A Full Background

In this appendix, for technical and theoretical completeness, we expand upon information given in the background of the paper.

A.1 Multi-Agent Reinforcement Learning

We consider the case of Decentralized Partially Observable Markov Decision Processes (Dec-POMDPs) [28] augmented with communication between agents. In a Dec-POMDP, at each timestep t every agent $i \in \{1, ..., N\}$ gets a local observation o_i^t, takes an action a_i^t, and receives a reward r_i^t. The objective is to maximize the agent rewards over the actions. We consider two agent paradigms: value-based and actor-critic.

In value-based methods, such as MADQN [53], the aim is to learn a function Q_θ with parameters θ that estimates the value of taking an action a_i after observing o_i. Such methods are often trained using a replay buffer, to which tuples (O, A, O', R) are added, where $O = \{o_1, ..., o_N\}$ is the set of observations, A is the set of actions, O' is the set of next observations, and R is the set of rewards. If the environment includes communication, the underlying communication graph C can also be included in the replay buffer. These experiences are added to the buffer whilst the agents interact with the environment. To collect diverse experiences, methods such as ϵ-greedy exploration [55] can be used. For training, random minibatches of size B are sampled from the buffer, and loss similar to the following is minimized:

$$L(\theta) = \frac{1}{B}\sum_{b=1}^{B}\frac{1}{N}\sum_{i=1}^{N}(y_i - Q_\theta(o_i, a_i))^2$$

where $y_i = r_i + \gamma \max_{a'} Q_{\theta'}(o_i', a_i')$. In this formula, $Q_{\theta'}$ is referred to as the *target network*, and its parameters θ' are updated softly or intermittently from θ during training.

In actor-critic methods such as MADDPG [32], the aim is to learn a policy function π_θ that maps observations onto distributions over actions, where the action most likely to maximize the reward is assigned the highest probability. The policy gradient is estimated by the following:

$$\nabla_\theta J(\theta) = \mathbb{E}_{o\sim\rho^\pi,\ a\sim\pi_\theta}[\sum_{t=1}^{T}\nabla_\theta \log \pi_\theta(a_t|o_t)(R_t - V(o_t))]$$

where ρ^π is the observation distribution, π_θ is the policy distribution, $R_t = \sum_{t'=t}^{T}\gamma^{t'-t}r(s_{t'}, a_{t'})$ is the discounted reward, and V is a learned value function, used to decrease the variance of the estimated policy gradient.

For brevity, we collectively refer to the policy network / value function or Q-network as the *actor network*. Often in MARL, instead of learning an actor network for each agent, a single shared network will be used for all agents. For example, COMA [15], Q-Mix [45], and Mean Field RL [62] all share parameters in their neural networks. This parameter sharing typically yields faster and more stable training [18].

A.2 Communication in MARL

Many environments require agents to coordinate to solve tasks. Communication is crucial for enabling this. Foerster et al. [14], Sukhbaatar et al. [51] were among the first to propose learned communication in multi-agent reinforcement learning. Since then, many different methods for communication in MARL have been proposed [67]. When communicating, there is a helpful structural inductive bias which can be used: the order in which incoming messages from other agents are processed should not affect the outcome. More formally: agents ought to be *permutation invariant* when using incoming messages. A function ρ is permutation invariant if for any input $X = (x_1, x_2, ..., x_k)$ and any permutation σ on X, $\rho(X) = \rho(\sigma \circ X)$.

In Section 3.1, we define Graph Decision Networks (GDNs) using the framework of GNNs, a neural architecture which respects permutation invariance between nodes when doing message passing.

Many of the most successful models for MARL communication fall within this paradigm, including CommNet [51], IC3Net [49], GA-Comm [29], MAGIC [37], Agent-Entity Graph [2], IP [44], TARMAC [9], IMMAC [52], DGN [24], VBC [64], MAGNet [33], and TMC [65]. Other models such as ATOC [23] and BiCNet [43] do not fall within the paradigm since they use LSTMs for combining messages, which are not permutation invariant, and models such as RIAL, DIAL [14], ETCNet [21], and SchedNet [25] do not since they used a fixed message-passing structure.

A.3 Graph Neural Networks

A graph consists of nodes and edges connecting them. Nodes and edges can have attributes (also referred to as features or labels), which often take the form of real vectors. Formally, we define an attributed graph G as a triple (V, E, a), where $V(G)$ is a finite set of nodes, $E(G) \subseteq \{(u, v) \mid u, v \in V(G)\}$ is a set of directed edges, and $a : V(G) \cup E(G) \to \mathbb{R}^d$ is an attribute function where $d > 0$. For $w \in V(G) \cup E(G)$, $a(w)$ is the attribute of w. Undirected graphs are ones in which $E(G)$ is a symmetric relation on $V(G)$. Graphs are found in many different areas of application, leading to a plethora of research of how to learn using graph structured data [3, 13, 48].

When considering functions operating on graphs, it is sensible to demand permutation invariance and equivariance. Intuitively: the output of any function on a graph should not depend on the order of the nodes (invariance), and if the function provides outputs for each node, then re-ordering the nodes of the input graph should be equivalent to applying the same re-ordering to the output values (equivariance). Formally, let $S(V(G))$ be the set of all permutations of $V(G)$, D a set of graphs, and L a set of potential output attributes (e.g. $\mathbb{R}^3$). Then a function $f : D \to L$ is *invariant* if:

$$\forall \text{ graphs } G \in D, \forall \text{ permutations } \sigma \in S(V(G)), f(G) = f(\sigma \circ G)$$

A function $f : D \to L^{|V(G)|}$ is likewise *equivariant* if instead $f(\sigma \circ G) = \sigma \circ f(G)$.

GNNs belong to a class of neural methods that operate upon graphs. The term is often used to refer to a large variety of models; in this paper, we define the term "Graph Neural Networks" to correspond to the definition of Message Passing Neural Networks (MPNNs) by Gilmer et al. [17], which is the most common GNN architecture. Notable instances of this architecture include Graph Convolutional Networks (GCNs) [12], GraphSAGE [19], and Graph Attention Networks (GATs) [56]. A GNN consists of multiple message-passing layers, where each layer aggregates node attribute information to every node from its neighbours in the graph, and then uses the aggregated information with the current node attribute to assign a new value to the attribute, passing all updated node attributes to the next GNN layer. GNNs are often augmented with global readouts, where in each layer we also aggregate a feature vector for the whole graph and use it in conjunction with the local aggregations [4]. For layer m and node i with current attribute v_i^m, the new attribute v_i^{m+1} is computed as

$$v_i^{m+1} := f_{\text{update}}^{\theta_m}(v_i^m,\ f_{\text{aggr}}^{\theta'_m}(\{v_j^m \mid j \in N(i)\}),\ f_{\text{read}}^{\theta''_m}(\{v_j^m \mid j \in V(G)\}))$$

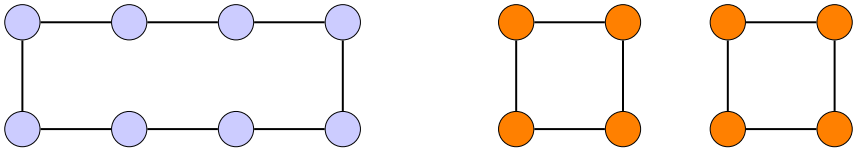
where $N(i)$ is all nodes with edges connecting to i and $\theta_m, \theta'_m, \theta''_m$ are the (possibly trainable) parameters of the update, aggregation, and readout functions for layer m. Parameters may be shared between layers, e.g. $\theta_0 = \theta_1$. The functions $f_{\text{aggr}}^{\theta'_m}, f_{\text{read}}^{\theta''_m}$ are permutation invariant. A global graph feature can be computed by having a final *readout layer* which aggregates all node attributes into a single feature. Importantly, GNNs are invariant / equivariant graph functions.

A.4 Expressivity and Related Work

Morris et al. [34], Xu et al. [61] concurrently showed that any GNN cannot be more powerful than the 1-WL graph-coloring algorithm in terms of distinguishing non-isomorphic graphs, meaning that there are pairs of non-isomorphic graphs G_1, G_2 which for any GNN f, $f(G_1) = f(G_2)$. Morris et al. [34] also define k-GNNs, a class of higher-order GNNs which have the same expressive power as the k-WL algorithm. Chen et al. [8] prove that GNNs cannot count certain types of sub-structures and that certain higher-order GNNs can.

Garg et al. [16] prove that some important graph properties cannot be computed by GNNs, and also provide data dependent generalization bounds for GNNs. Nt and Maehara [38] show that GNNs

Figure 6: A pair of graphs indistinguishable by 1-WL



only perform low-pass filtering on attributes and investigate their resilience to noise in the features. Barceló et al. [4] prove a direct correspondence between GNNs and Boolean classifers expressed in the first-order logic fragment FOC_2.

Loukas [30] demonstrates how GNNs lose expressivity when their depth and width are restricted. Loukas [31] further analyzes the expressive power of GNNs with respect to their *communication capacity*, a measure of how much information the nodes of a network can exchange during message-passing. Oono and Suzuki [40] analyze the expressive power of GNNs as the number of layers tends to infinity, proving that under certain conditions, the output will carry no information other than node degrees and connected components.

In the space of expressivity for reinforcement learning, Dong et al. [11] compare model-free reinforcement learning with the model-based approaches with respect to the expressive power of neural networks for policies and Q-functions. Castellini et al. [6] empirically evaluate the representational power of different value-based RL network architectures using a series of one-shot games. The simplistic games capture many of the issues that arise in the multi-agent setting, such as the lack of an explicit coordination mechanism, which provides good motivation for the inclusion of communication.

A.5 Weisfeiler-Lehman Expressivity

1-WL [60] is a graph coloring algorithm that tests if two graphs are non-isomorphic by iteratively recoloring the nodes. Given an initial graph coloring corresponding to the node labels, in each iteration, two nodes with the same color get assigned different colors if the number of identically colored neighbors is not equal. If, at some point, the number of nodes assigned a specific color is different across the two graphs, the algorithm asserts that the graphs are not isomorphic. However, there are non-isomorphic graphs which the algorithm will not recognize as non-isomorphic, e.g. in Figure 6. Morris et al. [34], Xu et al. [61] proved that for any two non-isomorphic graphs indistinguishable by 1-WL, there is no GNN that can produce different outputs for the two graphs. Furthermore, there is a fundamental link between this graph separation power and function approximation power. Chen et al. [7] proved that a class of models can separate all graphs if and only if it can approximate any continuous invariant function.

There are several GNN architectures which are designed to go beyond 1-WL expressivity. Morris et al. [34] propose k-GNNs, which are equivalent to the k-WL algorithm. Morris et al. [35] also show the link between k-order equivariant graph networks (EGNs) [26] and the k-WL algorithm. Other attempts also include using unique node IDs and passing matrix features [57], relational pooling [36], and random dropouts [42]. Morris et al. [35] provide an overview of many such higher-order models. However, many of these models do not scale well and are computationally infeasible in practice.

A.6 Random Node Initialization

Sato et al. [47] propose augmenting GNNs with *random node initialization* (RNI), where for each node in the input graph, a number of randomly sampled values are concatenated to the original node attribute. For all graphs / nodes, the random values are sampled from the same distribution. Abboud et al. [1] prove that such GNNs are universal and can approximate any permutation invariant graph function. Technically, random initialization breaks the node invariance in GNNs, since the result of the message passing will depend on the structure of the graph as well as the values of the random initializations. However, when one views the model as computing a random variable, the random variable is still invariant when using RNI. In expectation, the mean of random features will be used for GNN predictions, and is the same across each node. However, the variability of the random

samples allow the GNN to discriminate between nodes that have different random initializations, breaking the 1-WL upper bound.

The above is formally described by Abboud et al. [1] as follows. Let G_n be the class of all n-node graphs (i.e. graphs that consist of at most n nodes) and let $f : G_n \to \mathbb{R}$. We say that a randomized function X that associates with every graph $G \in G_n$ a random variable $X(G)$ is an (ϵ, δ)-*approximation* of f if for all $G \in G_n$ it holds that $\Pr(|f(G) - X(G)| \leq \epsilon) \geq 1 - \delta$. Note that an MPNN N with RNI computes such functions X. If X is computed by N, we say that N (ϵ, δ)-*approximates* f.

They state the following theorem:

Theorem. *Let $n \geq 1$ and let $f : G_n \to \mathbb{R}$ be invariant. Then for all $\epsilon, \delta > 0$, there is an MPNN with RNI that* (ϵ, δ)-approximates f.

A.7 Other Conditions for Expressivity

For another method of analyzing GDN expressivity through the lens of GNNs, consider the work of Loukas [31], which defines the *communication capacity / complexity* c_g of GNNs, a measure of how much information the nodes can exchange during message passing. The following intuitive theorem is proven:

Theorem. *Let f be an MPNN with d layers, where each has width w_m (attribute size), message size a_m (output of aggregation), and a global state of size γ_m (output of global readout). For any disjoint partition of V into V_a, V_b, where cut(V_a, V_b) is the size of the smallest cut separating V_a, V_b:*

$$c_g \leq \textit{cut}(V_a, V_b) \sum_{m=1}^{d} \textit{min}(a_m, w_m) + \sum_{m=1}^{d} \gamma_m$$

Loukas [31] prove that MPNNs with sub-quadratic and sub-linear capacity (with respect to the number of nodes) cannot compute the isomorphism class of graphs and trees respectively, demonstrating that capacity is an important consideration for GNN expressivity. This is also supported empirically. When designing a GNN model, we have no control over the structure of the input graphs. Thus, all features apart from cut(V_a, V_b) are important considerations for the architecture. In GDNs, this corresponds to the message sizes m and the number of rounds of message passing d. We provide the practical recommendation that when choosing $\{m, d\}$, one ought to scale them such that $m \cdot d \in \Omega(n^2)$, where n is the number of agents. In environments where communication is limited by range or obstacles, the number of rounds of message passing is also important when it comes to increasing an agent's *receptive field*: from how many edges away information is propagated to the agent.

B Proofs

Before we dive into the proofs, here follows a few brief notes / clarifications on the theory outlined in the paper.

Neural Network Function Approximation GNNs consist of the composition of update, aggregate, and readout functions, all of which are approximated by neural networks. Thus, given that neural networks are only universal approximators for *continuous* functions, if standard neural architectures are used within GNNs, then GNNs can only ever approximate continuous functions. As such, if standard neural networks are used, then all universal approximation results in this paper require the additional assumption that the function being approximated is continuous.

GNN Vector Targeting For all theorems and proofs that utilize $\mathbb{R}$, such as equivariant graph functions with codomain $\mathbb{R}^n$, note that the scalar $\mathbb{R}$ can be replaced with the vector $\mathbb{R}^k$ for any $k > 1$, whilst maintaining correctness.

To demonstrate this, let $k > 1$ and consider an equivariant graph function $f : G_n \to (\mathbb{R}^k)^n$ that we are trying to approximate. We can instead approximate k different functions $f_1, f_2, ..., f_k$ with $f_j : G_n \to \mathbb{R}^n$, such that $\forall G \in G_n \; \forall i \in \{1, 2, ..., n\}, \; f(G)_i = (f_1(G)_i, f_2(G)_i, ..., f_n(G)_i)$. These functions can be approximated using the theoretical results we currently have, with GNNs $g_1, g_2, ..., g_k$. We will simulate the application of all of these GNNs using a single GNN.

Thus, f can also be approximated by the following construction: use an initial GNN layer with the update function defined such that $f_{\text{update}}(v) := (v, v, ..., v)$, where v is transformed into k copies of itself. Then, define the update, aggregation, and readout functions of each layer using the GNNs $g_1, g_2, ..., g_k$, conditioning each one on a portion of the node attributes. For example, the update function of layer m for node i on attribute $(v_1, v_2, ..., v_k)$ is defined by $f^{\theta_m}_{\text{update}}(v_1, v_2, ..., v_k) := (g^{\theta_m}_{1\ \text{update}}(v_1), ..., g^{\theta_m}_{k\ \text{update}}(v_k))$.

This construction simulates the application of $f_1, f_2, ..., f_k$ in parallel and thus approximates f.

Recurrent GNNs In Section 3.1, it is stated that models with recurrent networks also fall within the GDN paradigm, where the hidden or cell states for the networks can be considered as part of the agent observations. In this section, we briefly demonstrate how this can be done.

More concretely, consider a scenario with n agents, where each GNN layer $m \in R \subseteq \{1, 2, ..., M\}$ uses hidden or cell states $C^{m-1} := \{c_1^{m-1}, c_2^{m-1}, ..., c_n^{m-1}\}$. In this case, R represents the list of layers that use recurrent networks. In a non-recurrent GNN, if layer m is expressed as a function, it takes as input only the node attributes from the previous layer: V^{m-1}. In the recurrent case, it also takes C^{m-1} as input: $m(V^{m-1}, C^{m-1})$.

To express this in terms of a non-recurrent GNN, instead modify the initial node attributes $V^0 = \{v_1^0, v_2^0, ..., v_n^0\}$ such that for each node i, $v_i^0 := (v_i^0, c_i^1, c_i^2, ..., c_i^{m'})$, where $m' :=$ max value in R. Then in each layer $m \in \{1, 2, ..., M\}$ of the GNN and for each node i, only update v_i^{m-1} to v_i^m in $(v_i^{m-1}, c_i^1, c_i^2, ..., c_i^{m'})$, leaving all other portions of the attribute as-is. As input to the update, if $m \in R$ then use only v_i^{m-1} and c_i^m, and otherwise use only v_i^{m-1}.

The above-described non-recurrent GNN computes the exact same output as the recurrent GNN it is emulating, by simply pulling hidden or cell states from a portion of the initial node attributes that has been set aside for them.

B.1 Theorem 1

Theorem. *Given a GDN f, observations $O = \{o_1, ..., o_n\}$, and communication graph G such that nodes i and j are similar in G and $o_i = o_j$, then it holds that $f(O)_i = f(O)_j$.*

Proof. Since f is a GDN, there exists a GNN g whose output when operating on the graph G', equal to G augmented with initial node attributes of O, coincides with that of f on O.

Since i and j are similar in G and $o_i = o_j$, i and j are similar in G' (using the same automorphism). Also g is a GNN, so g is an equivariant function on G'. Thus

$$\forall \text{ graphs } G, \forall \text{ permutations } \sigma \in S(V(G)), f(\sigma \circ G) = \sigma \circ f(G)$$

Define $\sigma := (i\ j)$, the permutation that maps $i \to j$ and $j \to i$, while mapping all other nodes to themselves. Note that since i and j are similar, $\sigma \circ G' = G'$. Furthermore, $\sigma = \sigma^{-1}$. So

$$f(O)_i = g(G')_i = (\sigma^{-1} \circ \sigma \circ g(G'))_i = (\sigma^{-1} \circ g(\sigma \circ G'))_i = g(\sigma \circ G')_j = g(G')_j = f(O)_j$$

□

B.2 Theorem 2

Theorem. *Let $n \geq 1$ and let $f : G_n \to \mathbb{R}^n$ be equivariant. Then for all $\epsilon, \delta > 0$, there is a GNN with RNI that* (ϵ, δ)-approximates f.

Proof. For this proof, we assume the reader to be familiar with the proofs in the appendix of Abboud et al. [1], as we make use of their definitions, notation, lemmas, and proofs.

Abboud et al. [1] state and prove the following. Let G_n be the class of all n-node graphs (i.e., graphs that consist of at most n nodes) and let $f : G_n \to \mathbb{R}$. We say that a randomized function X that associates with every graph $G \in G_n$ a random variable $X(G)$ is an (ϵ, δ)-*approximation* of f if for all $G \in G_n$ it holds that $\Pr(|f(G) - X(G)| \leq \epsilon) \geq 1 - \delta$. Note that an MPNN N with RNI computes such functions X. If X is computed by N, we say that N (ϵ, δ)-*approximates* f.

Theorem. *Let $n \geq 1$ and let $f : G_n \to \mathbb{R}$ be invariant. Then for all $\epsilon, \delta > 0$, there is an MPNN with RNI that* (ϵ, δ)-approximates f.

We extend this theorem to equivariant functions. Recall that we define the following. Let G_n be the class of all n-node graphs. Let $f : G_n \to \mathbb{R}^n$, a graph function which outputs a real value for each node in $V(G)$. We say that a randomized function X that associates with every graph $G \in G_n$ a sequence of random variables $X_1(G), X_2(G), ..., X_n(G)$, one for each node, is an (ϵ, δ)-*approximation* of f if for all $G \in G_n$ it holds that $\forall i \in \{1, 2, ..., n\}$, $\Pr(|f(G)_i - X_i(G)| \leq \epsilon) \geq 1 - \delta$, where $f(G)_i$ is the output of $f(G)$ for node i. Note that a GNN h with RNI computes such functions X. If X is computed by h, we say h (ϵ, δ)-*approximates* f.

We adapt the proof of Abboud et al. [1], shown in their appendix, to correspond to equivariant functions instead. Notice that equivariant functions have their output at the node level instead of the graph level, so instead of identifying graphs with C^2-sentences that have no input variables, we identify a graph and a node in the graph by a 1-variable formula $\phi(v)$, where v identifies the node.

Lemma A.3 from Abboud et al. [1] proves that for every individualized colored graph G there is a C^2-sentence χ_G that identifies G. Thus, for every individualized colored graph G and node u, the formula $\phi_{G,u}(v) := \chi_G \wedge \text{Node}_u(v)$ identifies G and the node u, where $\text{Node}_u(v)$ is a Boolean function that is only true when $u = v$. In fact, $\phi_{G,u}(v) := \phi_u(v) := \text{Node}_u(v)$ already identifies G by identifying the exact node.

We can similarly adapt Lemma A.4 and state the following:

Lemma. *Let $h : \mathcal{G}_{n,k} \to \{0, 1\}^n$ be an equivariant Boolean function. Then there exists a 1-variable formula $\phi_h(v)$ such that for all $G \in \mathcal{G}_{n,k}$ and all $v \in G$ it holds that $[[\phi_h(v)]](G) = h(G)_v$.*

To prove this, let $\mathcal{V} \subseteq \{V(G) \mid G \in \mathcal{G}_{n,k}\}$ be the subset consisting of all nodes u with $h(G)_u = 1$, where G is the graph such that $u \in V(G)$. Then let

$$\phi_h(v) := \bigvee_{u \in \mathcal{V}} \phi_u(v)$$

We eliminate duplicates in the disjunction. Since, up to isomorphism, the class $\mathcal{G}_{n,k}$ is finite, and the number of nodes in each graph is upper-bounded by n, the disjunction over $\mathcal{V}$ is finite and hence $\phi_h(v)$ is well-defined.

We adapt Corollary A.1 in the same way as Lemma A.4. Lemma A.5 can be used as-is to show that RNI yields individualized colored graphs with high probability. From here, the remainder of the proof works analogously, substituting in equivariant functions for invariant ones and $\phi_h(v)$ for ψ_h.

□

B.3 Theorem 3

Theorem. *Let $n \geq 1$ and consider a set T, where each $(G, A) \in T$ is a graph-labels pair, such that $G \in G_n$ and there is a multiset of target labels $A_k \in A$ for each orbit $r_k \in R(G)$, with $|A_k| = |r_k|$. Then for all $\epsilon, \delta > 0$ there is a GNN with RNI g which satisfies:*

$$\forall (G, A) \in T \;\; \forall r_k \in R(G), \{g(G)_i \mid i \in r_k\} \cong_{\epsilon,\delta} A_k$$

Proof. Recall that we say two multisets A, B containing random variables are (ϵ, δ)-equal, denoted $A \cong_{\epsilon,\delta} B$, if there exists a bijection $\tau : A \to B$ such that $\forall a \in A,\ \Pr(|a - \tau(a)| \leq \epsilon) \geq 1 - \delta$.

We will define a GNN with RNI g by construction which satisfies the property required in the theorem. We do this in 3 intuitive steps:

1. Define a GNN with RNI f that, for each node, outputs a unique identifier for the orbit of the node and the original RNI value given to the node. Such a GNN exists because the function it is approximating is equivariant.
2. Append n identical layers onto f, each of which identifies the node containing the highest RNI value, gives that node a value from the target multiset of labels corresponding to its orbit, marks off that particular value as claimed, and sets its RNI value to be small.
3. Append a final layer which extracts only the target labels from the node attributes; these were given to the nodes by the preceding n layers.

First, notice that there exists a GNN with RNI $f : G_n \to (\mathbb{R}^4)^n$ that approximates the outputs $(n_i, r_i, 0, 0)$ for each node i in the input graph G, where n_i is the random noise initially added by RNI (before any message passing) and r_i is a unique value corresponding to the graph orbit of i. Formally: $r_i = r_j \iff i$ and j are in the same orbit of the same graph (up to isomorphism). Put another way: $r_i = r_j \iff$ there exists an isomorphism $\alpha : G_i \to G_j$ (the graphs containing nodes i and j, respectively) such that $\alpha(i) = j$.

Such a GNN f exists because the function it is approximating is equivariant, allowing us to use Theorem 2. Without loss of generality, we assume that RNI values are sampled from the interval $(0, 1)$ and that we only augment each node with one RNI value.

We define a GNN with RNI g using f as a starting point: we will append further message-passing layers to f. Append n identical message-passing layers to f, each of which is defined as follows. Each node attribute in these layers will be a tuple (n_i, r_i, c_i, t_i), where c_i is used as a counter and t_i is used to store the eventual node output value, corresponding to some target label. For this proof, we assume that target labels A_k come from $\mathbb{R}$, but note that the proof is easily extended to vectors from $\mathbb{R}$ instead. Furthermore, we allow for A_k to be a multiset (i.e. with repeated elements). Define f_{read} by

$$f_{\text{read}}(\{(n_j, r_j, c_i, t_i) \mid j \in V(G)\}) := \text{argmax}_{(n_j, r_j, c_j, t_j)\ \forall j \in V(G)}\ n_j$$

In other words, f_{read} extracts the tuple containing the maximum value of n_j in the graph. Such a unique maximum exists with probability 1, since finitely many RNI values are sampled from an infinite distribution. Do not define f_{aggr}.

Define f_{update} on the output (n_j, r_j, c_j, t_j) of f_{read} and the current node value (n_i, r_i, c_i, t_i).

$$f_{\text{update}}((n_i, r_i, c_i, t_i), (n_j, r_j, c_j, t_j)) := \begin{cases} (n_i, r_i, c_i, t_i) & \text{if } r_i \neq r_j \\ (n_i, r_i, c_i + 1, t_i) & \text{if } r_i = r_j \text{ and } n_i \neq n_j \\ (0, r_i, c_i + 1, (A_k)_{c_i}) & \text{if } r_i = r_j \text{ and } n_i = n_j \end{cases}$$

In the above, $(A_k)_{c_i}$ denotes treating the multiset of target labels A_k as a sequence and retrieving the element with index c_i (first index is 0). The above has access to A_k since it can uniquely identify the input graph and orbit of the node using r_i, by how r_i is defined.

As a consequence of its definition, the update function will retrieve one value from the target labels at a time, updating exactly one node to store this value. If another node j within the same orbit as i is being updated with this value, then the counter of i is incremented to track that a value from the outputs has been claimed. Since the RNI value n_i is set to 0, it ensures that each node will be given exactly one target label after n rounds of message passing.

After appending the above n message-passing layers, append one final layer with only an update function that extracts only the target labels, defined by

$$f_{\text{update}}((0, r_i, c_i, t_i)) := t_i$$

The above construction of g satisfies the probability (δ) and approximation (ϵ) requirements of the property stated in the theorem, since f is an (ϵ, δ)-approximation, a unique maximum RNI value exists for each graph with probability 1, and all other required operations can be ϵ-approximated by neural networks. The exact bijection used to map between the output and target multisets will depend on the RNI values of the nodes, since they determine the order in which target values are assigned to node attributes in the construction. Whilst our provided construction requires at least $n+1$ message-passing layers, more efficient constructions exist using more complex readout functions than simple maximisation. The final update layer can also be merged with the previous layer. However, we presented the above construction due to its simplicity and how easy it is to understand the mechanism.

□

B.4 Theorem 4

Theorem. *Let $n \geq 1$ and let $f : G_n \to \mathbb{R}^n$ be equivariant. Then for all $\epsilon > 0$, there is a GNN with unique node IDs that* ϵ-approximates f.

Proof. For this proof, we assume the reader to be familiar with the theorems and proofs of Dasoulas et al. [10], particularly their Theorem 4. First, we need to prove that GNNs with unique node IDs are equivalent to 1-CLIP, which is k-CLIP with $k = 1$.

CLIP is defined as a 3-step process, the first of which is assigning colours to nodes. In CLIP, the essential part of each colouring chosen is that all nodes with the same attributes will be assigned different colours. By representing colours with unique node IDs (using a one-hot encoding) we ensure that the above property holds, since all nodes will be assigned different colours. Since we are considering 1-CLIP, we set $k = 1$ and only sample one colouring, which is the particular set of unique IDs we assign.

Step 2 of CLIP is just standard GNN message-passing on our created coloured graph. Step 3 of CLIP maximizes over all possible colourings, of which we have only one, so the maximization can be dropped. This yields a standard GNN global readout layer. Thus, GNNs with unique node IDs are equivalent to 1-CLIP. Theorem 4 of Dasoulas et al. [10] states the universality of 1-CLIP for invariant functions, which we provide here as a Lemma:

Lemma. *The 1-CLIP algorithm with one local iteration is a random representation whose expectation is a universal representation of the space* $\boldsymbol{Graph}_m$ *of graphs with node attributes.*

They further state that for any colouring, 1-CLIP returns an ϵ-approximation of the target function and that, given sufficient training, the variance can be reduced to an arbitrary precision. The above only applies to invariant functions because it is a representation of the space $\mathbf{Graph}_m$, which is defined using invariance by permutation of the labels. Note that when defining $\mathbf{Graph}_m$ in this paper, we use $n_{\text{max}} := n$, instead of just considering some arbitrary large n_{max}.

To apply this theorem to equivariant functions, we need to consider the space $\mathbf{Node}_m$, which we define to be the set of all nodes from graphs in $\mathbf{Graph}_m$. $\mathbf{Node}_m$ is Hausdorff as a trivial consequence of $\mathbf{Graph}_m$ being Hausdorff, using the same quotient space of orbits. If we can separate this space in a continuous and concatenable way, then we can utilize Corollary 1 of Dasoulas et al. [10] to show that it is universal. To do this, consider a GNN f which separates the space $\mathbf{Graph}_m$, which we know exists due to Theorem 4 of Dasoulas et al. [10]. We define a new GNN g using f as a starting point. Substitute the final global readout layer M of f for a new layer with the same readout function, but have the output of the readout be assigned to every node. Formally, define

$$f^{\theta_M}_{\text{update}}(v_i^M,\ f^{\theta'_M}_{\text{aggr}}(\{v_j^M \mid j \in N(i)\}),\ f^{\theta''_M}_{\text{read}}(\{v_j^M \mid j \in V(G)\})) := f^{\theta''_M}_{\text{read}}(\{v_j^M \mid j \in V(G)\}),$$

where $f^{\theta''_M}_{\text{read}}$ is the former global readout layer. Then, change the update functions of each layer of f such that a portion of each node attribute is reserved for the unique ID of the node, and do not use or change this ID in each layer of f. Then, after the final layer M, the attribute v_i^{M+1} of each node i will be

$$v_i^{M+1} = (u_i, r_i) := (u_i, f^{\theta''_M}_{\text{read}}(\{v_j^M \mid j \in V(G)\})),$$

where u_i is the unique ID given to node i. Since f separates $\mathbf{Graph}_m$, r_i uniquely identifies the graph provided in the input. Furthermore, u_i uniquely identifies each node in the input graph. Thus, $v_i^{M+1} = (u_i, r_i)$ uniquely identifies every node in the space $\mathbf{Node}_m$, meaning that g yields a separable representation of $\mathbf{Node}_m$. Furthermore, g is continuous and concatenable by its construction, so we can apply Corollary 1 of Dasoulas et al. [10] to state that g is universal.

□

B.5 Theorem 5

Theorem. *Let $n \geq 1$ and consider a set T, where each $(G, A) \in T$ is a graph-labels pair, such that $G \in G_n$ and there is a multiset of target labels $A_k \in A$ for each orbit $r_k \in R(G)$, with $|A_k| = |r_k|$. Then for all $\epsilon > 0$ there is a GNN with unique node IDs g which satisfies:*

$$\forall (G, A) \in T \;\; \forall r_k \in R(G), \{g(G)_i \mid i \in r_k\} \cong_\epsilon A_k$$

Proof. Recall that two multisets A, B without random variables are ϵ-equal, denoted $A \cong_\epsilon B$, if there exists a bijection $\tau : A \to B$ such that $\forall a \in A, \; |a - \tau(a)| \leq \epsilon$.

The proof for this theorem is by construction, where the construction is nearly identical to the one used in the proof of Theorem 3. A GNN f exists due to Theorem 4 instead of Theorem 2. Unique ID values are used instead of RNI values. One-hot encodings can be maximised over in a similar way, by treating them as binary numbers. A unique maximum unique ID will always exist by definition.

The construction otherwise proceeds analogously, except that it presents an ϵ-approximation of each target multiset instead of an (ϵ, δ)-approximation, since no randomness is used. □

Table 8: Architecture of the Baselines

Name	Communication Graph	MARL Paradigm	GNN Usage
CommNet [51]	Complete (or environment-based)	Recurrent A2C	Implicit
IC3Net [49]	Complete + Gating	Recurrent A2C	Implicit
TarMAC [9]	Complete + Learned Soft Edges	Recurrent A2C	Implicit GAT
T-IC3Net [49, 9]	Gating + Learned Soft Edges	Recurrent A2C	Implicit GAT
MAGIC [37]	Learned	Recurrent A2C	Explicit GAT
DGN [24]	Environment-based	Q-network	Explicit GCN

C Experiments

In this appendix, we provide full details about our experiments for reproducibility.

C.1 Baseline Communication Methods

For evaluation, we adopt a diverse selection of MARL communication methods which fall under the GDN paradigm. These are shown in Table 8, along with the respective paradigm (whether the method simply falls within GDNs or whether GNNs are explicitly used for communication), the MARL paradigm, and communication graph structure. We use the code provided by Niu et al. [37], Jiang et al. [24] as starting points. The code of Jiang et al. [24] uses an MIT license and the code of Niu et al. [37] does not have one. All of the implementations are extended to be able to support multiple rounds of message-passing and the baselines are augmented with the ability for their communication to be masked by the environment (e.g. based on distance or obstacles in the environment).

Sukhbaatar et al. [51] define CommNet, which has a single, basic, learnable communication channel. They define it in such a way that agents can enter and leave the communication range of other agents. It maps directly onto a GDN approach where mean is used for aggregation. Singh et al. [49] define IC3Net, which operates in a similar manner to CommNet, except the communication graph is complete and communication is controlled by gating, meaning each agent can decide whether or not to broadcast to another agent. Das et al. [9] define TarMAC, where a soft attention mechanism is used to decide how much of a message an agent should process. This implicitly yields a complete communication graph, which a graph attention network (GAT) is able to model. TarMAC is also extended to use IC3Net's reward and communication structure, which we refer to as *T-IC3Net*.

Jiang et al. [24] define DGN, which operates on graphs that arise deterministically from the environment (e.g. based on agent proximity). The model consists of an encoding layer from the observations, two convolutional layers that use multi-head dot-product attention as the convolutional kernel, and a shared Q-network between all agents. There are skip connections between the convolutional layers. Note that among our chosen methods, DGN is the only value-based one. Niu et al. [37] define MAGIC, which learns to construct a communication graph and then uses GNNs to operate on the graph. The Scheduler learns which agents should communicate with each other and outputs a communication graph. The Message Processor then uses GATs for multiple rounds of communication on the graph.

C.2 Environments

Predator-Prey [49, 9, 29, 27, 37] and Traffic Junction [51, 49, 9, 29, 27, 37] are common MARL communication benchmarks. We perform evaluations on them to test how well our universally expressive GDN models perform when there is not necessarily a benefit to having communication expressivity beyond 1-WL. We also introduce two new environments, Drone Scatter and Box Pushing, to respectively test symmetry-breaking and communication expressivity beyond 1-WL.

Predator-Prey, introduced by Singh et al. [49], consists of predators (agents) with limited vision trying to find stationary prey. They can communicate with each other within a range of 5 and at each time step move one grid cell in any cardinal direction. An episode is deemed a success if all agents have found and are sitting on top of the prey. We utilize the "cooperative" reward setting of the environment, meaning that reward is given at each time step proportional to the number of agents on the prey. The environment is demonstrated in Figure 7.

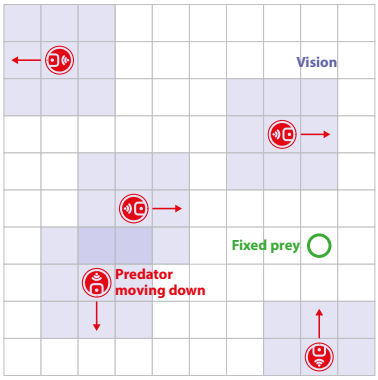


Figure 7: PredatorPrey Environment

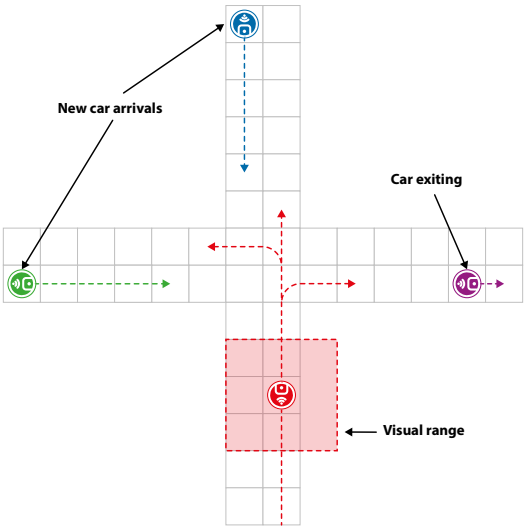


Figure 8: Medium TrafficJunction Environment

Traffic Junction, introduced by Sukhbaatar et al. [51], consists of intersecting roads with cars (agents) driving along them. The agents have limited vision and need to communicate to avoid collisions; an episode is deemed a success if it had no collisions. Each car can communicate with any other car within a range of 3. At each time step, cars enter the environment with a given probability, provided the number of cars in the environment does not exceed the allowed maximum. At each step, a car can either "gas" or "break", leading to it either moving forward one cell on its route or remaining stationary. We utilize both the "Easy" and "Medium" versions of the environment, which respectively consist of two intersecting 1-way roads and two intersecting 2-way roads. The environment is demonstrated in Figure 8.

The **Drone Scatter** environment is a grid environment, which we design to test the ability of communication models to perform symmetry-breaking. It consists of 4 drones in a 20x20 homogeneous field. The outer lines of the field are marked by fences. The drones can move any of the 4 cardinal directions at each time step. Their goal is to move around and find a target hidden in the field, which they can only notice when they get close to. The drones do not have GPS and can only see directly beneath them using their cameras. They also know which action they took in the last time step. The best way for them to locate the target is to split up and search in different portions of the field. Thus, in the "easy" version of the environment, they are encouraged to do this by being given a reward based on how far away they are from the rest of the drones. They are also always given a reward for finding the target. The environment is demonstrated in Figure 9.

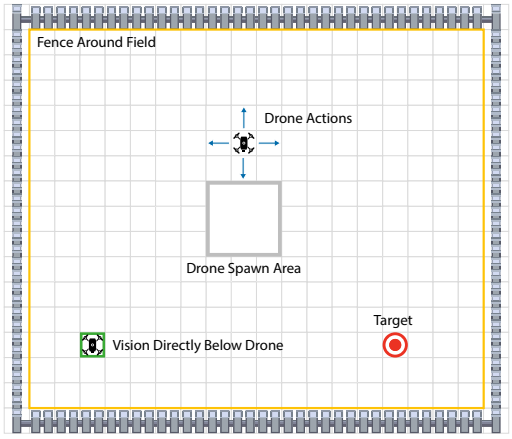


Figure 9: DroneScatter Environment

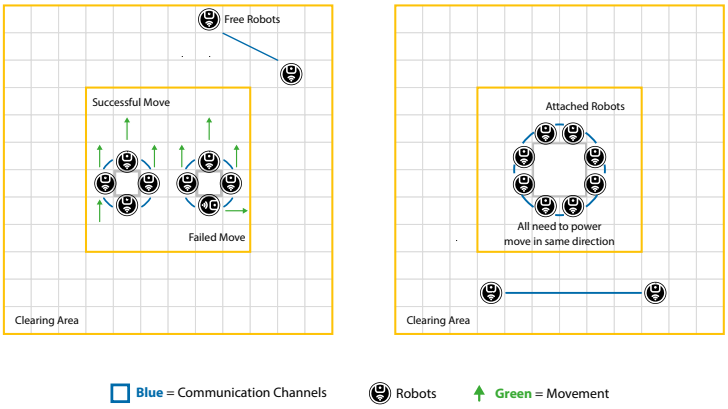


Figure 10: BoxPushing Environment

The **Box Pushing** environment is also a grid environment, which we design to test communication expressivity beyond 1-WL. It has 10 robots in a 12x12 construction site. The out-most 3 grid cells on every side represent the "clearing area", into which robots need to clear the boxes from the central area of the site. Robots can see the cells next to them. Robots attach themselves to boxes before they can move them. When they are attached, robots cannot see around them any more. Free-roaming robots can communicate with any other free-roaming robots, but attached robots can only communicate with the robots directly adjacent to them. The environment either spawns with one large box or two small boxes. 4 attached robots are needed to move a small box. 8 attached robots are needed to move a large box. The agents have 9 possible actions: stay, move in 1 of the 4 cardinal directions, or power move in 1 of the 4 cardinal directions. A small box only moves if all attached agents move in the same direction. A large box only moves if all attached agents power move in the same direction. Robots are penalised for exerting themselves without moving the box, and are rewarded for moving the box closer to the clearing area or clearing the box. Once a box has been cleared, it is removed from the site and the agents are free to continue moving around the environment.

To solve the environment, the robots need to be able to communicate with each other to figure out which type of box they are on and all push correctly, at the same time, and in the same direction. Since the communication graphs corresponding to the scenarios with small and large boxes are 1-WL indistinguishable, communication beyond 1-WL is needed to properly solve the environment. In the "easy" version, robots spawn already attached to the boxes. The environment is demonstrated in Figure 10.

C.3 Hybrid Imitation Learning

We use hybrid imitation learning for all Box Pushing experiments to solve the issue of the exceptionally sparse rewards. Since agents are only given a reward when they all take the same action and thus move the box, random policies will struggle to ever obtain a meaningful reward signal during exploration. When agents are attached to a large box, the probability of this happening at a single time step is $4/9 \times (1/9)^7 = 9.29\text{e}{-8}$, meaning $\approx 10^8$ time steps of experience are needed before any reward can be expected.

Hester et al. [20] propose using expert demonstrations for training and Subramanian et al. [50] propose using some expert demonstrations to help exploration. Inspired by this, during training, we interleave 100 expert experiences for every 500 experiences collected by the agents in the environment. While this is stable for the value-based method DGN, doing so for the A2C methods leads to very unstable performance during training. Wang et al. [58] propose several ways to improve such training of A2C methods, but we choose not to implement them as it is not the focus of this paper.

C.4 Model and Environment Hyperparameters

C.4.1 Fixed Model Hyperparameters

Model hyperparameters that are fixed across all experiments are shown in Table 9, along with which group they belong to, their fixed values, and their descriptions.

C.4.2 TrafficJunction-Easy

Model hyperparameters for the Easy Traffic Junction experiments are shown in Table 10, along with which group they belong to, their values (sometimes a set of values), and their descriptions.

C.4.3 PredatorPrey

Model hyperparameters for the Predator-Prey experiments are shown in Table 11, along with which group they belong to, their values (sometimes a set of values), and their descriptions.

C.4.4 TrafficJunction-Medium

Model hyperparameters for the Medium Traffic Junction experiments are shown in Table 12, along with which group they belong to, their values (sometimes a set of values), and their descriptions.

C.4.5 BoxPushing

Model hyperparameters for the Box Pushing experiments are shown in Table 13, along with which group they belong to, their values (sometimes a set of values), and their descriptions.

C.4.6 DroneScatter-Stochastic

Model hyperparameters for the Drone Scatter experiments with stochastic evaluation are shown in Table 14, along with which group they belong to, their values (sometimes a set of values), and their descriptions.

C.4.7 DroneScatter-Greedy

Model hyperparameters for the Drone Scatter experiments with greedy evaluation are shown in Table 15, along with which group they belong to, their values (sometimes a set of values), and their descriptions.

Table 9: Fixed model parameters for all experiments

Group	Parameter	Value	Description
Training	epoch_size	10	Number of update iterations in an epoch
	batch_size	500	Number of steps before each update (per thread)
	nprocesses	1	How many processes to run
DGN Training	update_interval	5	How many episodes between model update steps
	train_steps	5	How many times to train the model in a training step
	dgn_batch_size	128	Batch size
	epsilon_start	1	Epsilon starting value
	epsilon_min	0.1	Minimum epsilon value
	buffer_capacity	40000	Capacity of the replay buffer
Model	hid_size	128	Hidden layer size
	qk_hid_size	16	Key and query size for soft attention
	value_hid_size	32	Value size for soft attention
	recurrent	True	Make the A2C model recurrent in time
	num_evals	10	Number of evaluation runs for each training iteration
	env_graph	True	Whether the environment masks communication
	comm_passes	4	Number of comm passes per step over the model
Optimization	gamma	1	Discount factor
	normalize_rewards	False	Normalize rewards in each batch
	lrate	0.001	Learning rate
	entr	0	Entropy regularization coefficient
	value_coeff	0.01	Coefficient for value loss term
A2C Models	comm_mode	avg	Mode for communication tensor calculation
	comm_mask_zero	False	Mask all communication
	mean_ratio	1	How much cooperation? 1 means fully cooperative
	rnn_type	MLP	Type of RNN to use [LSTM | MLP]
	detach_gap	10	Detach hidden and cell states for RNNs at this interval
	comm_init	uniform	How to initialise comm weights [uniform | zeros]
	hard_attn	False	Whether to use hard attention: action - talk | silent
	comm_action_one	False	Whether to always talk
	advantages_per_action	False	Whether to multiply action log prob with advantages
	share_weights	True	Share model parameters between agents
MAGIC	directed	True	Whether the communication graph is directed
	self_loop_type	1	Self loop type in the GAT layers (1: with self loop)
	gat_num_heads	4	Number of heads in GAT layers except the last one
	gat_num_heads_out	1	Number of heads in output GAT layer
	gat_hid_size	32	Hidden size of one head in GAT
	message_decoder	True	Whether use the message decoder
	gat_normalize	False	Whether to normalize the GAT coefficients
	ge_num_heads	4	Number of heads in the GAT encoder
	gat_encoder_normalize	False	Normalize the coefficients in the GAT encoder
	use_gat_encoder	False	Whether use the GAT encoder
	gat_encoder_out_size	64	Hidden size of output of the GAT encoder
	graph_complete	False	Whether the communication graph is complete
	learn_different_graphs	False	Learn a new communication graph each round
	message_encoder	False	Whether to use the message encoder

Table 10: Hyperparameters for Easy Traffic Junction experiments

Group	Parameter	Value(s)	Description
Environment	difficulty	easy	Difficulty level [easy | medium | hard]
	dim	6	Dimension of box (i.e length of road)
	env_name	traffic_junction	Environment name
	max_steps	20	Force to end the game after this many steps
	nagents	5	Number of agents
	vision	1	Vision of car
	add_rate_min	0.3	Min probability to add car (till curr. start)
	add_rate_max	0.3	Max rate at which to add car
	curr_start	0	Start making harder after this epoch
	curr_end	0	When to make the game hardest
	vocab_type	bool	Type of location vector to use [bool | scalar]
	comm_range	3	Agent communication range
Model	epsilon_step	$2e{-5}$	Amount to subtract from epsilon each episode
	model	[commnet, tarmac, ic3net, tarmac_ic3net, dgn, magic]	Which baseline model to use
	num_epochs	2000	Number of training epochs
	rni	[0.75, 0.25, 0, 1]	RNI ratio. 0 for none. 1 for unique IDs
	seed	[1, 2, 3, 4, 5]	Random seed

Table 11: Hyperparameters for Predator-Prey experiments

Group	Parameter	Value(s)	Description
Environment	dim	10	Dimension of box (i.e side length)
	env_name	predator_prey	Environment name
	max_steps	40	Force to end the game after this many steps
	mode	cooperative	Reward mode
	nagents	5	Number of agents
	vision	1	Vision of predator
	nenemies	1	Total number of preys in play
	moving_prey	False	Whether prey is fixed or moving
	no_stay	False	Whether predators have an action to stay
	comm_range	5	Agent communication range
Model	epsilon_step	$2e{-5}$	Amount to subtract from epsilon each episode
	model	[commnet, tarmac, ic3net, tarmac_ic3net, dgn, magic]	Which baseline model to use
	num_epochs	2000	Number of training epochs
	rni	[0.75, 0.25, 0, 1]	RNI ratio. 0 for none. 1 for unique IDs
	seed	[1, 2, 3, 4, 5]	Random seed

Table 12: Hyperparameters for Medium Traffic Junction experiments

Group	Parameter	Value(s)	Description
Environment	difficulty	medium	Difficulty level [easy | medium | hard]
	dim	14	Dimension of box (i.e length of road)
	env_name	traffic_junction	Environment name
	max_steps	40	Force to end the game after this many steps
	nagents	10	Number of agents
	vision	1	Vision of car
	add_rate_min	0.3	Min probability to add car (till curr. start)
	add_rate_max	0.3	Max rate at which to add car
	curr_start	0	Start making harder after this epoch
	curr_end	0	When to make the game hardest
	vocab_type	bool	Type of location vector to use [bool | scalar]
	comm_range	3	Agent communication range
Model	epsilon_step	$2e{-5}$	Amount to subtract from epsilon each episode
	model	[commnet, tarmac, ic3net, tarmac_ic3net, dgn, magic]	Which baseline model to use
	num_epochs	2000	Number of training epochs
	rni	[0.75, 0.25, 0, 1]	RNI ratio. 0 for none. 1 for unique IDs
	seed	[1, 2, 3, 4, 5]	Random seed

Table 13: Hyperparameters for Box Pushing experiments

Group	Parameter	Value(s)	Description
Environment	difficulty	easy	Difficulty level. Easy: robots already attached
	dim	12	Dimension of area (i.e. side length)
	env_name	box_pushing	Environment name
	max_steps	20	Force to end the game after this many steps
	nagents	10	Number of agents
	vision	1	Vision of robot
Model	epsilon_step	$2e{-5}$	Amount to subtract from epsilon each episode
	imitation	True	Whether to use hybrid imitation learning
	model	[commnet, tarmac, ic3net, tarmac_ic3net, dgn, magic]	Which baseline model to use
	num_epochs	2000	Number of training epochs
	num_ imitation_ experiences	100	Number of experiences coming from imitation
	num_ normal_ experiences	500	Number of normal policy experiences
	rni	[0.75, 0.25, 0, 1]	RNI ratio. 0 for none. 1 for unique IDs
	seed	[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]	Random seed

Table 14: Hyperparameters for Drone Scatter experiments with stochastic evaluation

Group	Parameter	Value(s)	Description
Environment	difficulty	easy	Difficulty level. Easy: rewarded for splitting
	dim	20	Dimension of field area (i.e. side length)
	env_name	drone_scatter	Environment name
	max_steps	20	Force to end the game after this many steps
	nagents	4	Number of agents
	comm_range	10	Agent communication range
	find_range	3	Agent distance to target to count as find
	min_target_ distance	3	Min distance target can be from spawn area
Model	epsilon_step	$2e{-5}$	Amount to subtract from epsilon each episode
	model	[commnet, tarmac, ic3net, tarmac_ic3net, magic]	Which baseline model to use
	num_epochs	2000	Number of training epochs
	rni	[0.75, 0, 1]	RNI ratio. 0 for none. 1 for unique IDs
	seed	[1, 2, 3, 4, 5]	Random seed

Table 15: Hyperparameters for Drone Scatter experiments with greedy evaluation

Group	Parameter	Value(s)	Description
Environment	difficulty	easy	Difficulty level. Easy: rewarded for splitting
	dim	20	Dimension of field area (i.e. side length)
	env_name	drone_scatter	Environment name
	max_steps	20	Force to end the game after this many steps
	nagents	4	Number of agents
	comm_range	10	Agent communication range
	find_range	3	Agent distance to target to count as find
	min_target_ distance	3	Min distance target can be from spawn area
Model	epsilon_step	$2e{-5}$	Amount to subtract from epsilon each episode
	greedy_a2c_ eval	True	Whether to evaluate A2C methods greedily
	model	[commnet, tarmac, ic3net, tarmac_ic3net, dgn, magic]	Which baseline model to use
	num_epochs	2000	Number of training epochs
	rni	[0.75, 0, 1]	RNI ratio. 0 for none. 1 for unique IDs
	seed	[1, 2, 3, 4, 5]	Random seed

Table 16: Mean and 95% confidence interval for Easy TrafficJunction across all baselines

Baseline	Metric	**Baseline**	**Unique IDs**	**0.75 RNI**	**0.25 RNI**
CommNet	Success	$\mathbf{1 \pm 0}$	$\mathbf{1 \pm 0}$	$\mathbf{1 \pm 0}$	$\mathbf{1 \pm 0}$
	Reward	-1.7 ± 0.01	$\mathbf{-1.69 \pm 0}$	-1.78 ± 0.02	-1.7 ± 0.01
DGN	Success	0.987 ± 0	0.99 ± 0	0.848 ± 0.15	$\mathbf{0.996 \pm 0}$
	Reward	-4.48 ± 0.79	-4 ± 0.17	-9.35 ± 5.57	$\mathbf{-3.99 \pm 0.15}$
IC3Net	Success	$\mathbf{1 \pm 0}$	$\mathbf{1 \pm 0}$	$\mathbf{1 \pm 0}$	0.986 ± 0.02
	Reward	-1.71 ± 0	$\mathbf{-1.7 \pm 0.01}$	-1.74 ± 0.01	-2.02 ± 0.51
MAGIC	Success	0.634 ± 0.11	0.764 ± 0.13	0.684 ± 0.11	$\mathbf{0.787 \pm 0.09}$
	Reward	-16 ± 1.67	-15.8 ± 1.69	-15 ± 2.05	$\mathbf{-14.7 \pm 2.08}$
TarMAC	Success	0.994 ± 0.01	$\mathbf{1 \pm 0}$	0.933 ± 0.04	$\mathbf{1 \pm 0}$
	Reward	-2.07 ± 0.44	$\mathbf{-1.72 \pm 0.02}$	-3.59 ± 1.28	-1.76 ± 0.04
T-IC3Net	Success	$\mathbf{1 \pm 0}$	0.998 ± 0	0.94 ± 0.04	0.974 ± 0.04
	Reward	$\mathbf{-1.74 \pm 0.01}$	-1.79 ± 0.11	-3.16 ± 1.03	-2.25 ± 0.91

Table 17: Mean and 95% confidence interval for PredatorPrey across all baselines

Baseline	Metric	**Baseline**	**Unique IDs**	**0.75 RNI**	**0.25 RNI**
CommNet	Success	0.88 ± 0.03	$\mathbf{0.908 \pm 0.02}$	0.194 ± 0.02	0.476 ± 0.05
	Reward	23.15 ± 0.92	$\mathbf{23.71 \pm 1.18}$	1.828 ± 0.51	10.53 ± 1.92
DGN	Success	0.014 ± 0	0.016 ± 0	0.026 ± 0.03	$\mathbf{0.032 \pm 0.01}$
	Reward	-6.8 ± 0.71	-7.84 ± 0.26	-7.69 ± 0.91	$\mathbf{-4.37 \pm 2.63}$
IC3Net	Success	$\mathbf{0.952 \pm 0}$	0.93 ± 0.02	0.454 ± 0.08	0.933 ± 0.02
	Reward	22.54 ± 1.19	22.99 ± 0.52	10.19 ± 1.47	$\mathbf{24.38 \pm 1.63}$
MAGIC	Success	$\mathbf{0.892 \pm 0.02}$	0.888 ± 0.05	0.112 ± 0.03	0.451 ± 0.09
	Reward	$\mathbf{21.62 \pm 1.31}$	21.36 ± 1.65	-0.85 ± 1.63	9.487 ± 3.07
TarMAC	Success	0.169 ± 0.09	$\mathbf{0.24 \pm 0.11}$	0.068 ± 0.01	0.086 ± 0.02
	Reward	0.323 ± 3.65	$\mathbf{3.131 \pm 3.96}$	-5.22 ± 0.54	-3.14 ± 1.27
T-IC3Net	Success	$\mathbf{0.938 \pm 0.02}$	$\mathbf{0.938 \pm 0.01}$	0.27 ± 0.02	0.913 ± 0.02
	Reward	$\mathbf{23.77 \pm 1.03}$	23.24 ± 0.43	4.725 ± 0.97	22.79 ± 0.46

D Full Results

In this appendix, full results from all experiments are shown. Our experiments were done in parallel on an internal cluster, using only CPUs. With regards to compute time, 127 days were used for Easy Traffic Junction, 149 for Predator-Prey, 186 for Medium Traffic Junction, 502 for Box Pushing, 76 for stochastic Drone Scatter, and 89 for greedy Drone Scatter. This comes to a total of 1129 days.

D.1 Result Tables

In this section, scores for all metrics across all experiments are shown in Table 16 (Easy Traffic Junction), Table 17 (Predator-Prey), Table 18 (Medium Traffic Junction), Table 19 (Box Pushing), Table 20 (Drone Scatter with stochastic evaluation), and Table 21 (Drone Scatter with greedy evaluation).

Table 18: Mean and 95% confidence interval for Medium TrafficJunction across all baselines

Baseline	Metric	**Baseline**	**Unique IDs**	**0.75 RNI**	**0.25 RNI**
CommNet	Success	0.761 ± 0.31	$\mathbf{0.793 \pm 0.33}$	0.046 ± 0	0.614 ± 0.11
	Reward	-50.4 ± 36	-68.9 ± 73.8	-168 ± 7.13	$\mathbf{-48.5 \pm 7.87}$
DGN	Success	$\mathbf{1 \pm 0}$	$\mathbf{1 \pm 0}$	0.062 ± 0	0.619 ± 0.4
	Reward	$\mathbf{-62.7 \pm 0.09}$	$\mathbf{-62.7 \pm 0.1}$	-245 ± 2.61	-138 ± 80.7
IC3Net	Success	$\mathbf{0.971 \pm 0.04}$	0.804 ± 0.1	0.588 ± 0.03	0.855 ± 0.13
	Reward	$\mathbf{-22.6 \pm 1.31}$	-28.1 ± 3.3	-42.2 ± 2.54	-27.1 ± 4.96
MAGIC	Success	0.551 ± 0.28	0.526 ± 0.33	$\mathbf{0.734 \pm 0.21}$	0.4 ± 0.35
	Reward	-132 ± 61.1	$\mathbf{-112 \pm 59.7}$	-173 ± 55.1	-198 ± 60.1
TarMAC	Success	$\mathbf{0.064 \pm 0}$	0.052 ± 0	0.05 ± 0	0.054 ± 0.01
	Reward	-187 ± 24.4	$\mathbf{-182 \pm 28.7}$	-245 ± 2.79	-211 ± 20.7
T-IC3Net	Success	0.89 ± 0.17	0.909 ± 0.08	0.362 ± 0.18	$\mathbf{0.962 \pm 0.02}$
	Reward	-26.4 ± 7.2	-24.9 ± 2.48	-94.2 ± 65.8	$\mathbf{-23.7 \pm 1.1}$

Table 19: Mean and 95% confidence interval for BoxPushing across all baselines

Baseline	Metric	**Baseline**	**Unique IDs**	**0.75 RNI**	**0.25 RNI**
CommNet	Ratio Cleared	0.786 ± 0.08	$\mathbf{0.829 \pm 0.08}$	0.768 ± 0.08	0.795 ± 0.09
	Reward	3777 ± 728	$\mathbf{4439 \pm 687}$	4313 ± 514	4196 ± 615
DGN	Ratio Cleared	0.603 ± 0	0.756 ± 0.06	$\mathbf{0.958 \pm 0}$	0.957 ± 0.01
	Reward	3811 ± 51.5	4127 ± 278	$\mathbf{5536 \pm 57}$	5469 ± 90.2
IC3Net	Ratio Cleared	0.49 ± 0.15	0.617 ± 0.14	0.34 ± 0.18	$\mathbf{0.676 \pm 0.06}$
	Reward	2528 ± 950	2990 ± 864	1341 ± 711	$\mathbf{3306 \pm 556}$
MAGIC	Ratio Cleared	0.958 ± 0.04	0.985 ± 0.01	0.975 ± 0.04	$\mathbf{0.998 \pm 0}$
	Reward	5199 ± 221	5322 ± 214	5444 ± 332	$\mathbf{5464 \pm 118}$
TarMAC	Ratio Cleared	0.629 ± 0.14	0.578 ± 0.11	0.662 ± 0.06	$\mathbf{0.679 \pm 0.06}$
	Reward	3425 ± 820	2961 ± 729	3343 ± 555	$\mathbf{3610 \pm 475}$
T-IC3Net	Ratio Cleared	0.558 ± 0.13	0.596 ± 0.11	0.458 ± 0.18	$\mathbf{0.643 \pm 0.15}$
	Reward	2979 ± 753	$\mathbf{3062 \pm 785}$	1917 ± 763	2908 ± 847

Table 20: Mean and 95% confidence interval for DroneScatter across all baselines except DGN, including a purely random agent. Stochastic evaluation

Baseline	Metric	**Baseline**	**Unique IDs**	**0.75 RNI**
CommNet	Pairwise Distance	11.34 ± 0.9	$\mathbf{12.08 \pm 1.12}$	8.687 ± 1.4
	Steps Taken	11.5 ± 0.26	$\mathbf{9.767 \pm 0.32}$	11.74 ± 1.39
	Reward	269 ± 18.6	$\mathbf{314.2 \pm 22}$	236 ± 45.3
IC3Net	Pairwise Distance	9.108 ± 1.45	$\mathbf{13.3 \pm 0.71}$	10.99 ± 0.38
	Steps Taken	11.94 ± 0.84	$\mathbf{10.13 \pm 0.25}$	11.66 ± 0.22
	Reward	239.2 ± 33.7	$\mathbf{316.7 \pm 11.3}$	260.9 ± 8.62
MAGIC	Pairwise Distance	7.693 ± 1.47	$\mathbf{12.59 \pm 1}$	7.216 ± 0.76
	Steps Taken	13.05 ± 0.58	$\mathbf{11.12 \pm 1.05}$	13.54 ± 0.21
	Reward	205.6 ± 21.4	$\mathbf{273.9 \pm 36.7}$	180 ± 17.3
TarMAC	Pairwise Distance	7.448 ± 0.89	$\mathbf{10.26 \pm 0.69}$	8.486 ± 0.36
	Steps Taken	13.49 ± 0.1	$\mathbf{10.7 \pm 0.35}$	12.85 ± 0.57
	Reward	171.4 ± 13.8	$\mathbf{270.4 \pm 7.81}$	200.2 ± 14.8
T-IC3Net	Pairwise Distance	8.891 ± 0.27	$\mathbf{12.9 \pm 0.78}$	9.552 ± 0.57
	Steps Taken	12.28 ± 0.6	$\mathbf{10.33 \pm 0.46}$	12.22 ± 0.82
	Reward	219 ± 37.2	$\mathbf{309 \pm 12}$	224.9 ± 22.5
Random	Pairwise Distance	5.8 ± 0.02	–	–
	Steps Taken	17.39 ± 0.04	–	–
	Reward	44.59 ± 1.73	–	–

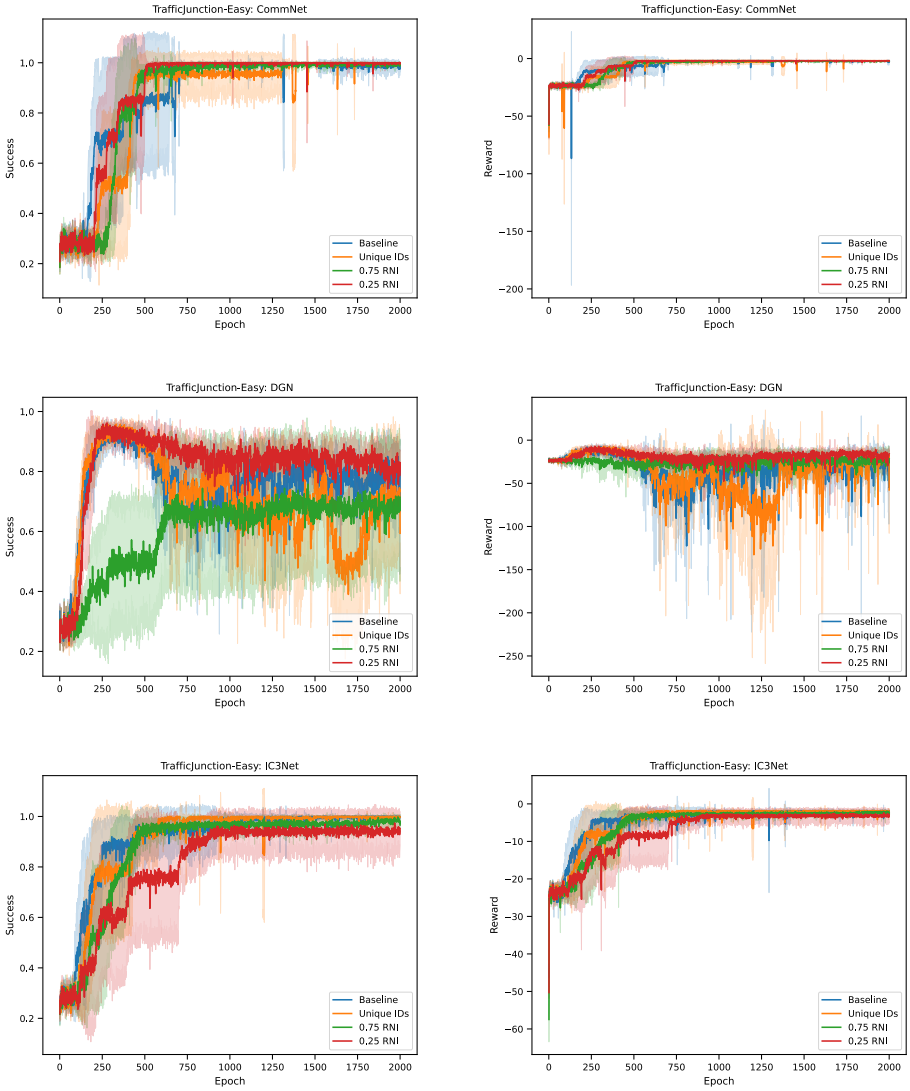
Table 21: Mean and 95% confidence interval for DroneScatter across all baselines. Greedy evaluation

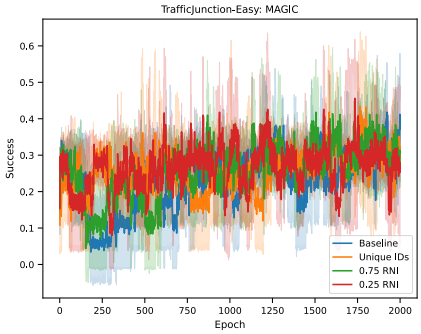
Baseline	Metric	**Baseline**	**Unique IDs**	**0.75 RNI**
CommNet	Pairwise Distance	8.849 ± 0.63	$\mathbf{13.28 \pm 1.27}$	8.589 ± 1.35
	Steps Taken	13.79 ± 0.12	$\mathbf{9.554 \pm 0.33}$	12.62 ± 1.19
	Reward	170.6 ± 6.72	$\mathbf{319.7 \pm 24.7}$	204.8 ± 40.5
DGN	Pairwise Distance	3.221 ± 0.18	$\mathbf{4.427 \pm 0.67}$	3.706 ± 0.83
	Steps Taken	13.36 ± 0.15	$\mathbf{13.27 \pm 0.21}$	13.46 ± 0.14
	Reward	147.9 ± 6.16	$\mathbf{154.4 \pm 6.63}$	149.3 ± 5.38
IC3Net	Pairwise Distance	7.69 ± 1.03	$\mathbf{14.09 \pm 0.54}$	11 ± 0.86
	Steps Taken	13.25 ± 0.4	$\mathbf{10.14 \pm 0.2}$	11.42 ± 0.48
	Reward	186.6 ± 8.87	$\mathbf{310.1 \pm 11}$	264.3 ± 21.9
MAGIC	Pairwise Distance	6.61 ± 1.28	$\mathbf{12.58 \pm 0.6}$	7.107 ± 1.59
	Steps Taken	13.27 ± 0.18	$\mathbf{11.84 \pm 0.68}$	13.61 ± 0.25
	Reward	193.3 ± 15.2	$\mathbf{222.7 \pm 26.6}$	161.3 ± 23.9
TarMAC	Pairwise Distance	8.666 ± 0.28	$\mathbf{12.09 \pm 0.73}$	8.999 ± 0.94
	Steps Taken	13.73 ± 0.21	$\mathbf{11.01 \pm 0.86}$	12.19 ± 0.82
	Reward	139.1 ± 4.43	$\mathbf{255.5 \pm 31.4}$	202.9 ± 32.6
T-IC3Net	Pairwise Distance	7.28 ± 0.69	$\mathbf{13.51 \pm 0.98}$	10.87 ± 1.17
	Steps Taken	13.96 ± 0.26	$\mathbf{10.63 \pm 0.66}$	11.73 ± 0.54
	Reward	156.9 ± 21.9	$\mathbf{278.6 \pm 32.1}$	240.3 ± 25.6

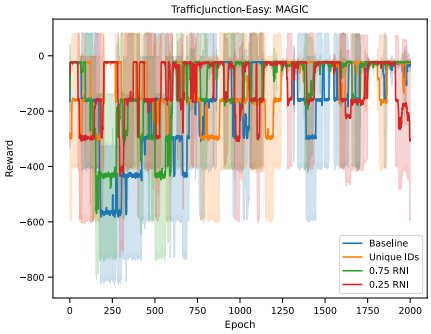
D.2 Result Plots

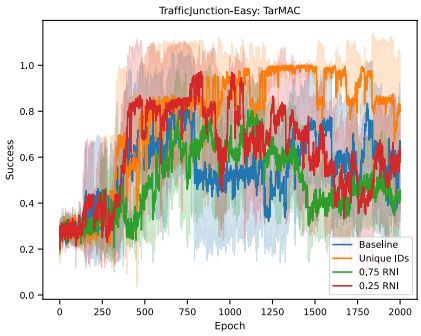
In this section, we provide the full result plots for all of our experiments. For all but the BoxPushing experiments, results are shown aggregated across 5 seeds, with a 95% confidence interval. For the BoxPushing experiments, the hybrid imitation learning paradigm yielded unstable training for all A2C methods. Thus, to better visualize the results, for each seed, we first denote the performance at each epoch to be the maximum performance achieved so far. Then, we aggregate these runs across 10 seeds, showing the mean and a 95% confidence interval.

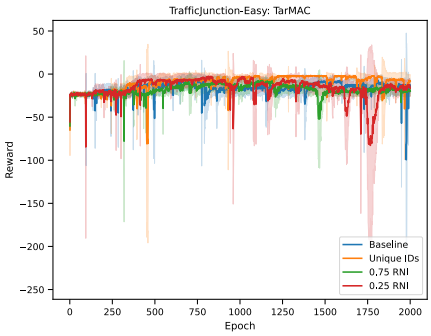
D.3 TrafficJunction-Easy

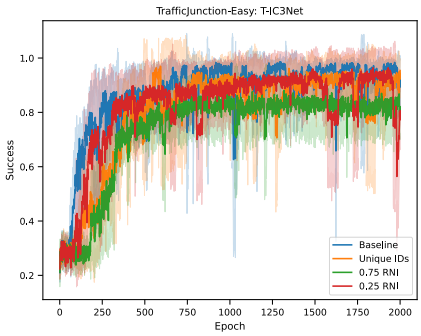


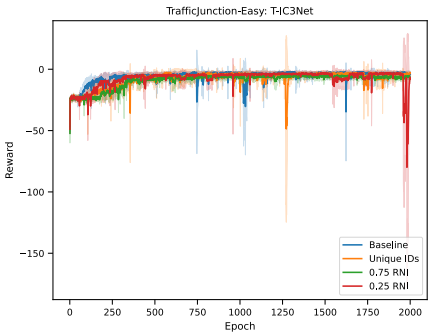

TrafficJunction-Easy: MAGIC
Success
Epoch
Baseline
Unique IDs
0.75 RNI
0.25 RNI


TrafficJunction-Easy: MAGIC
Reward
Epoch
Baseline
Unique IDs
0.75 RNI
0.25 RNI

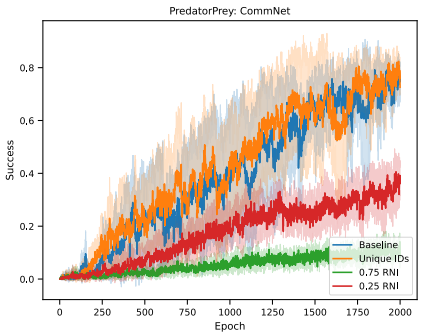

TrafficJunction-Easy: TarMAC
Success
Epoch
Baseline
Unique IDs
0.75 RNI
0.25 RNI

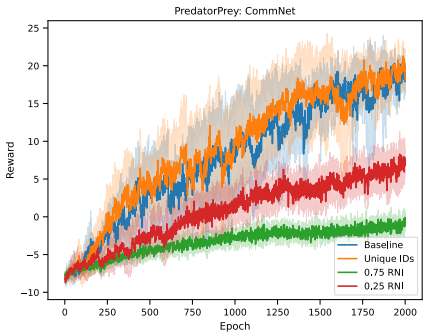

TrafficJunction-Easy: TarMAC
Reward
Epoch
Baseline
Unique IDs
0.75 RNI
0.25 RNI

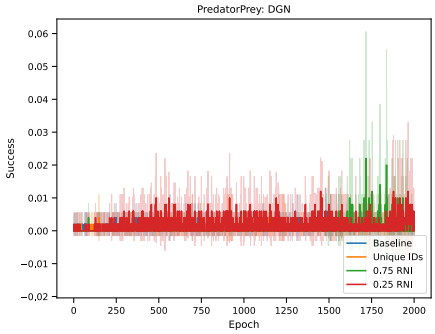

TrafficJunction-Easy: T-IC3Net
Success
Epoch
Baseline
Unique IDs
0.75 RNI
0.25 RNI

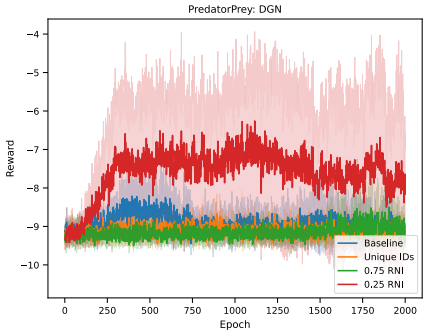

TrafficJunction-Easy: T-IC3Net
Reward
Epoch
Baseline
Unique IDs
0.75 RNI
0.25 RNI

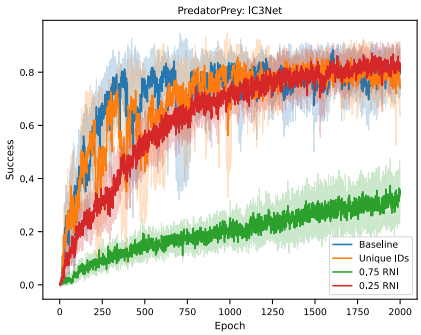
D.4 PredatorPrey

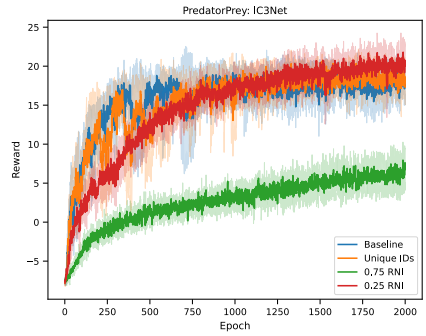


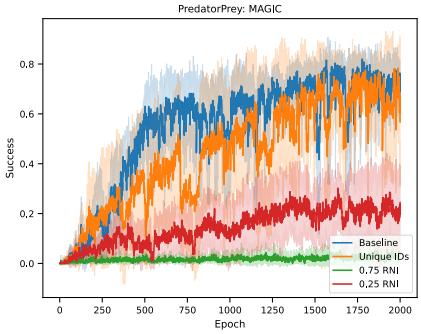


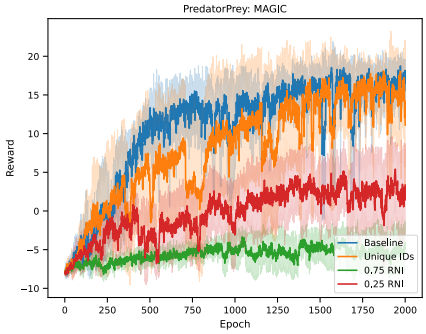


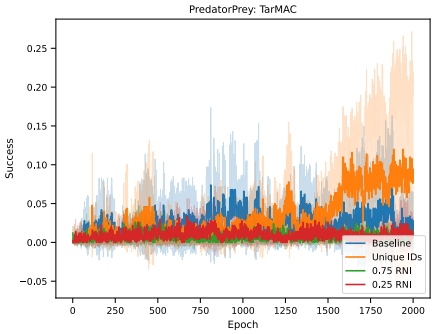


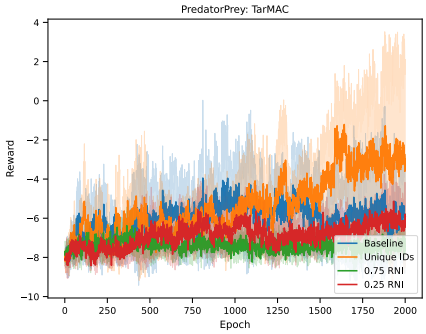


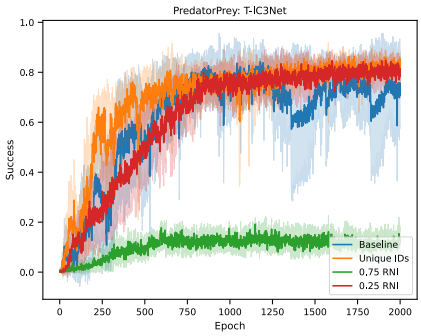


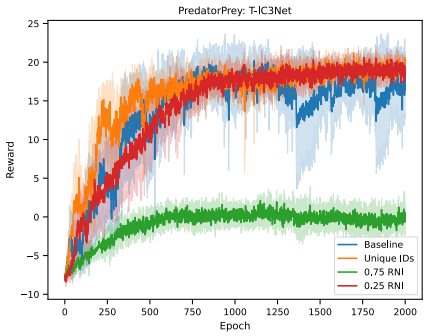




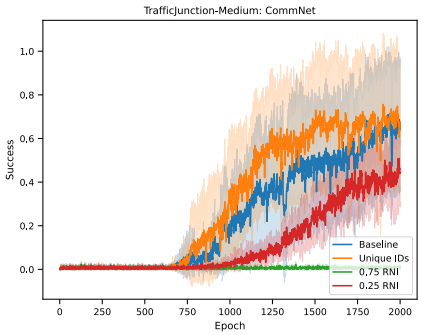


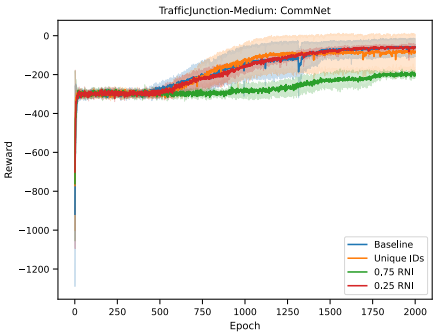


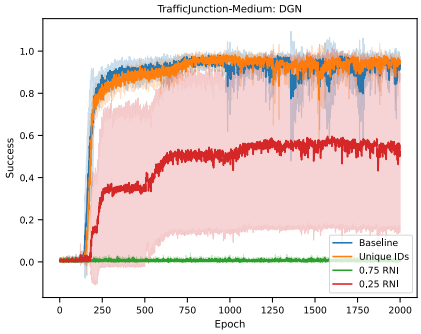


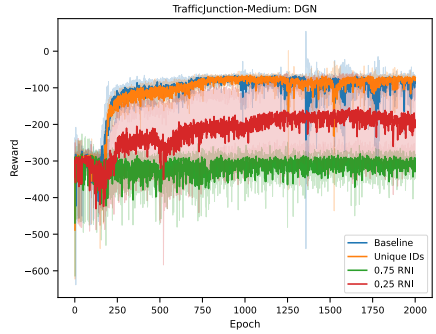


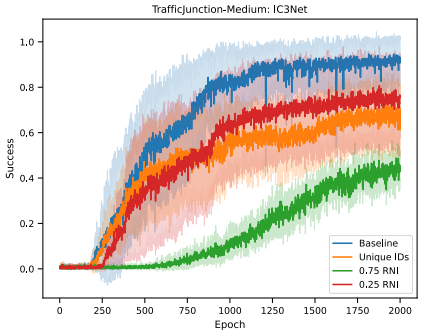
D.5 TrafficJunction-Medium

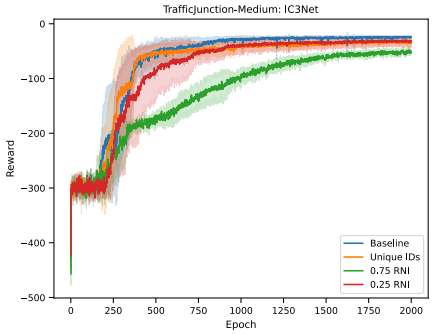


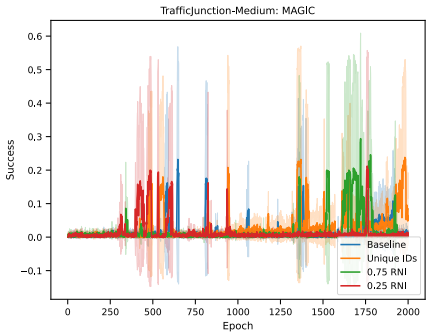


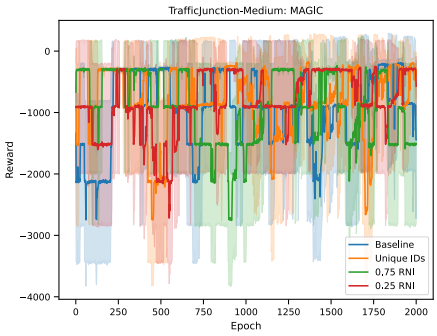


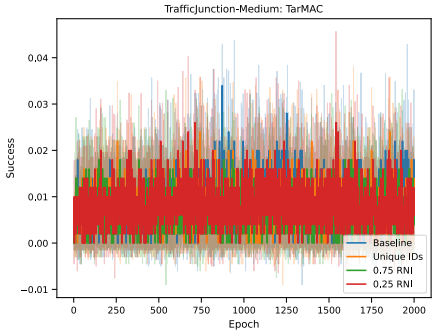


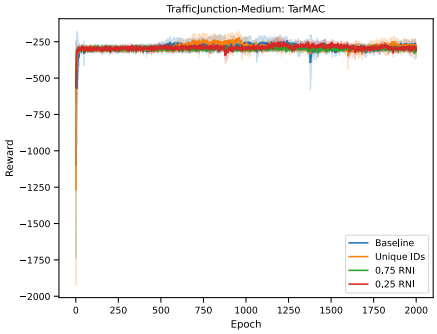

TrafficJunction-Medium: IC3Net
Success
Epoch
Baseline
Unique IDs
0.75 RNI
0.25 RNI

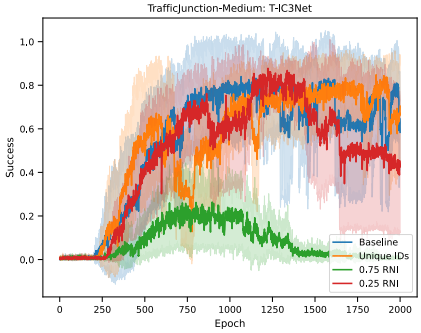

TrafficJunction-Medium: IC3Net
Reward
Epoch
Baseline
Unique IDs
0.75 RNI
0.25 RNI

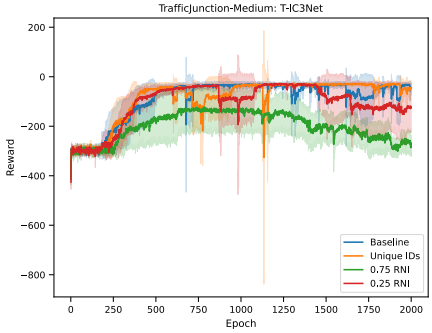

TrafficJunction-Medium: MAGIC
Success
Epoch
Baseline
Unique IDs
0.75 RNI
0.25 RNI


TrafficJunction-Medium: MAGIC
Reward
Epoch
Baseline
Unique IDs
0.75 RNI
0.25 RNI

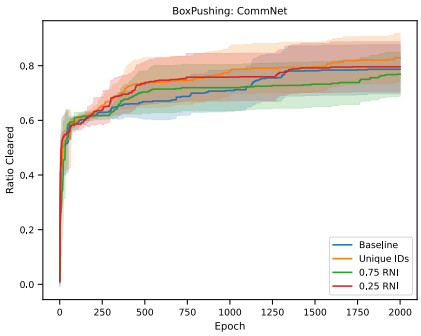

TrafficJunction-Medium: TarMAC
Success
Epoch
Baseline
Unique IDs
0.75 RNI
0.25 RNI

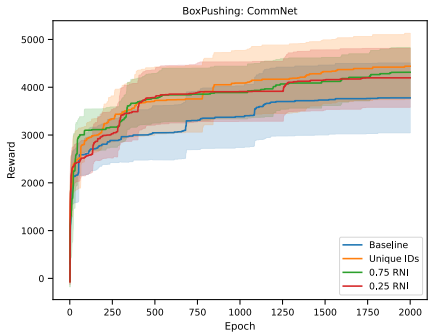

TrafficJunction-Medium: TarMAC
Reward
Epoch
Baseline
Unique IDs
0.75 RNI
0.25 RNI

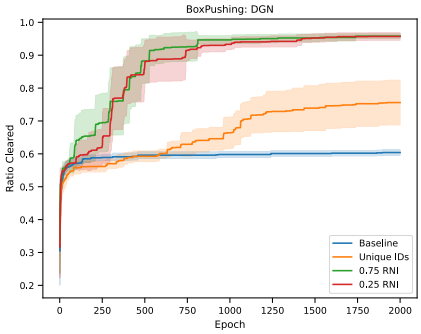

TrafficJunction-Medium: T-IC3Net
Success
Epoch
Baseline
Unique IDs
0.75 RNI
0.25 RNI

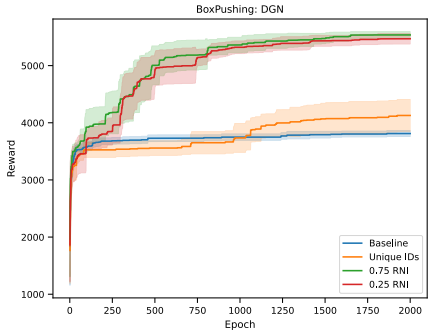

TrafficJunction-Medium: T-IC3Net
Reward
Epoch
Baseline
Unique IDs
0.75 RNI
0.25 RNI

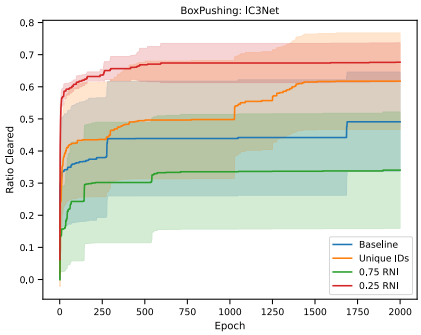
D.6 BoxPushing

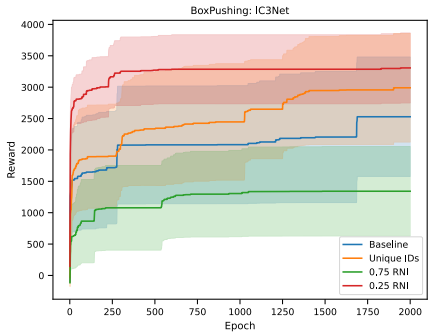


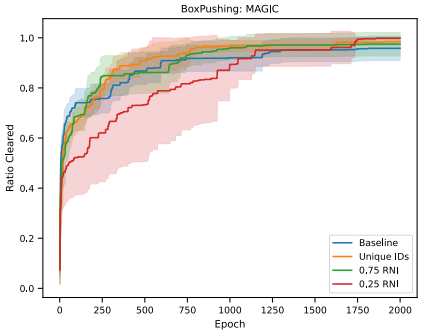


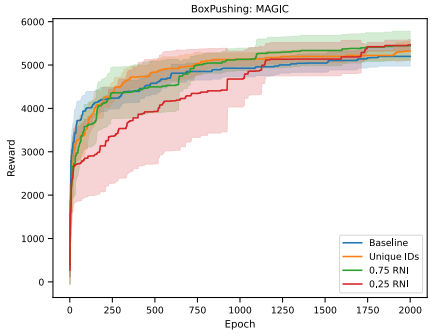


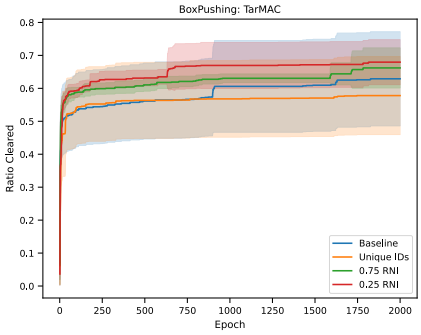


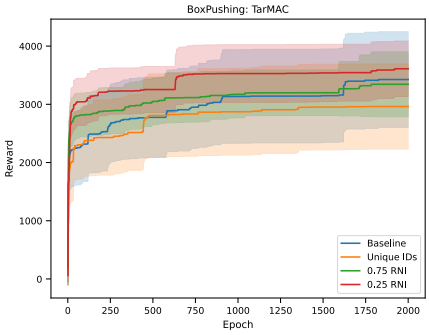


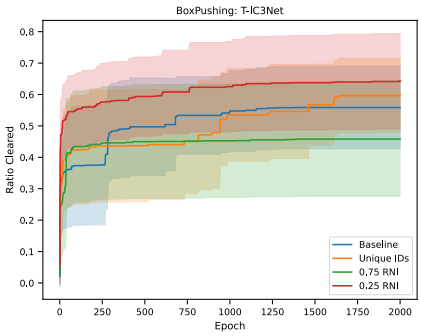


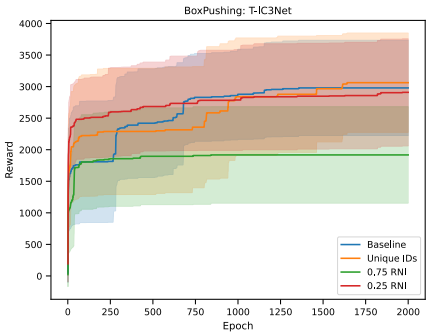




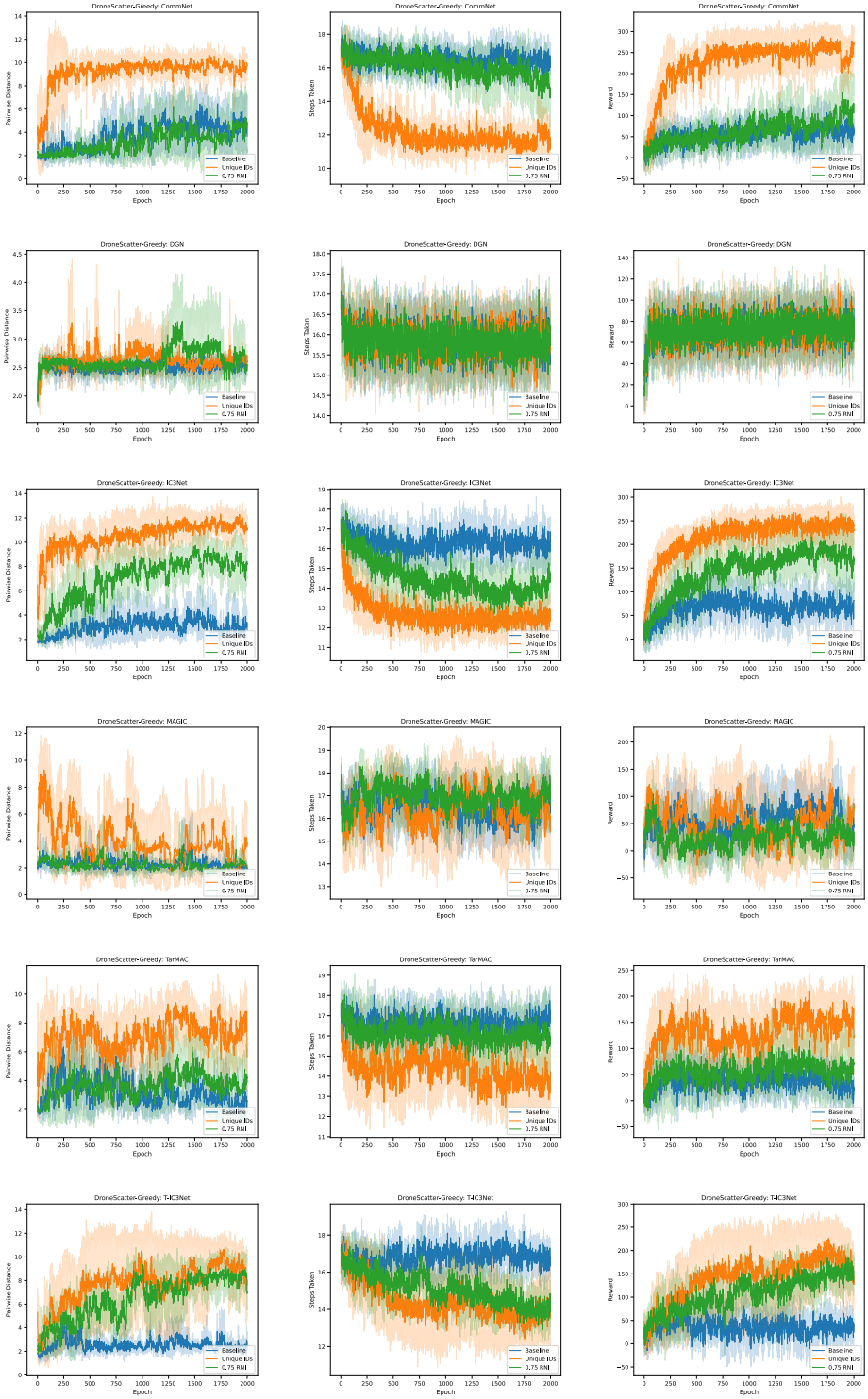

BoxPushing: TarMAC
Ratio Cleared
Epoch
Baseline
Unique IDs
0.75 RNI
0.25 RNI


BoxPushing: TarMAC
Reward
Epoch
Baseline
Unique IDs
0.75 RNI
0.25 RNI


BoxPushing: T-IC3Net
Ratio Cleared
Epoch
Baseline
Unique IDs
0.75 RNI
0.25 RNI


BoxPushing: T-IC3Net
Reward
Epoch
Baseline
Unique IDs
0.75 RNI
0.25 RNI

D.7 DroneScatter with Greedy Evaluation



D.8 DroneScatter with Stochastic Evaluation