

AI-QMIX: Attention and Imagination for Dynamic Multi-Agent Reinforcement Learning

Shariq Iqbal^{*} Christian A. Schroeder de Witt[†] Bei Peng[‡] Wendelin Böhmer[‡]

Shimon Whiteson[†]

Fei Sha^{*‡}

Abstract

Real world multi-agent tasks often involve varying types and quantities of agents and non-agent entities. Agents frequently do not know a priori how many other agents and non-agent entities they will need to interact with in order to complete a given task, requiring agents to generalize across a combinatorial number of task configurations with each potentially requiring different strategies. In this work, we tackle the problem of multi-agent reinforcement learning (MARL) in such dynamic scenarios. We hypothesize that, while the optimal behaviors in these scenarios with varying quantities and types of agents/entities are diverse, they may share common patterns within sub-teams of agents that are combined to form team behavior. As such, we propose a method that can learn these sub-group relationships and how they can be combined, ultimately improving knowledge sharing and generalization across scenarios. This method, Attentive-Imaginative QMIX, extends QMIX for dynamic MARL in two ways: 1) an attention mechanism that enables model sharing across variable sized scenarios and 2) a training objective that improves learning across scenarios with varying combinations of agent/entity types by factoring the value function into imagined sub-scenarios. We validate our approach on both a novel grid-world task as well as a version of the StarCraft Multi-Agent Challenge [28] minimally modified for the dynamic scenario setting. The results in these domains validate the effectiveness of the two new components in generalizing across dynamic configurations of agents and entities.

1 Introduction

Many real-world multi-agent tasks contain scenarios in which an agent must deal with varying numbers and/or types of cooperative agents, antagonist enemies or other entities. For example, consider the game of soccer. Soccer takes many forms, from casual 5 vs. 5 to full scale 11 vs. 11 matches. Players (i.e., agents) may have varying capabilities and specializations, formalized as types such as defenders, forwards, etc., and any game of soccer may contain different combinations of these player types, yielding a rich array of possible strategies. Throughout these scenarios however, the basic task remains the same, and people are able to use their experience in one form of the game to inform their play in another, as different scenarios

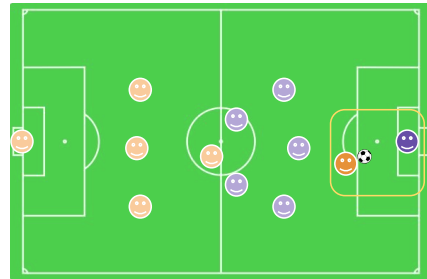


Figure 1: Breakaway sub-scenario in soccer. Agents in the yellow square can ignore the context outside of this region and still predict their success effectively.

^{*}University of Southern California, shariqiq@usc.edu

[†]University of Oxford

[‡]Google AI, fsha@google.com

may share common behavioral patterns among sub-groups of players. For example, in any form of soccer, the situation of a “breakaway” exists, where an attacker with the ball has passed the defense and only needs to beat the goalkeeper in order to score (see Figure 1). In this situation, only the opposing goalkeeper has any significant potential impact on the attacker’s expected success. Thus, an agent who experiences a breakaway in one scenario should be able to share knowledge of that experience across scenarios to improve their performance.

Inspired by these observations, our work focuses on cooperative multi-agent reinforcement learning (MARL) across dynamic scenarios, where there is an opportunity for knowledge sharing to expedite learning. While existing lines of research have explored curriculum learning to scale from small to larger scenarios [1, 21, 36] or collaboration with agents not seen during training [31, 4, 14], to our knowledge, no other works have explored the setting of simultaneous learning across diverse scenarios. In our setting, we define a scenario as the multiset (a set that allows duplicate elements) of agent/entity types present in the environment. Entities are objects that make up the environment, and agents are a subset of these that are controlled by learning policies. The type of an entity affects its dynamics and capabilities within the environment. As such, we classify different team sizes in soccer, as well as team formations involving varying amounts of each player type, as distinct scenarios. Due to the heterogeneity in entity types, optimal strategies can differ greatly across scenarios, even when the basic task is the same; however, training separate policies for each scenario is unfeasible due to the combinatorial number of possible scenarios. Thus, a successful approach requires sharing knowledge across scenarios.

Our contributions are two-fold: To facilitate knowledge sharing we develop a modeling paradigm that is able to process variable numbers of entities such that the same model can learn in multiple scenarios, as well as a technique for exploiting common patterns of behavior among sub-groups of entities. Concretely, to address the first issue we propose to use attention-based architectures [35], and for the second, we propose to factorize value functions into “imagined” sub-scenarios (i.e., subsets of scenarios). By randomly partitioning a given scenario into sub-scenarios and training a value function to represent all of these factorizations, we hypothesize that our model will share knowledge more effectively across scenarios due to presence of common patterns of behavior among sub-groups of entities (e.g., breakaways in soccer).

We note, however, that a sub-scenario common to two scenarios may not play out in the same way in those two scenarios due to interactions with entities outside of the sub-group. As such, our factorization approach must be expressive enough to model these *contextual dependencies*. To this end, we build our approach on QMIX [27], a state-of-the-art cooperative MARL algorithm in the centralized training with decentralized execution paradigm [26, 18]. QMIX factors the global Q -function into a non-linear monotonic mixture of individual agent utility functions, a more expressive factorization than prior work [32]. The flexibility of this factorization approach enables the modeling of contextual dependencies between sub-groups.

We propose Attentive-Imaginative QMIX (AI-QMIX) which extends QMIX to handle variable sized scenarios and introduces “imagined” sub-scenario factorization, attaining improved performance and generalization capabilities on both a complex grid-world environment as well as on challenging StarCraft multi-agent benchmark tasks [28]. We describe the background and necessary modeling techniques in §2. We then introduce our two-fold contributions in §3 and §4 respectively. We describe empirical studies in §5, followed by a discussion of related work (§6), and we conclude in §7.

2 Background and Preliminaries

In this work, we consider the *decentralized partially observable Markov decision process* (Dec-POMDP) [26], which describes fully cooperative multi-agent tasks. Specifically, we build on the setting of Dec-POMDPs with entities [29], introducing the concepts of entity types and scenarios. Our instantiation of a Dec-POMDP with entities can be described as a tuple: $(\mathbf{S}, \mathbf{U}, \mathbf{O}, P, r, \mathcal{E}, \mathcal{A}, \mu, \Phi, \Omega, \rho)$.

\mathcal{E} is the set of entities in the environment. Each entity e has a state representation s^e , and the global state is the set $\mathbf{s} = \{s^e | e \in \mathcal{E}\} \in \mathbf{S}$. Some entities can be agents $a \in \mathcal{A} \subseteq \mathcal{E}$. Non-agent entities are parts of the environment that are not controlled by learning policies (e.g., landmarks, obstacles, agents with fixed behavior). Not all entities may be visible to each agent, so we define a binary observability mask: $\mu(s^a, s^e) \in \{1, 0\}$, where agents can always observe themselves $\mu(s^a, s^a) = 1, \forall a \in \mathcal{A}$. Thus, an agent’s observation is defined as $o^a = \{s^e | \mu(s^a, s^e) = 1\} \in \mathbf{O}$. Each agent a can execute

actions u^a , and the joint action of all agents is denoted as $\mathbf{u} = \{u^a | a \in \mathcal{A}\} \in \mathbf{U}$. Φ is the set of all possible entity types in the environment, where each entity e has a type $\phi^e \in \Phi$.

A scenario is defined as the multiset (a set that allows duplicate elements) of entity types in the environment $\omega = \{\phi^e | e \in \mathcal{E}\} \in \Omega$, and possible scenarios are drawn from a distribution $\rho(\omega)$. As such, scenarios are distinct when they differ in the quantity of each type of entity, but order does not matter. P is the state transition function which defines the probability $P(\mathbf{s}' | \mathbf{s}, \mathbf{u}; \omega)$. $r(\mathbf{s}, \mathbf{u}; \omega)$ is the reward function which maps the global state and joint actions to a single scalar reward given the scenario. Since \mathbf{s} and \mathbf{u} are sets, their ordering does not matter, and our modeling construct should account for this (e.g., by modeling with permutation invariance). The environment dynamics and rewards depend on the scenario, so changing an entity's type affects how the environment behaves.

Reinforcement learning aims to learn a policy (or a set of policies in the case of cooperative MARL) that maximizes expected discounted reward (returns) in some MDP. Q -learning is specifically concerned with learning an accurate action-value function (defined below), and using this function to select the actions that maximize expected returns. The optimal Q -function for the Dec-POMDP setting is defined as:

$$\begin{aligned} Q_\omega^{\text{tot}}(\mathbf{s}, \mathbf{u}) &:= \mathbb{E} \left[\sum_{t=0}^{\infty} \gamma^t r(\mathbf{s}_t, \mathbf{u}_t, \omega) \mid \begin{array}{l} \mathbf{s}_0 = \mathbf{s}, \quad \mathbf{u}_0 = \mathbf{u}, \quad \mathbf{s}_{t+1} \sim P(\cdot | \mathbf{s}_t, \mathbf{u}_t, \omega) \\ \mathbf{u}_{t+1} = \arg \max Q_\omega^{\text{tot}}(\mathbf{s}_{t+1}, \cdot) \end{array} \right] \\ &= r(\mathbf{s}, \mathbf{u}, \omega) + \gamma \mathbb{E} \left[\max Q_\omega^{\text{tot}}(\mathbf{s}', \cdot) \mid \mathbf{s}' \sim P(\cdot | \mathbf{s}, \mathbf{u}, \omega) \right]. \end{aligned}$$

Q_ω^{tot} represents the *total* Q -value, which can be thought of as some combination of latent agent *utilities*. Partial observability is typically handled by using the history of actions and observations as a proxy for state [13]: $Q_\theta^{\text{tot}}(\boldsymbol{\tau}_t, \mathbf{u}_t) \approx \mathbb{E}[Q_\omega^{\text{tot}}(\mathbf{s}_t, \mathbf{u}_t) \mid \omega \sim \rho(\cdot)]$, where the trajectory (i.e., action observation history) is $\tau_t^a := (o_0^a, u_0^a, \dots, o_t^a)$ and $\boldsymbol{\tau}_t := \{\tau_t^a\}_{a \in \mathcal{A}}$.

Work in deep reinforcement learning [23] has popularized the use of neural networks as function approximators for learning Q -functions that are trained by minimizing the following loss function:

$$\mathcal{L}(\theta) := \mathbb{E} \left[\underbrace{\left(r_t + \gamma Q_{\bar{\theta}}^{\text{tot}}(\boldsymbol{\tau}_{t+1}, \arg \max Q_\theta^{\text{tot}}(\boldsymbol{\tau}_{t+1}, \cdot)) - Q_\theta^{\text{tot}}(\boldsymbol{\tau}_t, \mathbf{u}_t) \right)^2}_{y_t^{\text{tot}}} \mid (\boldsymbol{\tau}_t, \mathbf{u}_t, r_t, \boldsymbol{\tau}_{t+1}) \sim \mathcal{D} \right], \quad (1)$$

where $\bar{\theta}$ represents the parameters of a target network that is copied from θ periodically to improve stability [23] and \mathcal{D} is a replay buffer [20] that stores transitions collected by an exploratory policy (typically ϵ -greedy). Double deep Q -learning [34] mitigates overestimation of the learned values by using actions that maximize Q_θ^{tot} as the policy for the target network $Q_{\bar{\theta}}^{\text{tot}}$.

2.1 Value function factorization

Centralized training for decentralized execution (CTDE) has been a major focus in recent efforts in deep multi-agent RL [22, 11, 32, 27, 17]. Some work achieves CTDE by introducing methods for factoring Q -functions into monotonic combinations of per-agent utilities. This factorization allows agents to independently maximize their local utility functions in a decentralized manner with their selected actions combining to form the optimal joint action. This factored representation is less expressive than learning the full joint action value function [5]; however, these methods tend to perform better empirically than those that learn joint action value functions, potentially due to these representations exploiting independence properties among agents [25]. Sunehag et al. [32] introduce Value Decomposition Networks (VDN) which decompose the total Q -value as a sum of per-agent utilities: $Q^{\text{tot}}(\boldsymbol{\tau}_t, \mathbf{u}) := \sum_a Q^a(\tau_t^a, u^a)$.

Building on this work, QMIX [27] relaxes the representational constraints of VDN, by allowing Q^{tot} to be any monotonic function with respect to the agent-specific utilities. This maintains decentralizability while increasing the representational capacity of Q^{tot} . In practice, monotonicity is achieved by using a hypernetwork [12] to generate a mixing network that combines all agents' utilities into Q^{tot} . The hypernetwork generates a set of non-negative weights from the global state \mathbf{s} . We base our work on QMIX, as it is a more expressive factorization than competitive approaches (which we hypothesize is important for our sub-scenario factorization) and reaches state-of-the-art performance on most of the challenging cooperative tasks in the StarCraft Multi-Agent Challenge [28].

2.2 Multi-head attention

Attention models have recently generated intense interest due to their ability to incorporate information across large contexts. Importantly for our purposes, they are able to process variable length inputs. At the core of these models is a parameterized transformation known as multi-head attention [35]. This transformation allows entities to selectively extract information from other entities based on their local context – in our work, this is defined by which other entities are observed.

Our models consist of entity-wise feedforward layers (denoted as $\text{eFF}(\mathbf{X})$) and multi-head attention layers (denoted as $\text{MHA}(\mathcal{S}, \mathbf{X}, \mathbf{M})$), where \mathbf{X} is a matrix with each row corresponding to an entity’s state representation s^e , \mathcal{S} is a set of indices specifying which rows of \mathbf{X} should be used to compute queries, and \mathbf{M} is a binary matrix where each row indicates which entities can be seen by an entity in \mathcal{S} . Entity-wise feedforward layers apply an identical linear transformation to all input entities. Multi-head attention works by using specified entities (in our case all agents) to query all visible entities for information and weighting this information based on their local context. The inclusion of multiple heads allows agents to weight information differently in each head. The process of computing these layers is described in the Supplementary material. Crucially, using these layers we are able to process variable length sets of entities and receive outputs for each agent.

3 Augmenting QMIX with Attention

The standard QMIX algorithm, introduced in Section 2.1, relies on a fixed number of entities in three places: inputs of the agent-specific utility functions Q_a , inputs of the hypernetwork, and the number of utilities entering the mixing network, that is, the output of the hypernetwork. QMIX uses multi-layer perceptrons for which all these quantities have to be of fixed size. In order to adapt QMIX to the dynamic scenario setting, such that we can apply a single model across scenarios, we require components that accept variable sized sets of entities as inputs. By utilizing attention mechanisms, we can design components that are no longer dependent on a fixed number of entities taken as input. We define the following inputs: $\mathbf{X}_{ei}^{\mathcal{E}} := s_i^e$, $1 \leq i \leq d$, $e \in \mathcal{E}$; $\mathbf{M}_{ae}^{\mu} := \mu(s^a, s^e)$, $a \in \mathcal{A}$, $e \in \mathcal{E}$. The matrix $\mathbf{X}^{\mathcal{E}}$ is the global state \mathbf{s} reshaped into a matrix with a row for each entity, and \mathbf{M}^{μ} is a binary observability matrix which enables decentralized execution, determining which entities are visible to each agent. Our proposed architecture is shown in Figure 2.

While the standard agent utility functions map a flat observation, whose size depends on the number of entities in the environment, to a utility for each action, our attention-utility functions can take in a variable sized set of entities and return a utility for each action. The attention layer output for agent a is computed as $\text{MHA}(\{a\}, \mathbf{X}, \mathbf{M}^{\mu})$, where \mathbf{X} is an row-wise transformation of $\mathbf{X}^{\mathcal{E}}$ (e.g., an entity-wise feedforward layer). If agents share parameters, the layer can be computed in parallel for all agents by providing \mathcal{A} instead of $\{a\}$, which we do in practice.

Another challenge in devising a QMIX algorithm for dynamic scenarios is to adapt the hypernetworks that generate weights for the mixing network. Since the mixing network takes in utilities from each agent, we must generate feedforward mixing network parameters that change in size depending on the number of agents present, while incorporating global state information. Conveniently, the number of output vectors of a MHA layer depends on the input set \mathcal{S} and we can therefore generate mixing parameters of the correct size by using $\mathcal{S} = \mathcal{A}$ and concatenating the vectors to form a matrix with one dimension size depending on the number of agents and the other depending on the number of hidden dimensions. The exact architecture of these mixing networks is shown in Fig. 2 and provided in the supplementary material, including details of how all parameters are generated. Attention-based QMIX (A-QMIX) trains these models using the standard DQN loss in Equation 1.

4 Imagined Sub-Scenario Factorization

We now consider the challenge of learning across varied scenarios in environments with heterogeneous agents and present our full method, Attentive-Imaginative QMIX (AI-QMIX). We propose to factorize value functions, not only into agent-wise components as is common in the MARL literature [27, 32], but further into randomized partitions of the scenario for each agent. The use of factored value functions, in general, is motivated by their ability to exploit independence properties in the problem at hand [25], typically between agents. In other words, factored value functions allow for more efficient learning by restricting model capacity in smart ways, e.g., ignoring the effect of joint actions between agents whose actions do not need to be coordinated. We hypothesize that factorizing a scenario into random partitions during training has two beneficial effects: (i) the sub-scenario could have been part of a large set of potential scenarios, thus sub-scenario factorization teaches the agents to ignore the

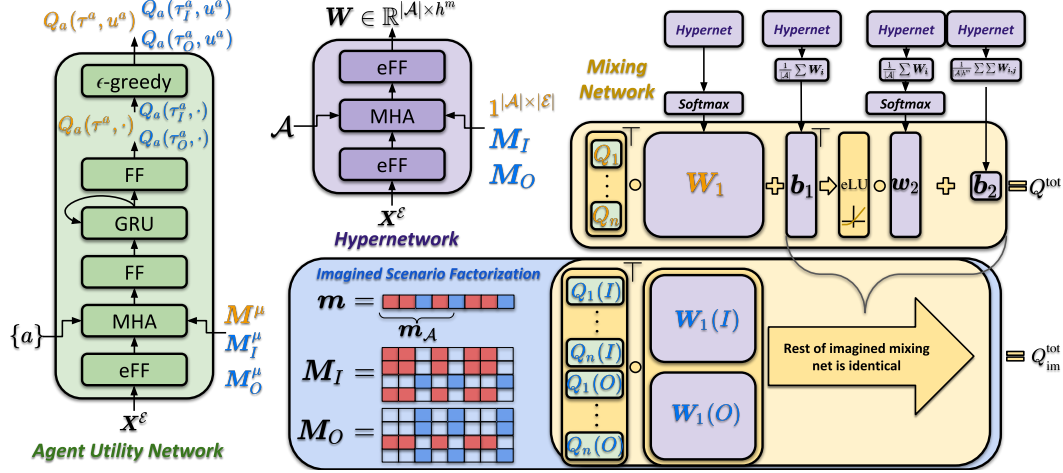


Figure 2: Models for AI-QMIX. Values colored orange or blue are used for computing Q^{tot} and $Q_{\text{im}}^{\text{tot}}$ respectively. **(left)** Agent-specific utility networks. These are decentralizable due to the use of an observability mask (M^μ). We include Gated Recurrent Units [8] to retain information across timesteps in order to handle partial observability. **(top center)** Hypernetworks used to generate weights for the mixing network. Note that the output is always a matrix with a dimension that depends on the number of agents, so we can use this matrix as a weight matrix for the mixing network’s input layer. Mixing parameters that do not depend on the number of agents are computed identically, but averaged across agents to yield the correct shape. The final layer’s bias (a scalar) is averaged across both dimensions. We use a softmax function across the hidden dimension to enforce non-negativity, which we find empirically to be more stable than the standard absolute value function. All hyper networks condition on the full environment state, that is, assume full visibility matrices. **(top right)** The mixing network used to calculate Q^{tot} . **(bottom right)** Procedure for performing imagined sub-scenario factorization. Agents are split into two groups and their utilities are calculated for both within their group, as well as to account for the interactions outside of their group.

context whenever it is helpful to do so; and (ii) it allows transfer of experiences across all consistent scenarios.

Fortunately, the attention-based architecture of A-QMIX proposed in Section 3 lends itself naturally to partitioning scenarios. We can easily *imagine* any sub-scenario of a given scenario by masking out entities. We create sub-scenarios by randomly partitioning all entities in \mathcal{E} into two disjunct groups, indicated by a random binary⁴ vector $\mathbf{m} \in \{0, 1\}^{|\mathcal{E}|}$. The subset of all agents is denoted as $\mathbf{m}_A := [m_a]_{a \in \mathcal{A}}$. We construct new observability masks M_I^μ and M_O^μ as such:

$$M_I^\mu := M^\mu \wedge M_I, M_O^\mu := M^\mu \wedge M_O, \text{ with } M_I := \mathbf{m}_A \mathbf{m}^\top \vee \neg \mathbf{m}_A \neg \mathbf{m}^\top, M_O := \neg M_I. \quad (2)$$

The entry $M_I^\mu[a, e]$ determines both whether agent a can see entity e and whether entity e is in agent a ’s group; whereas, the entry $M_O^\mu[a, e]$ is the same but for entities out of a ’s group. These new observability masks can now be used in the utility networks depicted in Fig. 2 to compute $Q_a(\tau_I^a, u^a)$, which represents the predicted utility of agent a within its group and $Q_a(\tau_O^a, u^a)$, a residual term that accounts for the utility of interactions that a would have with the other group.

Now that we have predicted utility factors for both partitions, we need a way to combine these into an imagined Q -value such that we can use the target from the full scenario (y_t^{tot} in (1)). One could just add them together, as in VDN [32], but we propose to use the dynamic mixing network introduced in Section 3 instead. As this network can mix the utilities of arbitrary numbers of agents, we simply concatenate two generated versions of the input layer (using M_I and M_O) and apply the network to the concatenated utilities $Q_a(\tau_I^a, u^a)$ and $Q_a(\tau_O^a, u^a)$ of all agents a , to compute the imaginary value $Q_{\text{im}}^{\text{tot}}$. This procedure is visualized in Figure 2. Attentive-Imaginative QMIX plugs $Q_{\text{im}}^{\text{tot}}$ in place of Q^{tot} in the DQN loss of (1) to get the imagined loss \mathcal{L}_{im} . Our total loss combines both real and imagined losses: $\mathcal{L} := (1 - \lambda)\mathcal{L}_Q + \lambda\mathcal{L}_{\text{im}}$, where λ is a hyper-parameter. In practice, this procedure requires two additional passes through the network (with M_O^μ and M_I^μ as masks instead of M^μ) per training step. These additional passes can be parallelized by computing all necessary quantities in one batch on GPU. It is feasible to include more than one random partition in any given batch, but we find one to be sufficient in our experiments.

⁴ We first draw $p \in (0, 1)$ uniformly, followed by $|\mathcal{E}|$ independent draws from a Bernoulli(p) distribution.

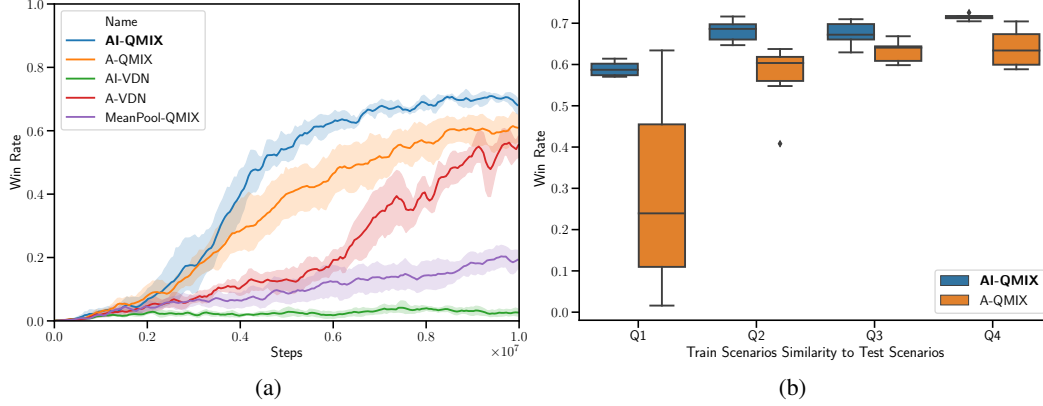


Figure 3: Results on SAVETHECITY environment. (Left) Win rate over time testing on held out set of test scenarios (15% of all scenarios) and training on random selection of 25% of all scenarios. (Right) Mean win rate on test scenarios while varying similarity of train scenarios to test scenarios. Train scenarios are ranked by their mean overlap coefficient with the testing scenarios and separated into quartiles (Q1, Q2, Q3, Q4).

5 Experimental Results

Our experiments aim to evaluate the effectiveness of our two main contributions (attention and imagination) in challenging and dynamic environments. We hypothesize that attention will improve the overall modelling capacity and performance over baseline methods for aggregating information from variable length sets of entities (i.e., pooling). Furthermore, we believe that imagined sub-scenario factorization will improve the knowledge transfer across scenarios, thus improving overall performance as well as generalization to unseen scenarios. We test on two environments, both containing heterogeneous agent types and varying scenario sizes enabling a diverse array of strategies.

5.1 SAVETHECITY

We first test on a novel gridworld resource allocation task, SAVETHECITY, that enables us to carefully control what scenarios we test and train on in order to measure our approach’s improvement in terms of generalization across scenarios. In this task there are 3 distinct types of agents, and their goal is to complete the construction of all buildings on the map while preventing them from burning down. Agents can select actions from: stay in place, move a space in one of the four cardinal directions, put out fire (if neighboring a building on fire), and build (if neighboring a building that is not complete and not on fire). Fires start, with a maximum strength, at a consistent but randomized rate such that agents cannot predict where the next fire will be. When a building is on fire, its health deteriorates at a rate proportional to the strength of the fire until the health reaches zero at which point it cannot be rebuilt. Agents win by completing all buildings without any burning down. Agent types include firefighters (20x speedup over the base rate in reducing fires), builders (20x speedup in building), or generalists (5x speedup in both as well as the ability to move two spaces in one step). Buildings also come in two varieties: fast burning and slow burning, where the fast burning buildings burn four times faster. These diverse agent and building types result in a variety of possible scenarios where agents must decide how to coordinate based on the composition of their team as well as their surroundings. Details on the reward specifications as well as exact rates of burning, building, etc are provided in the supplementary material.

This environment shares similarities with the Factored Firefighting Dec-POMDP benchmark task [25] but includes several key differences which increase the complexity of the challenge. First, the number of entities (agents and buildings) is variable across scenarios. Additionally, agents are permitted to move, so they must not only choose to fight fires at their two adjacent buildings but must also choose from all buildings on the map and physically move to be adjacent to that building. Furthermore, fires not only need to be extinguished but buildings also need to be rebuilt. Finally, agents have different capabilities and buildings burn at different rates, forcing agents to make decisions based on their expectations of what other agents will do and which buildings are most important.

5.2 STARCRAFT

Finally, we test on the StarCraft multi-agent challenge (SMAC) [28]. The tasks in SMAC involve micromanagement of units in order to defeat a set of enemy units in battle. While SMAC serves as a challenging benchmark for MARL, its existing tasks pit fixed teams against one another, enabling

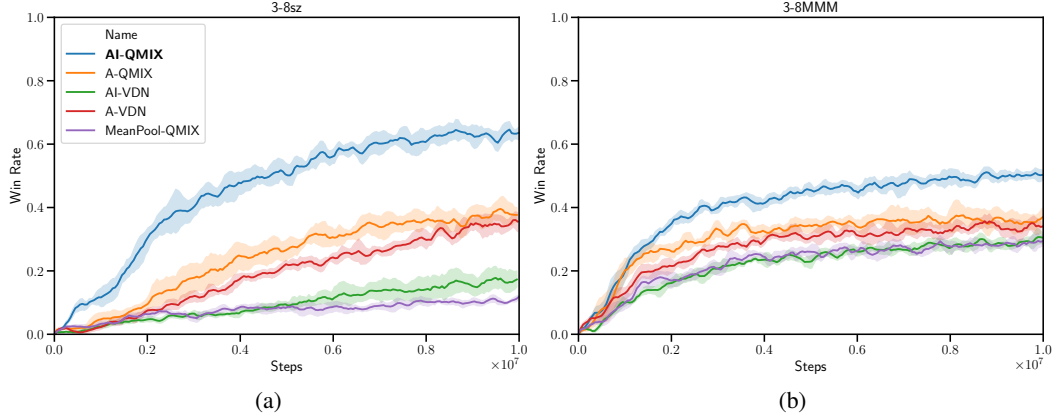


Figure 4: Test win rate over time on 3-8sz (left) and 3-8MMM (right) STARCraft environments.

algorithms to learn fixed strategies. Since we are interested in the dynamic setting where strategies may change depending on team composition, we propose some small modifications to SMAC that change the environment as little as possible while enabling training in dynamic settings. In the standard version of SMAC, both state and action spaces depend on the number of agents and enemies, so our modifications, discussed in detail in the supplementary material, alleviate these problems.

In our tests we evaluate on two settings we call 3-8sz and 3-8MMM. 3-8sz pits symmetrical teams of between 3 and 8 agents against each other where the agents are a combination of Zealots and Stalkers. This setting is designed to provide a continuum of settings encompassing the 2s3z and 3s5z tasks in the original SMAC. 3-8MMM pits symmetrical teams of between 0 and 2 Medics and 3 to 6 Marines/Marauders against each other, which is designed to encompass the MMM tasks from SMAC.

5.3 Baselines

We introduce several baseline adaptations of existing methods to our setting as well as ablations of our method that test the effectiveness of our contributions. A-QMIX is our method without the imagined sub-scenario factoring component. AI-VDN is our method without a non-linear monotonic mixing network. Factors are combined by summation. A-VDN is A-QMIX without the non-linear monotonic mixing network. MeanPool-QMIX is A-QMIX with attention layers replaced by mean pooling. We also test max pooling but find the performance to be marginally worse than mean pooling. Importantly, for pooling layers we add entity-wise linear transformations prior to the pooling operations such that the total number of parameters is comparable to attention layers.

5.4 Results and Discussion

Our results in SAVETHECITY can be found in Figure 3 and results in STARCraft can be found in Figure 4. All algorithms are tested with six random seeds in each setting, and error bands are drawn with a 95% confidence interval. In SAVETHECITY we aim to test the limits of the generalization capabilities of our approach. We first train on a subset of 25% of all possible scenarios and test on a separate testing set (15% of all scenarios), in order to test how well our approach can generalize to unseen scenarios when trained on a small subset of the possible scenarios. Results can be seen in Figure 3a. AI-QMIX consistently outperforms the strongest baseline, A-QMIX, in this setting.

To further evaluate AI-QMIX’s generalization across scenarios, we vary the similarity of the training scenarios to the testing scenarios and measure performance. We rank all possible training scenarios (the 85% not in the testing set) by their mean overlap coefficient with the test scenarios, computed as $\text{overlap}(X, Y) = \frac{|X \cap Y|}{\min(|X|, |Y|)}$. We break these ranked scenarios into quartiles, train on each quartile separately, and evaluate the trained models on the same test set. As seen in Figure 3b, AI-QMIX is far more robust to training scenarios that are dissimilar to the testing scenarios than the strongest baseline, while also taking better advantage when the training scenarios are similar. This indicates that imagined sub-scenario factorization is effective in knowledge transfer across scenarios.

In STARCraft, we do not split scenarios into train and test sets since agents can die during episodes, meaning that scenarios are not fixed during an episode and can overlap despite starting with different sets of units. Instead we test and train on all scenarios with the expectation that methods that generalize better across scenarios will learn faster and perform better, since there are many possible

scenarios. We find that AI-QMIX outperforms all baselines on both settings in a manner consistent with our results in SAVETHECITY. AI-VDN performs much worse than our approach as well as A-VDN, highlighting the importance of the mixing network to handle contextual dependencies between scenario partitions. Since a sub-scenario can play out differently based on the surrounding context, it’s important for our factorization approach to recognize and adjust for these situations. Our mixing network handles these dependencies by a) incorporating global state information into the mixing procedure, and b) mixing utilities in a non-linear monotonic fashion, rather than summing as in VDN. As such, the increased representative capacity of the QMIX mixing network, relative to VDN, is crucial. The use of mean-pooling in place of attention also performs poorly, indicating that attention is valuable for aggregating information from variable length sets of entities.

6 Related Work

Multi-agent reinforcement learning (MARL) is a broad field encompassing cooperative [11, 27, 32], competitive [3, 19], and mixed [22, 17] settings. This paper focuses on cooperative MARL with centralized training and decentralized execution. Several recent works have addressed the topic of generalization and transfer across multi-agent scenarios. Carion et al. [7] devise an approach for assigning agents to tasks, assuming the existence of low-level controllers to carry out the tasks, and show that it can scale to much larger scenarios than those seen in training. While improving generalization to larger tasks, this method assumes the existence of low-level action policies that can execute pre-defined sub-tasks; by contrast, we aim to learn low-level action policies from experience with no pre-defined sub-tasks. Our work considers environments with only low-level actions, where we must learn policies using these actions. Burden [6] propose a transfer learning approach using convolutional neural networks and grid-based state representations to scale to scenarios of arbitrary size. Agarwal et al. [1] propose a graph-neural-network-based model for MARL that is invariant to the number of agents as well as their permutations. Their approach allows communication between agents, and uses a hand-designed curriculum to train on increasingly difficult scenarios. The approach is tested in simple particle environments [24, 22] with homogenous agents. Long et al. [21] introduce an attention-based model for MARL as well as an evolution-based method for automated curriculum learning to enable scaling up to larger scenarios. This approach is also tested on particle environments. Finally, Wang et al. [36] propose using graph neural network models to enable the sharing of experience across scenarios. They test several methods of curriculum learning in order to improve performance on large-scale StarCraft environments with homogeneous agents.

While several of these works introduce attention or graph neural network based models for MARL in dynamic scenarios, they typically focus on curriculum learning across a limited number of similar scenarios, often with homogenous agents. These curriculum learning techniques generally learn in smaller scenarios in order to initialize training in larger scenarios and do not require policies to simultaneously perform well on several diverse scenarios. By contrast, our work focuses on generalization across a variety of scenarios with heterogeneous agent types, and we test on strategically complex environments. This setting requires policies to perform well across various scenarios, potentially including those that the agents have never seen.

Research on the topic of ad hoc teamwork [31, 4] is closely related to our setting, though with some key differences. Similar to our work, ad hoc teamwork aims to obtain policies that act with teams of agents that are not fixed. More specifically, ad hoc teamwork agents are placed in a team of agents with whom they have never coordinated or trained with before and are tasked with adapting in order to maximize performance on cooperative tasks. Along similar lines, Hu et al. [14] introduce the setting of zero-shot coordination where agents are placed in novel teams and are expected to coordinate *without* any adaptation. In contrast to these works, our setting does not require policies to coordinate with agents they have not seen during training. Instead, we are interested in training policies to control dynamic teams composed of varying agent types.

7 Conclusion

In this paper we consider the setting of dynamic multi-agent reinforcement learning, where we aim to learn policies to control teams of agents in scenarios with varying types of entities. We propose AI-QMIX, an approach that incorporates attention models to handle dynamic sized inputs and imagined sub-scenario factoring to promote generalization and knowledge transfer across scenarios. Our results show that both contributions yield performance improvements in complex cooperative tasks. In future work, we hope to explore more principled ways to partition scenarios (rather than randomized

splits) as well as finding ways to generalize to scenarios containing unseen entity types, potentially by incorporating ideas from the ad hoc teamwork literature.

References

- [1] Akshat Agarwal, Sumit Kumar, and Katia Sycara. Learning transferable cooperative behavior in multi-agent teams. *arXiv preprint arXiv:1906.01202*, 2019.
- [2] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.
- [3] Trapit Bansal, Jakub Pachocki, Szymon Sidor, Ilya Sutskever, and Igor Mordatch. Emergent complexity via multi-agent competition. In *International Conference on Learning Representations*, 2018.
- [4] Samuel Barrett, Peter Stone, and Sarit Kraus. Empirical evaluation of ad hoc teamwork in the pursuit domain. In *AAMAS*, pages 567–574, 2011.
- [5] Wendelin Böhrer, Vitaly Kurin, and Shimon Whiteson. Deep coordination graphs. *CoRR*, abs/1910.00091, 2019. URL <http://arxiv.org/abs/1910.00091>.
- [6] Nicholas Burden. Deep Multi-Agent Reinforcement Learning in Starcraft II. Master’s thesis, University of Oxford, 2020.
- [7] Nicolas Carion, Nicolas Usunier, Gabriel Synnaeve, and Alessandro Lazaric. A structured prediction approach for generalization in cooperative multi-agent reinforcement learning. In *Advances in Neural Information Processing Systems*, pages 8128–8138, 2019.
- [8] Junyoung Chung, Caglar Gulcehre, Kyunghyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. In *NIPS 2014 Workshop on Deep Learning, December 2014*, 2014.
- [9] Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. Fast and accurate deep network learning by exponential linear units (elus). *arXiv preprint arXiv:1511.07289*, 2015.
- [10] Karl Cobbe, Oleg Klimov, Chris Hesse, Taehoon Kim, and John Schulman. Quantifying generalization in reinforcement learning. In *International Conference on Machine Learning*, pages 1282–1289, 2019.
- [11] Jakob Foerster, Gregory Farquhar, Triantafyllos Afouras, Nantas Nardelli, and Shimon Whiteson. Counterfactual multi-agent policy gradients. In *AAAI Conference on Artificial Intelligence*, 2018.
- [12] David Ha, Andrew M Dai, and Quoc V Le. Hypernetworks. In *International Conference on Learning Representations*, 2017.
- [13] Matthew J. Hausknecht and Peter Stone. Deep recurrent q-learning for partially observable mdps. In *2015 AAAI Fall Symposia*, pages 29–37, 2015. URL <http://www.aaai.org/ocs/index.php/FSS/FSS15/paper/view/11673>.
- [14] Hengyuan Hu, Adam Lerer, Alex Peysakhovich, and Jakob Foerster. "other-play" for zero-shot coordination. *arXiv preprint arXiv:2003.02979*, 2020.
- [15] Maximilian Igl, Kamil Ciosek, Yingzhen Li, Sebastian Tschiatschek, Cheng Zhang, Sam Devlin, and Katja Hofmann. Generalization in reinforcement learning with selective noise injection and information bottleneck. In *Advances in Neural Information Processing Systems*, pages 13956–13968, 2019.
- [16] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning*, pages 448–456, 2015.

- [17] Shariq Iqbal and Fei Sha. Actor-attention-critic for multi-agent reinforcement learning. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 2961–2970, Long Beach, California, USA, 09–15 Jun 2019. PMLR. URL <http://proceedings.mlr.press/v97/iqbal19a.html>.
- [18] Landon Kraemer and Bikramjit Banerjee. Multi-agent reinforcement learning as a rehearsal for decentralized planning. *Neurocomputing*, 190:82–94, 2016.
- [19] Marc Lanctot, Vinicius Zambaldi, Audrunas Gruslys, Angeliki Lazaridou, Karl Tuyls, Julien Perolat, David Silver, and Thore Graepel. A unified Game-Theoretic approach to multiagent reinforcement learning. In I Guyon, U V Luxburg, S Bengio, H Wallach, R Fergus, S Vishwanathan, and R Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 4193–4206. Curran Associates, Inc., 2017.
- [20] Long-Ji Lin. Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine Learning*, 8(3):293–321, 1992.
- [21] Qian Long, Zihan Zhou, Abhinav Gupta, Fei Fang, Yi Wu, and Xiaolong Wang. Evolutionary population curriculum for scaling multi-agent reinforcement learning. In *International Conference on Learning Representations*, 2020. URL <https://openreview.net/forum?id=SJxbHkrKDH>.
- [22] 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 6382–6393, 2017.
- [23] 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, 2015.
- [24] Igor Mordatch and Pieter Abbeel. Emergence of grounded compositional language in multi-agent populations. In *AAAI Conference on Artificial Intelligence*, 2018.
- [25] Frans A Oliehoek, Matthijs TJ Spaan, Nikos Vlassis, and Shimon Whiteson. Exploiting locality of interaction in factored dec-pomdps. In *Int. Joint Conf. on Autonomous Agents and Multi-Agent Systems*, pages 517–524, 2008.
- [26] Frans A Oliehoek, Christopher Amato, et al. *A concise introduction to decentralized POMDPs*, volume 1. Springer, 2016.
- [27] Tabish Rashid, Mikayel Samvelyan, Christian Schroeder, Gregory Farquhar, Jakob Foerster, and Shimon Whiteson. QMIX: Monotonic value function factorisation for deep multi-agent reinforcement learning. In *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 4295–4304, Stockholm Småttan, Stockholm Sweden, 10–15 Jul 2018.
- [28] Mikayel Samvelyan, Tabish Rashid, Christian Schroeder de Witt, Gregory Farquhar, Nantas Nardelli, Tim GJ Rudner, Chia-Man Hung, Philip HS Torr, Jakob Foerster, and Shimon Whiteson. The starcraft multi-agent challenge. In *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems*, pages 2186–2188. International Foundation for Autonomous Agents and Multiagent Systems, 2019.
- [29] Christian Schroeder de Witt, Jakob Foerster, Gregory Farquhar, Philip Torr, Wendelin Boehmer, and Shimon Whiteson. Multi-agent common knowledge reinforcement learning. In *Advances in Neural Information Processing Systems 32*, pages 9927–9939. Curran Associates, Inc., 2019. URL <http://papers.nips.cc/paper/9184-multi-agent-common-knowledge-reinforcement-learning.pdf>.
- [30] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.

- [31] Peter Stone, Gal A Kaminka, Sarit Kraus, and Jeffrey S Rosenschein. Ad hoc autonomous agent teams: Collaboration without pre-coordination. In *Twenty-Fourth AAAI Conference on Artificial Intelligence*, 2010.
- [32] Peter Sunehag, Guy Lever, Audrunas Gruslys, Wojciech Marian Czarnecki, Vinicius Zambaldi, Max Jaderberg, Marc Lanctot, Nicolas Sonnerat, Joel Z. Leibo, Karl Tuyls, and Thore Graepel. Value-decomposition networks for cooperative multi-agent learning based on team reward. In *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems, AAMAS '18*, pages 2085–2087, Richland, SC, 2018. International Foundation for Autonomous Agents and Multiagent Systems.
- [33] Tijmen Tieleman and Geoffrey Hinton. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning*, 4(2): 26–31, 2012.
- [34] Hado van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. In *Proceedings of the 13th AAAI Conference on Artificial Intelligence*, pages 2094–2100, 2016. URL <https://arxiv.org/pdf/1509.06461.pdf>.
- [35] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, pages 6000–6010, 2017.
- [36] Weixun Wang, Tianpei Yang, Yong Liu, Jianye Hao, Xiaotian Hao, Yujing Hu, Yingfeng Chen, Changjie Fan, and Yang Gao. From few to more: Large-scale dynamic multiagent curriculum learning. In *AAAI Conference on Artificial Intelligence*, 2020.

A Details on Attention Model Layers

We now define the building blocks of our attention models. Given the input \mathbf{X} , a matrix where the rows correspond to entities, we define an entity-wise feedforward layer as a standard fully connected layer that operates independently and identically over entities:

$$\text{eFF}(\mathbf{X}; \mathbf{W}, \mathbf{b}) = \mathbf{X}\mathbf{W} + \mathbf{b}^\top, \mathbf{X} \in \mathbb{R}^{n^x \times d}, \mathbf{W} \in \mathbb{R}^{d \times h}, \mathbf{b} \in \mathbb{R}^h \quad (3)$$

Now, we specify the operation that defines an attention head, given the additional inputs of $\mathcal{S} \subseteq \mathbb{Z}^{[1, n^x]}$, a set of indices that selects which rows of the input \mathbf{X} are used to compute queries such that $\mathbf{X}_{\mathcal{S}} \in \mathbb{R}^{|\mathcal{S}| \times d}$, and \mathbf{M} , a binary obserability mask specifying which entities each query entity can observe (i.e. $M_{i,j} = 1$ when $i \in \mathcal{S}$ can incorporate information from $j \in \mathbb{Z}^{[1, n^x]}$ into its local context):

$$\text{Atten}(\mathcal{S}, \mathbf{X}, \mathbf{M}; \mathbf{W}^Q, \mathbf{W}^K, \mathbf{W}^V) = \text{softmax} \left(\text{mask} \left(\frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{h}}, \mathbf{M} \right) \right) \mathbf{V} \in \mathbb{R}^{|\mathcal{S}| \times h} \quad (4)$$

$$\mathbf{Q} = \mathbf{X}_{\mathcal{S}}\mathbf{W}^Q, \mathbf{K} = \mathbf{X}\mathbf{W}^K, \mathbf{V} = \mathbf{X}\mathbf{W}^V, \quad \mathbf{M} \in \{0, 1\}^{|\mathcal{S}| \times n^x}, \mathbf{W}^Q, \mathbf{W}^K, \mathbf{W}^V \in \mathbb{R}^{d \times h} \quad (5)$$

The $\text{mask}(\mathbf{Y}, \mathbf{M})$ operation takes two equal sized matrices and fills the entries of \mathbf{Y} with $-\infty$ in the indices where \mathbf{M} is equal to 0. After the softmax, these entries become zero, thus preventing the attention mechanism from attending to specific entities. This masking procedure is used in our case to uphold partial observability. Only one attention layer is permitted in the decentralized execution setting; otherwise information from unseen agents can be propagated through agents that are seen. \mathbf{W}^Q , \mathbf{W}^K , and \mathbf{W}^V are all learnable parameters of this layer. Queries, \mathbf{Q} , can be thought of as vectors specifying the type of information that an entity would like to select from others, while keys, \mathbf{K} , can be thought of as specifying the type of information that an entity possesses, and finally, values, \mathbf{V} , hold the information that is actually shared with other entities.

We define multi-head-attention as the parallel computation of attention heads as such:

$$\text{MHA}(\mathcal{S}, \mathbf{X}, \mathbf{M}) = \text{concat} \left(\text{Atten} \left(\mathcal{S}, \mathbf{X}, \mathbf{M}; \mathbf{W}_j^Q, \mathbf{W}_j^K, \mathbf{W}_j^V \right), j \in (1 \dots n^h) \right) \quad (6)$$

The size of the parameters of an attention layer does not depend on the number of input entities. Furthermore, we receive an output vector for each query vector.

B Generating Dynamic Sized Mixing Networks

Our two layer mixing network requires the following parameters to be generated: $\mathbf{W}_1 \in \mathbb{R}^{(|\mathcal{A}| \times h^m)}$, $\mathbf{b}_1 \in \mathbb{R}^{h^m}$, $\mathbf{w}_2 \in \mathbb{R}^{(h^m)}$, $b_2 \in \mathbb{R}$, where h^m is the hidden dimension of the mixing network and $|\mathcal{A}|$ is the set of agents.

Note from Eq. (4) that the output size of the layer is dependent on the size of the query set. As such, using attention layers, we can generate a matrix of size $|\mathcal{A}| \times h^m$, by specifying the set of agents, \mathcal{A} , as the set of queries \mathcal{S} from Eq. (4). We do not need observability masking since hypernetworks are only used during training and can be fully centralized. For each of the four components of the mixing network ($\mathbf{W}_1, \mathbf{b}_1, \mathbf{w}_2, b_2$), we introduce a hypernetwork that generates parameters of the correct size. Thus, for the parameters that are vectors (\mathbf{b}_1 and \mathbf{w}_2), we average the matrix generated by the attention layer across the $|\mathcal{A}|$ sized dimension, and for b_2 , we average all elements. This procedure enables the dynamic generation of mixing networks whose input size varies with the number of agents. Assuming $\mathbf{q} = [Q^1(\tau^1, u^1), \dots, Q^n(\tau^n, u^n)]$, then Q^{tot} is computed as:

$$Q^{\text{tot}}(\mathbf{s}, \tau, \mathbf{u}) = \sigma((\mathbf{q}^\top \mathbf{W}_1) + \mathbf{b}_1^\top) \mathbf{w}_2 + b_2 \quad (7)$$

where σ is an ELU nonlinearity [9].

C Environment Details

C.1 SAVETHECITY

The number of agents ranges from 2 to 8, and the number of buildings is always within 1 of the number of agents to keep each scenario approximately balanced in terms of difficulty. Agents are rewarded by 0.1 times the change in any buildings health (i.e. rewarded for building and penalized for burning). They receive a reward of 1 for completing a building, a penalty of -1 for a building burning down, a reward of 0.5 for extinguishing a fire, and a reward of 20 for solving the task (i.e. completing all buildings without letting a single one burn down). The base rate of fire reduction is 80 steps to put out a fire. A fire, if left alone will grow at a rate of 10% of its current strength. The base building rate is 80 steps to complete a building from scratch. A full strength fire will burn down a full-health fast burning building in 20 steps and a slow burning building in 80 steps. We use a 16x16 grid with nine possible building locations (randomized at each episode) for our experiments.

C.2 STARCRAFT

The standard version of SMAC loads map files with pre-defined and fixed unit types, where the global state and observations are flat vectors with segments corresponding to each agent and enemy. Partial observability is implemented by zeroing out segments of the observations corresponding to unobserved agents. The size of these vectors changes depending on the number of agents placed in the map file. Furthermore, the action space consists of movement actions as well as separate actions to attack each enemy unit. As such the action space also changes as the number of agents changes.

Our version loads empty map files and programmatically generates agents, allowing greater flexibility in terms of the units present to begin each episode. The global state is split into a list of equal-sized entity descriptor vectors (for both agents and enemies), and partial observability is handled by generating a matrix that shows what entities are visible to each agent. The variable-sized action space is handled by assigning each enemy a tag at the beginning of each episode and designating an action to attack each possible tag, of which there are a maximum number (i.e. the maximum possible number of agents in a scenario we would like to test on).

D Experimental Details

Our experiments were performed on a desktop machine with a 6-core Intel Core i7-6800K CPU and 3 NVIDIA Titan Xp GPUs, and a server with 2 16-core Intel Xeon Gold 6154 CPUs and 10 NVIDIA Titan Xp GPUs. Each experiment is run with 8 parallel environments for data collection and a single GPU. AI-QMIX on takes about 19.5 hours to run on SAVETHECITY and 24 hours on STARCRAFT. A-QMIX on takes about 12 hours to run on SAVETHECITY and 16 hours on STARCRAFT. Reported

times are on the desktop machine and the server runs approximately 15% faster due to more cores being available for running the environments in parallel.

E Hyperparameters

Hyperparameters were based on the PyMARL [28] implementation of QMIX and are listed in Table 1. All hyperparameters are the same in both `SAVETHECITY` and `STARCRAFT`. Since we train for 10 million timesteps (as opposed to the typical 2 million in standard SMAC), we extend the epsilon annealing period (for epsilon-greedy exploration) from 50,000 steps to 500,000 steps. For hyperparameters new to our approach (hidden dimensions of attention layers, number of attention heads, λ weighting of imagined loss), the specified values in Table 1 were the first values tried, and we found them to work well. For λ we also tried the values of 0.75 and 0.25, which we found to also work well but not quite as well as 0.5. The robustness of our approach to hyperparameter settings, as well as the fact that we do not tune hyperparameters per environment, is a strong indicator of the general applicability of our method.

Table 1: Hyperparameter settings across all runs and algorithms/baselines.

Name	Description	Value
lr	learning rate	0.0005
optimizer	type of optimizer	RMSProp [33]
optim α	RMSProp param	0.99
optim ϵ	RMSProp param	$1e - 5$
target update interval	copy live params to target params every _ episodes	200
bs	batch size (# of episodes per batch)	32
grad clip	reduce global norm of gradients beyond this value	10
$ D $	maximum size of replay buffer (in episodes)	5000
γ	discount factor	0.99
starting ϵ	starting value for exploraton rate annealing	1.0
ending ϵ	ending value for exploraton rate annealing	0.05
anneal time	number of steps to anneal exploration rate over	500000
h^a	hidden dimensions for attention layers	128
h^r	hidden dimensions for RNN layers	64
h^m	hidden dimensions for mixing network	32
# attention heads	Number of attention heads	4
nonlinearity	type of nonlinearity (outside of mixing net)	ReLU
λ	Weighting between standard QMIX loss and imagined loss	0.5

F Additional Results

F.1 Standard Regularization Methods for Improving Generalization

Inspired by prior work in generalization for single-agent RL [15, 10], we test several common regularization methods for supervised learning suggested by these works to be effective in improving generalization. These include Batch Normalization [16], Dropout [30], and weight decay. We also test Layer Normalization [2] due to its common use in attention models. These methods are tested with A-QMIX to compare their effectiveness in improving generalization with our approach. We tested the following values for weight decay: $5e - 3$, $1e - 3$, $5e - 4$, $1e - 4$, $5e - 5$, $1e - 5$, $5e - 6$, $1e - 6$, $5e - 7$, $1e - 7$ finding $5e - 7$ to work best. For dropout we test with drop probabilities of 0.05, 0.15, 0.25 finding 0.15 to work best. We also test a variant of dropout where, rather than dropping out individual entries from tensors at random, we mask out random entries from the attention weights, as this procedure is more similar to our method; however, we find that this hurts performance. Overall, we find that standard dropout with a drop probability of 0.15 improves most on A-QMIX, though its improvement is only marginal. We include it in our generalization tests as A-QMIX (Dropout). We do not use dropout in our exploration policy or in the computation of targets y^{tot} as suggested by Igl

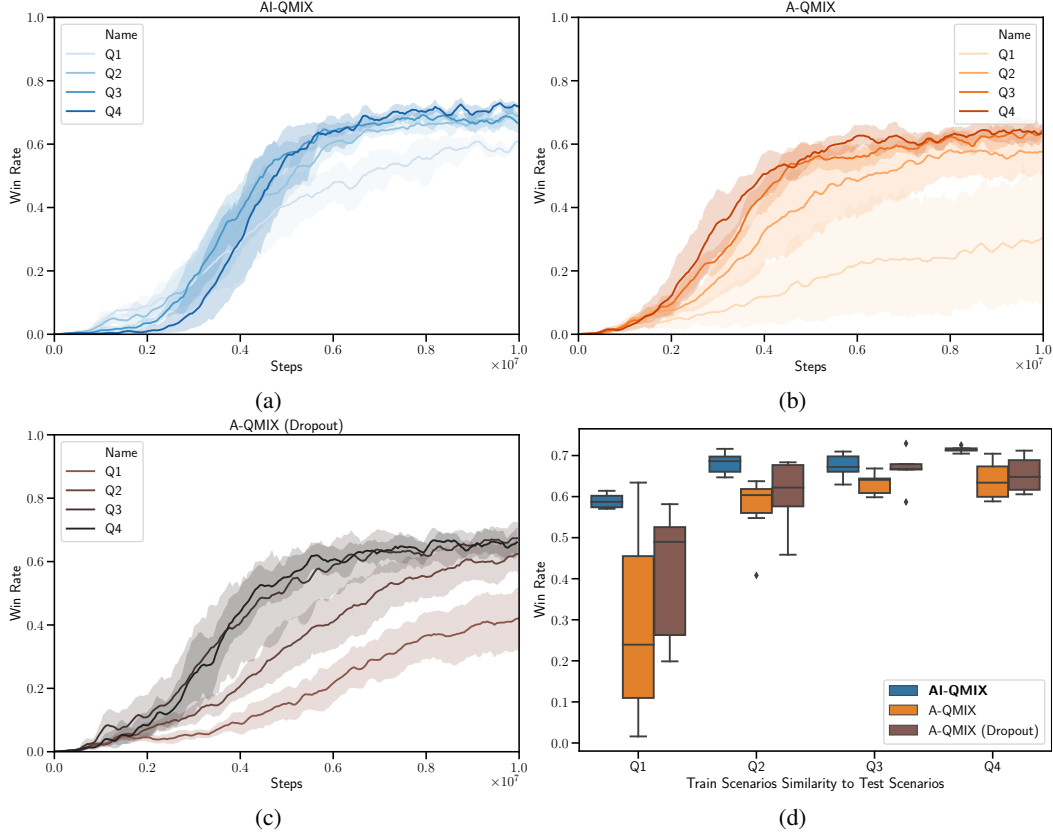


Figure 5: Generalization results on SAVETHECITY environment w/ dropout. Train scenarios are ranked by their mean overlap coefficient with the testing scenarios and separated into quartiles (Q1, Q2, Q3, Q4). **(top left)** Win rate over time for AI-QMIX w/ varying training scenario similarity to test scenarios. **(top right)** Win rate over time for A-QMIX w/ varying training scenario similarity to test scenarios. **(bottom left)** Win rate over time for A-QMIX (Dropout) w/ varying training scenario similarity to test scenarios. **(bottom right)** Final mean win rate on test scenarios while varying similarity of train scenarios to test scenarios. Dropout only helps generalization marginally.

et al. [15]. The results for dropout can be seen in Figure 5, where we see that it only has a marginal improvement on generalization over the baseline A-QMIX. In general, we find that methods used in prior work for improving generalization in the image domain are relatively ineffective when applied to our setting.