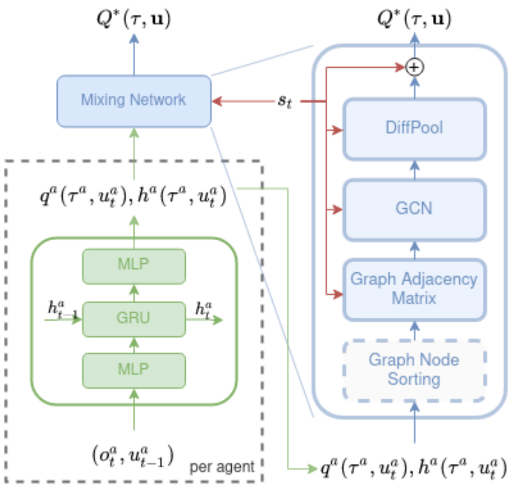

Figure 2: QCGraph network structure. In blue is the mixing network, and in green is the agent network structure. On the left is the overall network structure, on the right is the mixing network, where h represents the hidden layer of the agent utility network. The dotted outline around "Graph Node Sorting" indicates that there are no learnable parameters as part of this operation.

3.1.1 Adjacency matrix generation

We denote the learned node assignment matrix for the feature matrix as $\mathcal{V} \in \mathbb{R}^{d \times n}$ for feature matrix $F \in \mathbb{R}^{n \times d}$. Each column of $\mathcal{V}$ represents one of the n agent nodes. With a given assignment matrix, the generate adjacency matrix is defined as $A = \sigma(F\mathcal{V}) \in \mathbb{R}^{n \times n}$. Where σ is chosen to be a suitable activation function. Intuitively, $\mathcal{V}$ provides a soft assignment based on the feature matrix. The values of the adjacency matrix denote the connectivity strength between each pair of nodes. Then for the node pairs $\{a^i, a^j\}$, $\widetilde{A}$ will denote the measure of *relevance*. If the choice of σ is the sigmoid function, then elements i, j in $\widetilde{A}$ will denote $Prob\big(a^i \in Relevant(f^j)\big)$. This output can be used as-is as input to further graph neural network layers.

3.1.2 Addressing Permutation invariance

For the adjacency matrix to be useful for graph classification, the generation of the adjacency matrix should be invariant under node permutations. In order to address this, in our construction of the dynamic adjacency matrix, the node feature matrix should be pre-sorted before computing the adjacency matrix. This can be approached as a variable to k-max pooling used in Graph U-Nets[4], where the node feature matrix is projected to 1D through a trainable projection vector $\mathbf{p}$ given by $y^i = F^i\mathbf{p}/\|\mathbf{p}\|$. Then each node can be sorted according to the scalar projection value y^i.

3.1.3 Adjacency Matrix Generation with Hard Assignments

With all these components in place, we can construct our approach to generate hard assignments specifically for coordination graph problem. In order to ensure that hard assignments are generated, we use the gumbel-softmax trick[8, 12] and the Heaviside step function which are denoted by σ_{gumbel}, σ_{step} respectively. As we presume our coordination graph is undirected, symmetry is achieved through updated the adjacency matrix $\widetilde{A} = \max(\widetilde{A} + \widetilde{A}^\top, 1)$. This approach is given in pseudo-code form in Algorithm 1.

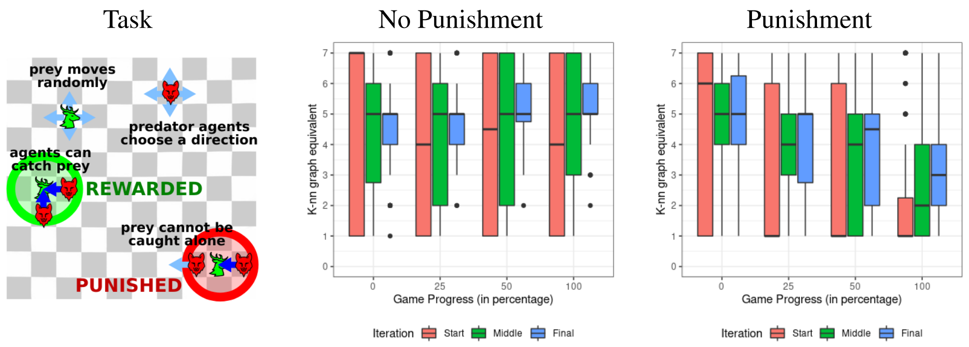

Figure 3: Visualized k-nearest neighbor adjacency matrix representation in QCGraph in the modified predatory prey task (left) over two environments where there is no penalty for catching a prey alone (middle), and where there is a penalty (right). The box plots represents the distribution of the dynamic k-nearest neighbor adjacency matrix representation generated during training at the start, middle and end of training for the task. The k is determined by the number of agents deemed *Relevant* according to Definition 2.1. In the scenario where task emits no punishment, QCGraph tends towards QTRAN-style model (i.e. higher k) (middle), on the other hand, when the task emits punishment, QCGraph favors coordination with lower number of agents, tending towards DCG-style model.

Algorithm 1 Adjacency Matrix Generation with Hard Assignments

```
Require: $F, S$, with learnable weights $\mathbf{p}_{\text{node}}, \mathbf{p}_{\text{state}}, \mathcal{V}$
  1: $F \leftarrow \text{nodesort}(F, \mathbf{p}_{\text{agent}})$ {Sort nodes based on projection derived from node feature matrix as in Section 3.1.2}
  2: $y_{\text{state}} \leftarrow S\mathbf{p}_{\text{state}} \in \mathbb{R}$ {The state specific bias through calculating projection derived from state $S$}
  3: $\widetilde{A}_{\text{agent}} \leftarrow \sigma_{\text{gumbel}}(F\mathcal{V}), \widetilde{A}_{\text{state}} \leftarrow \sigma_{\text{step}}(F\mathcal{V} + y_{\text{state}})$ {Compute adjacency matrices with and without bias as in sec. 3.1.1}
  4: $\widetilde{A} \leftarrow \widetilde{A}_{\text{agent}} + \widetilde{A}_{\text{state}}$
  5: $\widetilde{A} = \max(\widetilde{A} + \widetilde{A}^{\top}, 1)$ {Ensure symmetry in undirected coordination graph, with an element-wise max function}
Ensure: $\widetilde{A}$
```

The application of the heaviside step function discretizes $Prob(a^i \in Relevant(f^j))$ according to threshold determined by b_{state}, it is equivalent to performing

$$\widetilde{A}_{\text{state}} = \begin{cases} 1, & \text{if } \sigma_{\text{sigmoid}}(F\mathcal{V}) \geq b_{\text{state}} \\ 0, & \text{if } \sigma_{\text{sigmoid}}(F\mathcal{V}) < b_{\text{state}} \end{cases}$$

Recall in Section 2.1, we define an agent to be $Relevant$ to payoff for agent i is defined by $Relevant(f^i) = \{a^i \in \mathcal{A} | a^i \in Scope(f^i) \subset S \cup \mathbf{U}\}$. This suggests the importance of including global state information to determine the CG from one step in our environment to another. An example of this is if there is an immediate threat outside a set of agent's partially observable state, but can be observed from a global level.

3.2 QCGraph

We introduce the Q-value Coordination Graph (QCGraph), which learns the utility and payoff functions of a coordination graph. This approach aims to unify other CG approaches (QTRAN and DCG) in an overall framework, as well as remove constraints on having fixed pre-determined coordination structures. Figure 2 illustrates the overall setup for QCGraph.

The agent network (shown in green, or bottom-left side of Figure 2) describes the agent utility network. Each agent network represents its individual utility function $q^i(\tau^i, u^i)$. At each time step, each agent network receives a local observation as input followed by a GRU recurrent layer and a fully-connected layer with $|U|$ outputs. This setup is the same one used in QMIX, LICA, DCG [17, 23, 1]. The output of the utility network is both the utility value itself, q^i and also the utility network's hidden feature layer. This approach of emitting both the utility value and recurrent network's hidden state, h^i to the mixing network for centralized learning is used in both QTRAN and DCG [1, 18].

The mixing network (shown in blue, or right side of Figure 2). The first "module" is the graph node sorting operation. For each agent i, as q^i can be interpretted as a 1D projection of the hidden state h^i, this can be used to pre-sort the node feature matrix as required for permutation invariance as part of the approach outlined by Equation 3.1.2. The next hypernetwork generates the graph adjacency matrix as outlined by Algorithm 1. The learnable parameters consists of a single linear layer and the state bias, as the node feature matrix project (step 1. in Alg 1) is subsumed in the first module of the mixing network. The next two modules are the GCN and DiffPool modules explained in Section 2.2, which are constructed with one layer each (that is, the GCN only consists of one forward layer, and DiffPool consists of one forward layer to pool all nodes). The final state bias is constructed through a two layer hypernetwork with ReLU non-linearity, in the same manner used by QMIX. Through the use of graph neural network techniques in its setup, QCGraph generally would have comparable number of parameters or in most cases less than QMIX[2].

The agents and mixing network are trained in a centralized manner, and each agent uses its own utility function q^i to take an action during decentralized execution. QCGraph is trained end-to-end to minimize the loss as defined by QTRAN [18] as described by Equation 2, which ensures that QCGraph satisfies IGM condition.

In general CG approaches are used to prevent *relative overgeneralization* during the exploration phase of agents. DCG and QTRAN both enable estimating a value function over a joint action space. This approach is useful in certain environments where cooperation is important. For example, in the predator-prey environment, the predators may need to cooperate together in order to capture large or dangerous prey. From the perspective of each predator, the expected reward will depend on the action of the other agent. An example of this is shown in Figure 3, which demonstrates QCGraph's representational ability to adapt and build structures which transition towards more rewarding joint behaviour which existing CG approaches would struggle to accomplish. In models which do not take into account joint actions, (QMIX, IQL) it is not possible to learn the optimal policy. Furthermore QCGraph has the ability to dynamically adapt the level of coordination graph over various states which becomes important in wargame scenarios where the level of appropriate cooperation may differ according to the formation or number of agents available at a particular point in time.

4 Experiments

In our experiments, we study how QCGraph compares against other CG approaches and against other MARL algorithms in a variety of benchmark tasks leveraging the PettingZoo library [21]. More detailed descriptions of these environments and the hyperparameters used in training can be found in Appendix A.

4.1 Modified Predator Prey

The modified predator prey task was introduced in Section 3.2, with the QCGraph representation shown in Figure 3. The task involves moving agents controlled by decentralized agents which need to cooperate in order to catch prey. If the agents capture prey alone, they get either get no reward (Predator Prey w/o Negative Rewards) or a negative reward (Predator Prey Negative Rewards). The purpose of this environment is to compare QCGraph with other CG approaches, being QTRAN and DCG. From the results in Figure 4(a), we observe in both scenarios the CG approaches (QCGraph, QTRAN, DCG) outperform QMIX and IQL. However, in the scenario with negative rewards in Figure 4(a), there is a larger degradation in performance for QTRAN and in particular DCG. This demonstrates the increased flexibility QCGraph allows which allows it to prevent *relative overgeneralization*.

Ablations We explore ablations in relation to the choice of network construction in QCGraph as shown in Figure 4(b). We denote the model "QCG-obs" to explore the choice of $\tilde{h}$, whether the local observation or hidden layer transformation, we label the model "QCG-nosort" for the

[2]In QMIX, the state bias is projected to every agent in the hypernetwork. This is replaced with graph neural network modules instead. More details are provided in supplementary materials.

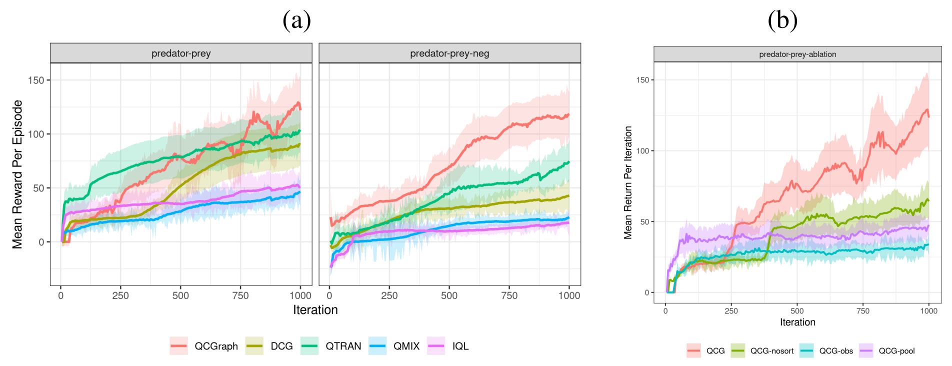

Figure 4: (Left) Performance comparison in the modified predator prey task first introduced in Figure 3, where *predator-prey* is the environment with no negative penalty, and *predator-prey-neg* has negative penalty. Plots show comparison across Q-learning approaches. Error bars represent the standard deviation of the mean return. (Right) Ablation analysis to justify our choice of neural network architecture. The environment is the variation where no negative penalty is provided.

removal of node sorting, and "QCG-pool" for the removal of DiffPool, which is replaced with global sum pooling instead (analogous to scenario in [1] and [18]). We perform all our ablation experiments on the Predator Prey w/o Negative Rewards task described above. Without pre-sorting the nodes, the convergence is slower as it does not need to encode all possible node permutations, without graph neural network module, the agent fails to capture agent relationships, using the local observation instead of hidden layer may result in lower quality policies, as suggested in QTRAN [18] yields improved performance, which suggests noise or redundant information would be removed as part of the hidden layer representation. The use of DiffPool also allows for information to propagate for cooperative learning over global pooling layer. This demonstrates the importance in our mixing network choices and construction. Additional ablation studies were performed using the PettingZoo environments which demonstrates consistency across variety of environment and is shown in Appendix in Figure 7.

4.2 PettingZoo Benchmarks

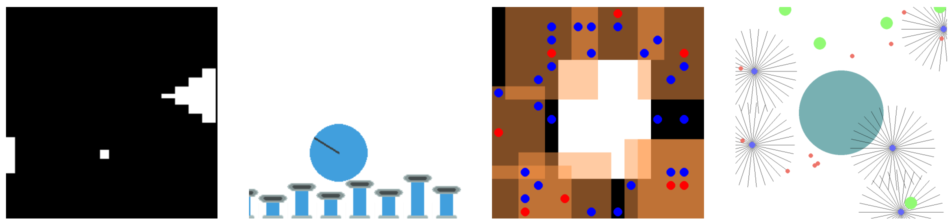

Figure 5: Left to right: Pong, Pistonball, Pursuit, Waterworld environments.

PettingZoo offers a unified API across a multitude of MARL environments to facilitate benchmarking MARL algorithms. We access a variety of benchmark algorithms using standardized preprocessing as suggested in the original benchmarks [21]. We assess a wide variety of environments from different MARL benchmark tasks including "Butterfly" [21], "MPE" [11] and "SISL" [7] environments, with their tasks depicted in Figure 5. We also use variations of the environment which lower the number of agents by 25%, which have been marked by *medium* suffix in the scenarios where the default agents exceeded 4.

The "Butterfly" cooperative tasks used were *cooperative pong* and *pistonball*. In *cooperative pong* the aim is to keep the ball in play, and the game is over if the ball goes out of bounds from either the left or right edge of the screen.

	QCGraph	QMIX	IQL	SEAC	MADDPG	LICA
reference	10.64 ± 14.94	3.41 ± 18.27	-4.1 ± 16.38	*12.76 ± 17.92*	9.71 ± 16.31	**27.84 ± 19.21**
spread	*30.35 ± 28.08*	18.56 ± 27.11	12.13 ± 28.71	**42.85 ± 28.7**	19.64 ± 28.81	**43.7 ± 29.66**
pursuit	**244.2 ± 30.33**	157.13 ± 31.07	96.39 ± 32.04	168.24 ± 29.91	*179.69 ± 30.04*	**248.03 ± 29.17**
waterworld	**204.59 ± 30.24**	126.35 ± 28.87	48.78 ± 27.14	135.7 ± 27.88	156.99 ± 29.61	**206.73 ± 29.52**
pistonball	**286.02 ± 43.54**	187.43 ± 45.85	158.62 ± 39.87	*193.56 ± 49.65*	158.18 ± 30.79	172.06 ± 34.83
pong	28.91 ± 13.85	28.39 ± 8.85	22.25 ± 3.82	*46.84 ± 3.01*	**55.43 ± 5.02**	22.25 ± 2.94
pursuit$_{\text{medium}}$	**169.57 ± 30.71**	29.24 ± 23.46	28.57 ± 20.91	100.84 ± 31.08	123.8 ± 29.45	*161.51 ± 29.67*
waterworld$_{\text{medium}}$	**193.49 ± 30.45**	22.69 ± 21.87	32.08 ± 24.63	147.73 ± 28.21	20.39 ± 22.27	*192.47 ± 29.58*
pistonball$_{\text{medium}}$	165.73 ± 55.94	58.52 ± 45.36	82.38 ± 24.95	*243.47 ± 52.34*	136.8 ± 26.95	**296.77 ± 65.96**
pong$_{\text{base}}$	29.14 ± 14.87	27.42 ± 2.96	24.37 ± 8.08	**40.74 ± 13.61**	*32.81 ± 3.79*	21.2 ± 3.27
MRR$_{\text{default only}}$	**0.524**	0.233	0.181	0.524	0.405	*0.512*
MRR$_{\text{overall}}$	**0.561**	0.291	0.189	0.462	0.348	*0.553*

Table 1: Final Evaluation Returns with standard deviation on a selection of tasks. Highest and second highest returns are shown in bold and italics respectively. The mean reciprocal rank (MRR) is provided (higher is better) to allow for comparison across different MARL approaches. MRR (default only) refers to the default PettingZoo environments, whilst MRR (overall) includes the additional modified PettingZoo environments.

The "MPE" cooperative tasks used were *reference* and *spread*. The *reference* task aimed to move agents towards target landmark which is known only by the other agents. In the *spread* task, agents learn to cover all landmarks while avoiding collisions with one another.

The "SISL" cooperative tasks used were *pursuit* and *waterworld*. In pursuit, the agents control pursuer agents which aim to capture randomly moving evader agents; similar to the predator prey task. An evader agent is captured when 2 or more pursuers surround it. In *waterworld* the pursuer agents aim to consume food while avoiding randomly moving poison.

We assess our approach, QCGraph, against QTRAN, QMIX, IQL, SEAC, LICA, and MADDPG algorithms, using the Q-learning approaches as our baseline, and comparison against the state-of-the-art MARL algorithms using actor-critic approaches.

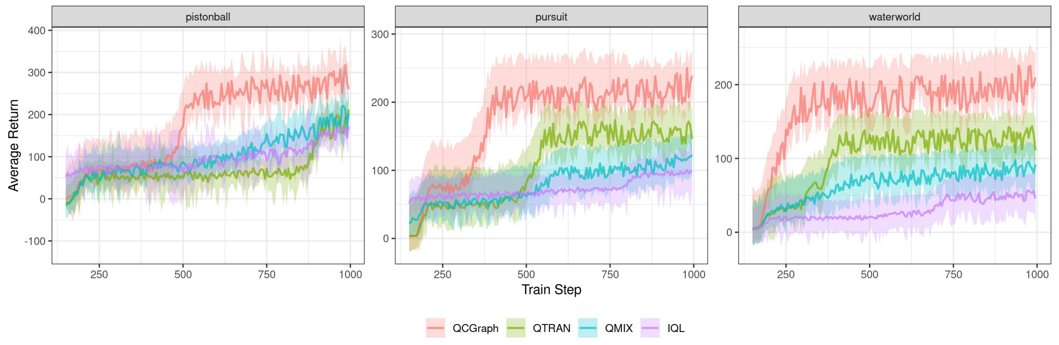

Figure 6: Performance of QCGraph against other Q-learning approaches over a variety of benchmark environments.

Q-learning Baseline. The performance of QCGraph against other Q-learning algorithms is shown in Figure 6. We observe that QCGraph yields higher returns across a range of environments. In the environments similar to predator-prey with penalties (*pursuit* and *waterworld*), we observe QTRAN generally outperforms QMIX and IQL, which is reflected in the QTRAN results [18], but fails to yield the same level of consistency in other environments [1, 13, 23], in this particular sets of tasks, QTRAN notably performs the worst on *pistonball* of all Q-learning approaches. QCGraph's considerable performance improvements over Q-learning approaches confirm the effectivenss of our proposed methods.

Comparison with Actor-Critic Approaches. The performance of QCGraph against other state-of-the-art MARL approaches is shown in Table 1. We observe that QCGraph achieves competitive results across all tasks over a wide variety of cooperative MARL environments. Although QCGraph did not “win” more tasks than LICA algorithm, it provided more consistent results across all scenarios, demonstrated by its relative higher mean reciprocal rank. We hypothesis this is due to the ability for QCGraph to dynamically alter its coordination structure during training to yield the ideal agent formulation in the Q-learning settings.

5 Conclusion

This paper introduced QCGraph, an architecture for value factorization over dynamically generated *coordination graphs*, which can be maximized through creating subgraphs dynamically. We evaluated our approach on a variety of environments to demonstrate the efficacy of our approach. We demonstrate that our approach improves over existing coordination graph approaches experimentally, in both data efficiency and representational capability. We demonstrate how Graph Neural Networks approaches through node ordering and DiffPool can be used to resolve agent reshuffling problem, as well as expanding multi-agent reinforcement learning to a dynamic cooperative environment at both a local and global level.

Acknowledgements

This research is supported by ONRG research grant (N62909-19-1-2141).

References

[1] Wendelin Boehmer, Vitaly Kurin, and Shimon Whiteson. Deep coordination graphs. In *ICML 2020: Proceedings of the Thirty-Seventh International Conference on Machine Learning*, 2020.

[2] Jacopo Castellini, Frans A Oliehoek, Rahul Savani, and Shimon Whiteson. The representational capacity of action-value networks for multi-agent reinforcement learning. In *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems*, pages 1862–1864. International Foundation for Autonomous Agents and Multiagent Systems, 2019.

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

[4] Hongyang Gao and Shuiwang Ji. Graph u-nets. In *International Conference on Machine Learning*, pages 2083–2092, 2019.

[5] Carlos Guestrin, Michail G Lagoudakis, and Ronald Parr. Coordinated reinforcement learning. In *Proceedings of the Nineteenth International Conference on Machine Learning*, pages 227–234. Morgan Kaufmann Publishers Inc., 2002.

[6] Carlos Guestrin, Shobha Venkataraman, and Daphne Koller. Context-specific multiagent coordination and planning with factored mdps. In *Eighteenth National Conference on Artificial Intelligence*, page 253–259, USA, 2002. American Association for Artificial Intelligence.

[7] Jayesh K Gupta, Maxim Egorov, and Mykel Kochenderfer. Cooperative multi-agent control using deep reinforcement learning. In *International Conference on Autonomous Agents and Multiagent Systems*, pages 66–83. Springer, 2017.

[8] Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax. In *International Conference on Learning Representations*, 2018.

[9] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

[10] Landon Kraemer and Bikramjit Banerjee. Multi-agent reinforcement learning as a rehearsal for decentralized planning. *Neurocomputing*, 190:82–94, 2016.

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

[12] Chris J. Maddison, Andriy Mnih, and Yee Whye Teh. The Concrete Distribution: A Continuous Relaxation of Discrete Random Variables. In *International Conference on Learning Representations*, 2018.

[13] Anuj Mahajan, Tabish Rashid, Mikayel Samvelyan, and Shimon Whiteson. Maven: Multi-agent variational exploration. In *Advances in Neural Information Processing Systems 32*, pages 7613–7624. Curran Associates, Inc., 2019.

[14] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[15] Frans A. Oliehoek and Christopher Amato. *A Concise Introduction to Decentralized POMDPs*. Springer Publishing Company, Incorporated, 1st edition, 2016.

[16] Frans A Oliehoek, Matthijs TJ Spaan, and Nikos Vlassis. Optimal and approximate q-value functions for decentralized pomdps. *Journal of Artificial Intelligence Research*, 32:289–353, 2008.

[17] 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. *International Conference on Machine Learning*, 2018.

[18] Kyunghwan Son, Daewoo Kim, Wan Ju Kang, David Earl Hostallero, and Yung Yi. Qtran: Learning to factorize with transformation for cooperative multi-agent reinforcement learning. *International Conference on Machine Learning*, 2019.

[19] Peter Sunehag, Guy Lever, Audrunas Gruslys, Wojciech Marian Czarnecki, Vinicius Zambaldi, Max Jaderberg, Marc Lanctot, Nicolas Sonnerat, Joel Z Leibo, Karl Tuyls, et al. Value-decomposition networks for cooperative multi-agent learning based on team reward. *International Foundation for Autonomous Agents and Multiagent Systems*, pages 2085–2087, 2018.

[20] Ming Tan. Multi-agent reinforcement learning: independent versus cooperative agents. In *Proceedings of the Tenth International Conference on International Conference on Machine Learning*, pages 330–337. Morgan Kaufmann Publishers Inc., 1993.

[21] Justin K Terry, Benjamin Black, Mario Jayakumar, Ananth Hari, Luis Santos, Clemens Dieffendahl, Niall L Williams, Yashas Lokesh, Ryan Sullivan, Caroline Horsch, and Praveen Ravi. Pettingzoo: Gym for multi-agent reinforcement learning. *arXiv preprint arXiv:2009.14471*, 2020.

[22] Zhitao Ying, Jiaxuan You, Christopher Morris, Xiang Ren, Will Hamilton, and Jure Leskovec. Hierarchical graph representation learning with differentiable pooling. *Advances in Neural Information Processing Systems*, pages 4800–4810, 2018.

[23] Meng Zhou, Ziyu Liu, Pengwei Sui, Yixuan Li, and Yuk Ying Chung. Learning implicit credit assignment for multi-agent actor-critic. *Advances in Neural Information Processing Systems*, 2020.

A Experimental Details

A.1 Environment Details

All environments used were the default settings with the exception of *pistonball*, where we lowered the number of agents from 20 to 10, due to the nature of the mixing networks. The non-standard environments used are as outline; *pong-base* describes the environment where the tiered cake paddle was not used. For consistency with all other tasks, waterworld action space was discretized so that all algorithms and their variants could be used. We evaluated on 1000 evaluation episode rollouts (separate from the train distributions) every training iteration and used the average score and variance for the plots and tables, and is shown in Figure 8. Additional ablation studies to demonstrate the consistency of our neural architecture choices are shown in Figure 7.

A.2 Hyper-parameters

The policies used leverage the default hyper-parameters based on the official implementation of LICA, QMIX, IQL, QTRAN their accompanying code based on the pymarl libary [17, 23, 18]. SEAC was reimplemented in rlkit framework with the same hyperparameters proposed in their paper, however using Soft Actor-Critic instead of A2C. Similarly, MADDPG is extended the rlkit implementation of DDPG to support multi-agent setting. In order to facilitate discrete action spaces gumbel softmax

Hyperparameters	Value
Policy Hidden Sizes	[64, 64, 64]
Mixer Hidden Sizes	[32, 32, 32]
Policy Hidden Activation	ReLU
Target Network τ	0.005
Learning Rate	0.0003
Batch Size	256
Replay Buffer Size	1000000
Number of pretraining steps	1000
Steps per Iteration	1000
Discount	0.99
Reward Scale	1

Table 2: Hyper-parameters used for MARL experiments

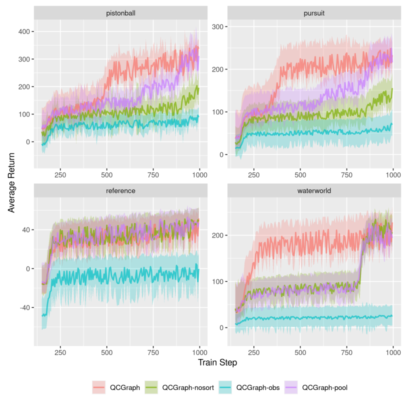

Figure 7: Ablation over PettingZoo environments.

[8, 12] was used as suggested by [3]. Similar to the usage in the pymarl library, parameter sharing was used in all models in order to improve the general performance of the agents.

The preprocessing used the same set of preprocessing for each PettingZoo environment as suggested by [21], which involved resizing observations to 84×84 images with linear interpolation, converting to grayscale, then normalized. Further global hyperparameter details are provided in Table 2.

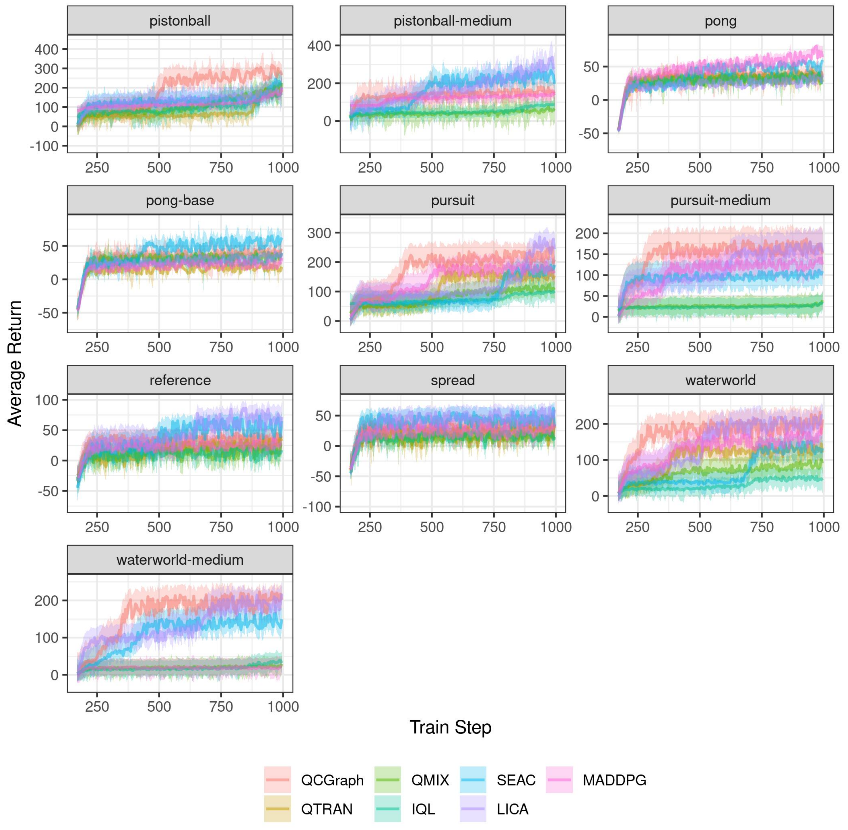

Figure 8: Performance over a variety of benchmark environments.