This paper describes objective technical results and analysis. Any subjective views or opinions that might be expressed in the paper do not necessarily represent the views of the U.S. Department of Energy or the United States Government.

SAND2022-8384C

Scaling Multi-Agent Reinforcement Learning via State Upsampling

Luis Pimentel[1,*], Rohan Paleja[1,*], Zheyuan Wang[1], Esmaeil Seraj[1], James E. G. Pagan[2], and Matthew Gombolay[1]
[1]Atlanta, GA, USA, [2]Albuquerque, NM, USA
[1]Georgia Institute of Technology, [2]Sandia National Laboratories
{lpimentel3, rohan.paleja, pjohnwang, eseraj3}@gatech.edu, jepagan@sandia.gov, matthew.gombolay@cc.gatech.edu

***Abstract*—We consider the problem of scaling Multi-Agent Reinforcement Learning (MARL) algorithms toward larger environments and team sizes. While it is possible to learn a MARL-synthesized policy on these larger problems from scratch, training is difficult as the joint state-action space is much larger. Policy learning will require a large amount of experience (and associated training time) to reach a target performance. In this paper, we propose a transfer learning method that accelerates the training performance in such high-dimensional tasks with increased complexity. Our method upsamples an agent's state representation in a smaller, less challenging, *source task* in order to pre-train a *target policy* for a larger, more challenging, *target task*. By transferring the policy after pre-training and continuing MARL in the target domain, the information learned within the source task enables higher performance within the target task in significantly less time than training from scratch. As such, our method enables the scalability of coordination problems. Furthermore, as our method only changes the state representation of agents across tasks, it is agnostic to the policy's architecture and can be deployed across different MARL algorithms. We provide results showing that a policy trained under our method is able to achieve up to a 7.88× performance improvement under the same amount of training time, compared to a policy trained from scratch. Moreover, our method enables learning in difficult target task settings where training from scratch fails.**

I. Introduction

Many real world domains including traffic control [7], sensor networks [30, 22], and Search and Rescue (SAR) [18] can be modeled as Multi-Agent Systems (MAS), where multiple agents must individually make decisions in a decentralized fashion to achieve high-performance collaboration [33]. Collaboration in these domains can greatly improve performance and task efficiency, and is often a required step to accomplish an overarching mission successfully [23]. However, learning high-performance team coordination strategies proves difficult due to the non-stationarity and credit-assignment problems [5]. Several MARL algorithms have been developed to overcome these challenges through the use of fully observable critics [5, 13, 14], value function factorization [19, 26], and communication [3, 4, 16, 23, 24, 25].

These algorithms have made significant advancements in MARL; however, challenges remain in scaling these algorithms to domains with larger environments and larger numbers of agents. Learning high-performance coordination policies in larger tasks is difficult due to the increasing joint state-action space that agents must explore, resulting in an increasingly complex credit-assignment problem [5] and a large variance of policy gradients during training [13]. More practically, often, training a MARL algorithm in these high-complexity configurations requires increased training time and high-performance computational resources. As the complexity of the problem increases, the computational effort required to learn a policy grows exponentially [6]. Our goal is to enable efficient training of MARL policies in high-dimensional and high-complexity problem configurations, by re-using knowledge learned from training in simpler configurations.

Previous work in scaling/varying the number of agents includes approaches that enable transfer through the use of Curriculum Learning and Evolutionary Learning [12], Graph Neural Network architectures [31], Transformer architectures [9, 34], and task representation learning [17]. However, these methods rely on an algorithm and/or specific architecture to enable performance improvements when transferring to new MARL tasks. Furthermore, these works do not explore transferring to larger environment sizes.

In this work, we propose an upsampling method to enable efficient training of a policy for a *target* MARL task with larger environment size and increased number of agents. In our method, we first pre-train the target policy on a *source* task with a smaller environment size and fewer agents. Once this policy has converged, we transfer to the target configuration. To achieve this, we propose a tensor-based state representation of agents and perform upsampling on the states to achieve a state representation of equal dimensions in both source and target tasks. Through our state upsampling method, we are able to transfer spatial knowledge to the target domain and achieve greater performance significantly faster than training from scratch. Our contributions are as follows:

1) We design a novel and efficient upsampling methodology that can be applied to MARL problems to upscale both the domain size and number of agents.
2) We ground our upsampling methodology within a POMDP formulation and provide a detailed description of how our method can be applied across various MARL frameworks and domains.
3) We present empirical evidence that our proposed method reduces aggregate training time of three published MARL frameworks, displaying its utility across both homogeneous and heterogeneous domains.

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology & Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.

II. Related Work

MARL Frameworks – The goal of MARL is to learn high-performance collaboration among agents via RL techniques. Recent works in MARL have considered problem settings where agents in the environment are allowed to communicate with each other in order to increase performance. In this setting, agents send messages to each other and integrate their received messages in order to make decisions [3, 24, 25, 16, 23]. Our work focuses on augmenting these frameworks with our upsampling method to increase training efficiency in more complex environments.

Transfer Learning – One technique to scale reinforcement learning is through transfer learning, which utilizes knowledge learned in one task to improve performance on a different but related task [29]. Within transfer learning for MARL, proposed techniques have included Intra-Agent Transfer [2], where an agent's knowledge is reused on a new task, and Curriculum Learning (CL), where learning is first done on an easier task and then transferred to more difficult target tasks. Our approach can be classified as a *policy transfer/policy reuse* technique, where knowledge is transferred by building a target policy from pre-trained source policies [35].

Tensor-Based State Representation – Tensor-based representations are ideal for cross-task state representations. Several works in MARL [1, 8, 21] utilize a tensor-based state representation for representing multiple agents within a grid. Han et al. [8] considers using a tensor-based state representation to transfer their MARL policy across team sizes by implementing per-grid policies. However, they are unable to transfer policies to larger environment scales as their method is dependent on their encoder-decoder architecture. The closest related work to ours is Schwab et al. [21], which also provides results transferring a policy trained on a small domain with few agents to a larger domain with more agents. However, their MARL formulation uses a fully-observable MDP, while ours is a POMDP augmented with message communication between agents to mitigate partial observability. To the best of our knowledge, we are the first to propose an upsampling-based method on MARL state representations that allows transferring a learned MARL policy across tasks (both varying in domain size and number of agents), and that can be integrated to benefit any existing training algorithms.

III. Problem Formulation

We model the learning process as a Multi-Agent Partially Observable Markov Decision Process (MA-POMDP) [10]. This can be represented as a tuple $\langle \mathcal{N}, \mathcal{S}, \mathcal{O}, \mathcal{A}, \mathcal{T}, r, \gamma \rangle$. Here, $\mathcal{N}$ is the number of agents, $\mathcal{S} = \{\times_i \mathcal{S}^i\}_{i=1}^{\mathcal{N}}$ is the joint set of agent state-spaces. Note that this joint state-space increases linearly as the environment size becomes larger and exponentially as the number of agents in the environment increases. $\mathcal{O} = \{\times_i \mathcal{O}^i\}_{i=1}^{\mathcal{N}}$ is the joint set of agent observation-spaces, $\mathcal{A} = \{\times_i \mathcal{A}^i\}_{i=1}^{\mathcal{N}}$ is the joint set of agent action-spaces, and $\mathcal{T}$ is the probability density function defining agent state transitions: $\mathcal{T} : \mathcal{S} \times \mathcal{A} \mapsto \mathcal{S}$. At each time-step t, agents can receive an observation, o_t^i, from the environment. Regardless of an agent having received an observation, at each time-step t, all agents in the joint state $\bar{s}_t = \left[s_t^i\right] \in \mathcal{S}$ take an action forming a vector of joint actions $\bar{a}_t = \left[a_t^i\right] \in \mathcal{A}$. At the next joint state $\bar{s}_{t+1}$, all agents received an individual immediate reward $r_t(s_t^i, a_t^i)$. Our objective is to learn an optimal policy, $\pi_i^*(s_t^i) : \mathcal{S} \mapsto \mathcal{A}$, that maximizes the total expected, discounted reward accumulated by an agent such that $\pi^{i*}(s_t^i) = \underset{\pi^i(s_t^i)}{\text{argmax}}\ \mathbb{E}_{\pi^i(s_t^i)}\left[R_t \,|\, \pi^i(s_t^i)\right]$ where $R_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k}$ and γ is the the temporal discount factor for each unit of time.

IV. Method

In this work, we define the *target task* as a challenging environment configuration and seek to accelerate the learning of a policy that is trained under the target task's difficult environment settings. Accordingly, we define this policy as the *target policy*. Furthermore, we define the *source task* as a less challenging environment configuration, under which we can pre-train the target policy in order to rapidly learn knowledge that can be used to warm-start learning in the more challenging target task. This involves transferring a coordination policy learned in a small domain with few agents to a large domain with more agents. In our MARL formulation with communication, an agent's available information at each time-step consists of its state (e.g., an agent's current location) and observation (e.g., positions of other agents and/or entities within a limited vision). An agent's state can be represented by its position in a one-hot encoded matrix, of which the dimensions are determined by the size of the environment. Therefore, directly transferring a policy from a smaller environment to a larger one is not possible as the dimensions of the state representation in the small environment are unequal to that in the larger environment. In the following sections, we explain how we utilize upsampling to enable this transfer.

A. State Upsampling Transformation

We transform the state representations of each agent in the source task, $s_t^i \in \mathbb{R}^{p \times p}$, to a state representation, $z_t^i \in \mathbb{R}^{q \times q}$, that can then be used to pre-train the target policy, π_i^{target}, on the source task. Here, p is the length of the source task's environment, and q is the length of the target task's environment. Our only assumptions are that the state can be represented by a matrix in both tasks, and that $q > p$. For our state transformation operator, $\phi : \mathbb{R}^{p \times p} \mapsto \mathbb{R}^{q \times q}$, we utilize nearest-neighbor upsampling interpolation [28], with an upsampling factor, $f = \frac{q}{p}$. In this operation, one cell in s_t^i, becomes an $f \times f$ grid of cells in the target state representation z_t^i, with each cell having the same feature value as in the s_t^i cell. This is repeated for all cells in s_t^i to form the upsampled target state representation, z_t^i. This upsampling can be visualized in Figure 1, where a 5x5 state representation is transformed to a 10x10 state representation via state upsampling. We note that our method is not limited to environments with equal dimensions as we can choose an appropriate state transformation function, ϕ.

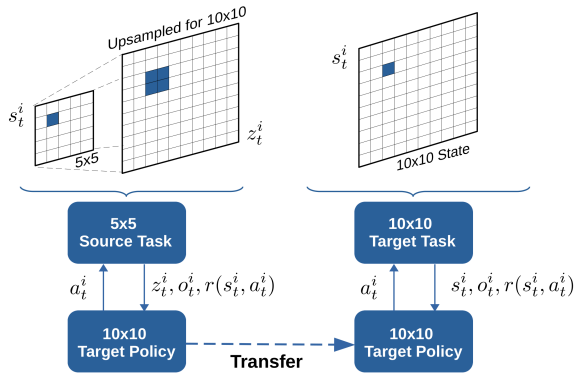


Fig. 1: Visual of our transfer method. Our SUMA-POMDP formulation is deployed on a 5×5 source task to enable transferring the target policy to a 10×10 target task.

B. State-Upsampled Multi-Agent POMDP (SUMA-POMDP)

The learning processes of the source and target tasks can be modeled by the MA-POMDP formulation introduced in Section III. As policy learning in the source task is much easier, requiring less environment experience and training times, we pre-train the target policy on a source task using a State-Upsampled Multi-Agent POMDP (SUMA-POMDP) represented as the tuple $\langle \mathcal{N}, \phi, \mathcal{Z}, \mathcal{S}, \mathcal{O}, \mathcal{A}, \mathcal{T}, r, \gamma \rangle$. This problem formulation follows the same definitions as the regular MA-POMDP, with the addition of the state transformation function, ϕ, and the upsampled-state space, $\mathcal{Z}$. Here, the agents' joint state is represented by $\bar{z}_t = \left[\phi(s_t^i)\right] \in \mathcal{Z}$. We note that, while currently the observation space remains fixed, our method could also be utilized to upsample an agent's observation in the source task, allowing an agent to gain spatial knowledge of other agents, entities and/or environment features. Furthermore, the reward that agents receive remain unchanged and the state transition probability density function is still defined by $\mathcal{T} : \mathcal{S} \times \mathcal{A} \mapsto \mathcal{S}$. As such, we are able to warm-start the target policy by pre-training in a simpler task, $\pi_i^{\text{target}}(z_t^i) : \mathcal{Z} \mapsto \mathcal{A}$. We utilize the Actor-Critic method described in VII-A to pre-train the target policy within the source task. Once $\pi_i^{\text{target}}(z_t^i)$ has converged, we transfer this policy to the target task: $\pi_i^{\text{target}}(s_t^i) \leftarrow \pi_i^{\text{target}}(z_t^i)$. We continue training, following the MA-POMDP formulation, as we no longer need the state transformation function ϕ. We learn the optimal target policy, $\pi_i^*(s_t^i)$, by training in the target domain using the Actor-Critic method described in Section VII-A.

C. Tensor Representation for Team Size Scaling

As learning in the source task is done with less agents, our method also requires the ability to scale to larger team sizes. This capability is enabled via our tensor-based state representation which can be described by the following multi-dimensional tensor: $\left(i, c, s_x, s_y\right)$. Here, the i dimension indexes an agent in the environment, the c dimension corresponds to that agent's type, and the s_x and s_y dimensions correspond to coordinates in the grid. As a brief example, Agent 0 of Type 1 at grid location $(3, 4)$, would have the following value: $(0, 1, 3, 4) = 1$. Additionally, an agent's observations are processed through the same tensor-based representation, and we note that the observation tensor's c dimension can encode other agents, entities, and/or environment features such as obstacles. In our method, these tensors are processed through convolutional and fully-connected layers, generating an encoded representation of the agent's state-observation for the MARL algorithm. As we treat the first dimension as the batch-input, and the second dimension as the channel input to the preprocessing layers, we are able to scale to larger team sizes without additional parameterization.

V. Experiments and Results

To evaluate the efficacy of our method, we conduct experiments across two MARL domains and three MARL algorithms with communication. The first domain we use is Predator-Prey (PP) [24], a *homogeneous* coordination problem, where Predator agents with limited vision must find a stationary Prey and move to its location within a grid world environment. The second domain we use is Predator-Capture-Prey (PCP) [23], a *heterogeneous* coordination problem where agents maintain different observation and action-spaces. This team heterogeneity further increases the complexity of this domain, when compared to PP [24], leading to increased training times to achieve a target performance, in the same grid size and total number of agents. Across domains, the objective is to find/find-and-capture the prey in the least number of time-steps, respectively. We refer readers to [23] for further details on the PP and PCP domains. We note that our method is not limited to grid-world environments; rather, it is limited to environments that can be represented as tensors. Therefore, our method can still be deployed on a variety of continuous MARL environments and tasks, such as MPE [13], GRF [11], and SMAC [20].

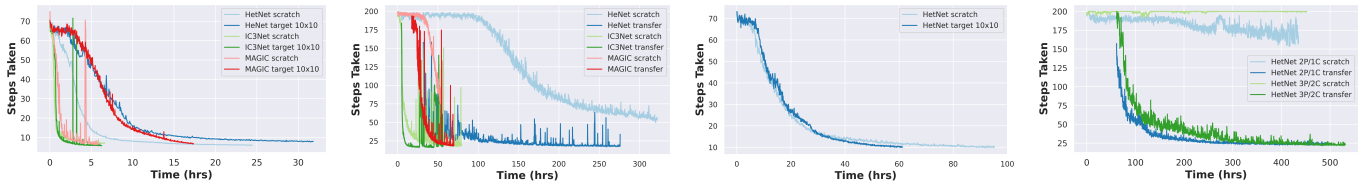
A. Environment Representation and Baselines

Environment Representation – We utilize the tensor-based representation (Section IV-C) to represent the state and observation within these grid-based domains as opposed to the flattened representation used in the original code-base of Singh et al. [24]. We provide further details in Section VII-B.

Baselines – We deploy our upsampling method to IC3Net [24], MAGIC [16], and HetNet [23] to evaluate whether our method can help scale different MARL algorithms. We use all methods within the PP [24] domain, and only utilize HetNet within the PCP [23] domain as it is specifically designed to handle communication across heterogeneous agents.

B. Transfer Experiments

Experiment Details – In our experiments, the source tasks are 5×5 environments with 3 total agents: 3 Predator (P) agents for the PP [24] domain, and 2 Predator (P) and 1 Capture (C) agents in the PCP [23] domain. The target tasks are 10×10 environments with 5 total agents: 5 Predator (P) agents for the PP domain, and 3 Predator (P) and 2 Capture (C) agents in the PCP domain. We pre-train the target policy from scratch, on the source tasks, under the SUMA-POMDP



(a) 5×5 PP source task. (b) 10×10 PP target task. (c) 5×5 PCP source task. (d) 10×10 PCP target task.

Fig. 2: In Figures 2a – 2d, we display source/target learning curves with respect to estimated aggregate training time as detailed in Section V-C. We note that in Figures 2b and 2d, the transferred policies are shifted to account for time spent pre-training.

formulation introduced in Section IV-B. Once the target policy has converged in the source task, we transfer the policy to the target task and continue training, without state upsampling. In the PP [24] domain, we increase the team size in the target task simultaneously. In the PCP [23] domain, we do one set of experiments keeping the team size the same as in the source task, and another increasing the team size.

C. Evaluation Metrics

We assess two aspects of performance: **Q1)** Does training under our SUMA-POMDP formulation affect performance within the source task? and **Q2)** What are the benefits obtained via our policy transfer procedure? 1) is assessed directly via the performance metric in PP and PCP, steps-taken, by comparing performance between training from scratch and training under the SUMA-POMDP formulation. 2) is assessed via the aggregated training time required for a policy to reach a target performance and the performance achieved within a fixed amount of training time.

D. Results

Q1) Pre-training Results – From our results in Figures 2a and 2c, we observe that the target policies, pre-trained on the source tasks, are able to converge and achieve similar performance as training from scratch, for IC3Net [24], MAGIC [16], and HetNet [23] on the PP [24] domain, and for HetNet [23] on the PCP domain. In Figure 2a, the target policy using HetNet takes longer training time to converge. However, even with this difference, the aggregate training time after transfer is still less compared to training from scratch, as we discuss in Q2). From these results, we conclude that pre-training the target policy under our SUMA-POMDP formulation does not negatively effect learning performance on the source task. As the target policy is able to achieve convergence on the source task, it is able to learn knowledge about the domain that can be leveraged when transferring to the more difficult target task.

Q2) Transfer Results – For the PP target task, we transfer the IC3Net, MAGIC, and HetNet target policies after pre-trainining on the source task for 350, 590, and 500 epochs, respectively. In Figure 2b, we observe that the transferred policies in the PP domain are able to achieve convergence and better performance with less aggregate training time, compared to training from scratch, for all baselines. Specifically, in Table Ia, we observe that given a fixed amount of training time, transferred policies are able to achieve up to a 2.68× - 6.99× performance improvement, compared to training from scratch,

Experiment			Time (hrs)	Performance (Avg. Steps-Taken)	
				Scratch	Transfer
HetNet	10 × 10	5P	100.00	189.70	**27.13**
IC3Net	10 × 10	5P	10.00	55.59	**20.72**
MAGIC	10 × 10	5P	30.00	192.61	**69.67**

(a) Experiments in the homogeneous PP [24] domain.

Experiment			Time (hrs)	Performance (Avg. Steps-Taken)	
				Scratch	Transfer
HetNet	10 × 10	2P/1C	200.00	184.62	**30.19**
HetNet	10 × 10	3P/2C	350.00	199.85	**25.34**

(b) Experiments in the heterogeneous PCP [23] domain.

TABLE I: Performance in a fixed amount of aggregate training time as detailed in Section VII-C.

across all baselines. For the PCP target task, we transfer the HetNet target policy after 700 epochs of pre-training on the source task. For this domain, we include experiments where the same transferred HetNet policy is used to initialize training with a 2P/1C team (as in the source task), and training with a larger 3P/2C team. For both experiments, we observe in Figure 2d that the transferred policies are able to achieve convergence and better performance with less aggregate training time, compared to training from scratch. Specifically, in Table Ib, we observe that given a fixed amount of training time, transferred policies are able to achieve up to a 6.11× - 7.88× performance improvement, compared to training from scratch, across both the 2P/1C and 3P/2C tasks. Notably, we observe that the policies trained from scratch are unable to achieve high-performance in over 400 hours of training while the transferred policies are able to within a few hours after transfer.

VI. Conclusion

In order to scale MARL algorithms to larger environments and team sizes, we present a transfer learning method that leverages learned knowledge from training on a simpler task with a smaller environment and fewer agents. We introduce the State-Upsampled Multi-Agent Partially Observable Markov Decision Process (SUMA-POMDP) problem formulation to allow pre-training a target policy on these simpler source tasks. Furthermore, we present a tensor-based state representation that allows for scaling to larger team sizes. We present empirical evidence that shows our method is able to achieve greater performance with significantly less training experience and time, across multiple MARL algorithms and domains. Notably, our method allows for policy convergence in complex problems, providing up to a 7.88× performance gain, while conventional training from scratch is unable to make progress.

ACKNOWLEDGMENTS

This work is supported by the Laboratory Directed Research and Development program at Sandia National Laboratories, a multimission laboratory managed and operated by National Technology and Engineering Solutions of Sandia LLC, a wholly owned subsidiary of Honeywell International Inc. for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525. This paper describes objective technical results and analysis. Any subjective views or opinions that might be expressed in the paper do not necessarily represent the views of the U.S. Department of Energy or the United States Government.

REFERENCES

[1] Niranjan Balachandar, Justin Dieter, and Govardana Sachithanandam Ramachandran. Collaboration of ai agents via cooperative multi-agent deep reinforcement learning. *arXiv preprint arXiv:1907.00327*, 2019.

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

[3] Abhishek Das, Théophile Gervet, Joshua Romoff, Dhruv Batra, Devi Parikh, Mike Rabbat, and Joelle Pineau. Tarmac: Targeted multi-agent communication. In *International Conference on Machine Learning*, pages 1538–1546. PMLR, 2019.

[4] Jakob Foerster, Ioannis Alexandros Assael, Nando De Freitas, and Shimon Whiteson. Learning to communicate with deep multi-agent reinforcement learning. *Advances in neural information processing systems*, 29, 2016.

[5] Jakob Foerster, Gregory Farquhar, Triantafyllos Afouras, Nantas Nardelli, and Shimon Whiteson. Counterfactual multi-agent policy gradients. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32, 2018.

[6] Sven Gronauer and Klaus Diepold. Multi-agent deep reinforcement learning: a survey. *Artificial Intelligence Review*, 55(2):895–943, 2022.

[7] Hodjat Hamidi and Ali Kamankesh. An approach to intelligent traffic management system using a multi-agent system. *International Journal of Intelligent Transportation Systems Research*, 16(2):112–124, 2018.

[8] Lei Han, Peng Sun, Yali Du, Jiechao Xiong, Qing Wang, Xinghai Sun, Han Liu, and Tong Zhang. Grid-wise control for multi-agent reinforcement learning in video game ai. In *International Conference on Machine Learning*, pages 2576–2585. PMLR, 2019.

[9] Siyi Hu, Fengda Zhu, Xiaojun Chang, and Xiaodan Liang. Updet: Universal multi-agent reinforcement learning via policy decoupling with transformers. *arXiv preprint arXiv:2101.08001*, 2021.

[10] Leslie Pack Kaelbling, Michael L Littman, and Anthony R Cassandra. Planning and acting in partially observable stochastic domains. *Artificial intelligence*, 101(1-2):99–134, 1998.

[11] Karol Kurach, Anton Raichuk, Piotr Stańczyk, Michał Zajac, Olivier Bachem, Lasse Espeholt, Carlos Riquelme, Damien Vincent, Marcin Michalski, Olivier Bousquet, and Sylvain Gelly. Google research football: A novel reinforcement learning environment. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34(04):4501–4510, Apr. 2020. doi: 10.1609/aaai.v34i04.5878. URL https://ojs.aaai.org/index.php/AAAI/article/view/5878.

[12] Qian Long, Zihan Zhou, Abhibav Gupta, Fei Fang, Yi Wu, and Xiaolong Wang. Evolutionary population curriculum for scaling multi-agent reinforcement learning. *arXiv preprint arXiv:2003.10423*, 2020.

[13] Ryan Lowe, Yi I Wu, Aviv Tamar, Jean Harb, OpenAI Pieter Abbeel, and Igor Mordatch. Multi-agent actor-critic for mixed cooperative-competitive environments. *Advances in neural information processing systems*, 30, 2017.

[14] Hangyu Mao, Zhengchao Zhang, Zhen Xiao, and Zhibo Gong. Modelling the dynamic joint policy of teammates with attention multi-agent ddpg. *arXiv preprint arXiv:1811.07029*, 2018.

[15] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pages 1928–1937. PMLR, 2016.

[16] Yaru Niu, Rohan Paleja, and Matthew Gombolay. Multi-agent graph-attention communication and teaming. In *Proceedings of the 20th International Conference on Autonomous Agents and MultiAgent Systems*, pages 964–973, 2021.

[17] Rongjun Qin, Feng Chen, Tonghan Wang, Lei Yuan, Xiaoran Wu, Zongzhang Zhang, Chongjie Zhang, and Yang Yu. Multi-agent policy transfer via task relationship modeling. *arXiv preprint arXiv:2203.04482*, 2022.

[18] Jorge Peña Queralta, Jussi Taipalmaa, Bilge Can Pullinen, Victor Kathan Sarker, Tuan Nguyen Gia, Hannu Tenhunen, Moncef Gabbouj, Jenni Raitoharju, and Tomi Westerlund. Collaborative multi-robot search and rescue: Planning, coordination, perception, and active vision. *Ieee Access*, 8:191617–191643, 2020.

[19] 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 *International Conference on Machine Learning*, pages 4295–4304. PMLR, 2018.

[20] 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. *arXiv preprint arXiv:1902.04043*, 2019.

[21] Devin Schwab, Yifeng Zhu, and Manuela Veloso. Tensor action spaces for multi-agent robot transfer learning. In *2020 IEEE/RSJ International Conference on Intelligent*

Robots and Systems (IROS), pages 5380–5386. IEEE, 2020.

[22] Esmaeil Seraj and Matthew Gombolay. Coordinated control of uavs for human-centered active sensing of wildfires. In *2020 American Control Conference (ACC)*, pages 1845–1852. IEEE, 2020.

[23] Esmaeil Seraj, Zheyuan Wang, Rohan Paleja, Daniel Martin, Matthew Sklar, Anirudh Patel, and Matthew Gombolay. Learning efficient diverse communication for cooperative heterogeneous teaming. In *Proceedings of the 21st International Conference on Autonomous Agents and Multiagent Systems*, pages 1173–1182, 2022.

[24] Amanpreet Singh, Tushar Jain, and Sainbayar Sukhbaatar. Learning when to communicate at scale in multiagent cooperative and competitive tasks. *arXiv preprint arXiv:1812.09755*, 2018.

[25] Sainbayar Sukhbaatar, Rob Fergus, et al. Learning multiagent communication with backpropagation. *Advances in neural information processing systems*, 29, 2016.

[26] Peter Sunehag, Guy Lever, Audrunas Gruslys, Wojciech Marian Czarnecki, Vinicius Zambaldi, Max Jaderberg, Marc Lanctot, Nicolas Sonnerat, Joel Z Leibo, Karl Tuyls, et al. Value-decomposition networks for cooperative multi-agent learning. *arXiv preprint arXiv:1706.05296*, 2017.

[27] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[28] Richard Szeliski. Computer vision - algorithms and applications. In *Texts in Computer Science*, 2011.

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

[30] Gabriel Villarrubia, Juan F De Paz, Daniel H Iglesia, and Javier Bajo. Combining multi-agent systems and wireless sensor networks for monitoring crop irrigation. *Sensors*, 17(8):1775, 2017.

[31] 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 *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 7293–7300, 2020.

[32] Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3):229–256, 1992.

[33] Michael Wooldridge. *An introduction to multiagent systems*. John wiley & sons, 2009.

[34] Tianze Zhou, Fubiao Zhang, Kun Shao, Kai Li, Wenhan Huang, Jun Luo, Weixun Wang, Yaodong Yang, Hangyu Mao, Bin Wang, et al. Cooperative multi-agent transfer learning with level-adaptive credit assignment. *arXiv preprint arXiv:2106.00517*, 2021.

[35] Zhuangdi Zhu, Kaixiang Lin, and Jiayu Zhou. Transfer learning in deep reinforcement learning: A survey. *arXiv preprint arXiv:2009.07888*, 2020.

VII. Appendix

A. Actor-Critic (AC) Methods

We utilize an Actor-Critic (AC) method to learn an Actor policy, $\pi^i_\theta(a^i_t|s^i_t)$, parameterized by θ. At every time-step t, the Actor specifies the action a^i_t that an agent i must take, while in the state s^i_t, in order to maximize the expected future discounted reward. We apply gradient descent on the parameters of the Actor, based on the on a Critic $Q(s^i_t, a^i_t)$. The Critic is a state-action value function that assesses the value of the actions taken by the Actor [27]. By utilizing a centralized Critic [13], we are able to approximately solve the credit-assignment problem in our MARL formulation [5]. Our goal is to maximize an agent's expected reward $J(\theta) = \mathbb{E}_{\pi^i_\theta}\left[R_t \,|\, \pi^i_\theta\right]$. Utilizing R_t is an estimate of $Q(s^i_t, a^i_t)$ [15], and a learned state-value function $V^\psi(s^i_t)$, parameterized by ψ, as an approximate baseline function to reduce variance in the policy gradient [32], we obtain the following unbiased gradient: $\nabla_\theta J(\theta) = \mathbb{E}_{\pi^i_\theta}\left[\nabla_\theta \log \pi^i_\theta(a^i_t|o^i_t)\big(R_t - V^\psi(s^i_t)\big)\right]$.

B. Additional Environment Details

Within the PP [24] and PCP[23] environments, we represent the state with the same tensor-based representation detailed in Section IV-C: $\big(i, c, s_x, s_y\big)$. The cardinality of the i dimension is determined by the number of agents in the environment and the cardinality of the s_x and s_y dimensions are determined by the size of the environment. We note that for the state's tensor, the size of the c dimension is 1. Additionally, an agent's observation is encoded as $m \times m$ grid representation, where m is the range of the predator's vision. An agent's observation tensor can also be represented by the same multi-dimensional tensor. The dimensions are defined the same as in the state tensor-based representation, however the cardinality of the dimensions s_x and s_y are of m. Furthermore, the dimension c, representing the agent type, is of cardinality two for the PP environment and three for the PCP environment.

C. Additional Training Details

Training Resources – All of our experiments are run on a GPU cluster at the Georgia Institute of Technology with two 64-core AMD 7452 CPUs, eight Nvidia A40 48GB GPUs, and 512 GB of RAM. For each experiment, we only utilize the CPU for training and distribute the training over 4 processes.

D. Additional Details on Evaluation Metrics

Time Estimation – Each experiment was run once, utilizing a fixed random seed, until convergence or a maximum time threshold was reached. We estimate training time required for training each framework, under the same domain and experiment settings, by computing the average time-per-epoch over 100 epochs of training on the same machine. We multiply this by the number of training epochs across every experiment to generate the time data seen in Figures 2a-2d. We summarize these computed time statistics in Table II. We note that there is not a large difference in the average time-per-epoch between training a policy on the source task from scratch and training the target policy on the source task with state upsampling. (e.g. Table IIa Row 1 and Row 2). Likewise, there is not a large difference in the average time-per-epoch for a policy trained from scratch on the target task, and a policy transferred to the target task after pre-training (e.g. Table IIb Row 3, scratch v.s. transfer). As the training times are similar, we are able to make fair comparisons in Figure 2 and conclude that the performance improvement in training time is due to the knowledge transfer enabled by our method.

Performance Comparison – In Table I, we compare training performance between a policy trained from scratch on the target task, and a policy that is transferred to the target task after pre-training on the source task. To compute this metric, we take the average performance (measured by the average steps-taken by an agent) during a specified fixed hour of training. As the total training time can vary across algorithms and task configurations, we choose an appropriate fixed hour for each calculation.

Experiment				Avg. time per epoch (sec)	
				Scratch	Transfer
HetNet	5×5	3P		199.96 ± 0.73	-
HetNet	5×5	3P	US ×2	229.45 ± 0.66	-
IC3Net	5×5	3P		47.72 ± 0.24	-
IC3Net	5×5	3P	US ×2	50.03 ± 0.25	-
MAGIC	5×5	3P		98.69 ± 0.52	-
MAGIC	5×5	3P	US ×2	105.89 ± 0.49	-
HetNet	10×10	5P		580.76 ± 2.44	439.49 ± 4.93
IC3Net	10×10	5P		143.13 ± 0.51	132.62 ± 0.87
MAGIC	10×10	5P		313.80 ± 1.16	256.24 ± 0.79

(a) Experiments in the homogeneous PP [24] domain.

Experiment				Avg. time per epoch (sec)	
				Scratch	Transfer
HetNet	5×5	2P/1C		490.64 ± 2.46	-
HetNet	5×5	2P/1C	US ×2	315.05 ± 0.87	-
HetNet	10×10	2P/1C		784.00 ± 5.45	791.37 ± 5.05
HetNet	10×10	3P/2C		835.35 ± 7.57	846.94 ± 3.44

(b) Experiments in the heterogeneous PCP domain.

TABLE II: Time related statistics for all experiments. We estimate the average amount of training time per epoch (± Standard Error) over 100 epochs of training on the same machine.