Decentralized Multi-Robot Control with Deep Reinforcement Learning

Nathan Butler, Noah Boehme, Ashutosh Gupta, Ajay Ratty
Oregon State University
{butlnath, boehmen, guptaash, rattya}@oregonstate.edu

Abstract

Multi-robot systems require both task allocation and motion control to operate efficiently in cluttered environments. Deep Reinforcement Learning (DRL) has shown promise, with Convolutional Neural Networks (CNNs) excelling in motion control and Graph Neural Networks (GNNs) enabling coordination through message passing. However, their comparative effectiveness in multi-robot systems remains unexplored. This paper evaluates CNN- and GNN-based DRL policies for decentralized multi-robot control, assessing whether CNNs can implicitly develop coordination from local sensor inputs and whether GNNs offer advantages through explicit communication. While training performance shows that these policies can learn useful control inputs, qualitative analysis of the resulting trajectories indicates limited support for coordinated actions.

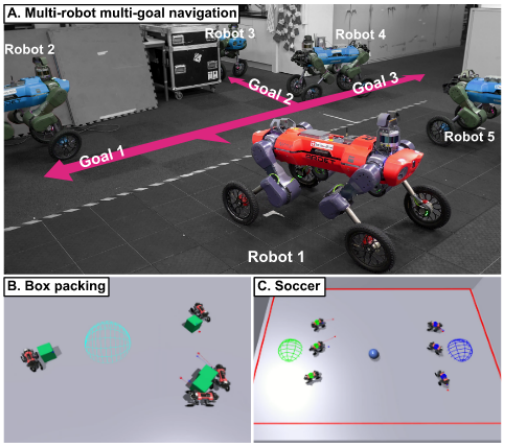

Figure 1. Multi-robot goal navigation is important for numerous domains, including high-congestion office settings, warehouse applications, and more.

1. Introduction

Multi-robot systems are increasingly being deployed in real-world domains such as environmental monitoring, warehouse automation, and search and rescue [11], [9] [13]. These domains require robots to navigate through cluttered environments (see Figure 1) while coordinating to efficiently complete tasks. To achieve high performance in these settings, the system must be capable of 1) allocating tasks between robots, and 2) controlling robot motions to avoid obstacles and reach desired locations. Designing policies that enable multi-robot teams to accomplish these objectives in a decentralized manner remains a fundamental challenge in robotics.

Deep Reinforcement Learning (DRL) has emerged as an effective method for training robot control policies for cluttered environments. Convolutional Neural Network (CNN)-based DRL policies have shown strong performance in motion control tasks by learning how to interpret raw sensor data, such as images or LiDAR readings, to produce safe trajectories [16] [18]. Meanwhile, Graph Neural Network (GNN)-based DRL policies have shown to be effective for multi-robot task allocation by facilitating structured message passing between robots [6] [3]. These approaches have shown promise in their respective domains; however, to the best of our knowledge, no prior works have evaluated the strengths and weaknesses of CNN-based and GNN-based policies for multi-robot control *and* task allocation.

Understanding whether CNN-based policies can implicitly develop coordination strategies using only local sensor inputs, or whether explicit message-passing via GNNs provides a critical advantage, is essential for designing scalable multi-robot learning frameworks. In this paper, we evaluate each method's ability to 1) process a robot's surroundings to take informed actions and 2) guide the robot to take coordinated actions within the context of the multi-robot system.

For CNN-based policies, we investigate whether coordination can emerge from local camera images. For GNN-based policies, we model each robot's surroundings as a graph and incorporate message passing to enable robots to share relevant information. Through this analysis, we aim to provide insights into the strengths and limitations of each approach, informing future developments in DRL-based multi-robot coordination.

2. Background

We review related works in DRL, CNNs for robot control, and GNNs for robot control. DRL has been shown to be an effective method for training these policies in both single agent and multi-agent settings. However, despite their advantages, multi-robot coordination remains a complex problem, particularly in dynamic and cluttered environments where decentralized decision-making and obstacle avoidance present fundamental challenges.

2.1. Deep Reinforcement Learning for Multi-Robot Control

DRL has emerged as a powerful technique for training control policies in complex, high-dimensional spaces. DRL has demonstrated success in solving challenging problems such as playing complex games [7] and decentralized collision avoidance [17]. By leveraging neural networks, DRL-based policies can learn to map raw sensor inputs to optimal actions, reducing the need for manually engineered control strategies.

Recent research in multi-agent reinforcement learning (MARL) has further expanded the applicability of DRL to collaborative multi-robot settings. Two popular approaches in MARL include Independent Proximal Policy Optimization (IPPO) and Multi-Agent Proximal Policy Optimization (MAPPO). IPPO treats each agent as an independent learner with its own policy and critic, while MAPPO enables centralized training with decentralized execution by sharing the critic information across agents. Although simple, IPPO has been shown to perform competitively in complex cooperative scenarios [5], while MAPPO improves sample efficiency and coordination in tightly coupled tasks [20]. These algorithms form the backbone of many current DRL-based approaches for scalable multi-robot control.

2.2. CNNs for Multi-Robot Control

CNNs are widely used in DRL for their ability to extract spatial features from high-dimensional sensor inputs, such as camera images or LiDAR scans. CNN-based control policies have been successfully applied to autonomous navigation tasks, enabling robots to learn motion strategies based on visual or depth information through CNN-based feature extraction [16, 19]. In other works, CNNs have been provided with global and local map information to generate multi-robot trajectories [17]. However, a key limitation of prior work in CNN-based policies in multi-robot systems is a reliance on additional external coordination strategies.

2.3. GNNs for Multi-Robot Control

GNNs offer a structured way to represent multi-robot interactions by modeling robots as nodes in a graph and their relationships (e.g., proximity, communication links) as edges. Through message passing, GNNs allow robots to share information and coordinate their actions efficiently. Recent work has demonstrated the effectiveness of GNN-based DRL for large-scale task allocation and multi-agent collaboration [12, 6]. Many GNN-based policy implementations, such as [10], rely on expert demonstrations to learn high-performing behaviors, and are often unavailable and costly to acquire. Additional works have proposed robust frameworks for real-world deployments of GNN-based policies, highlighting the potential of these powerful networks [3]. Compared to CNNs, GNNs provide an explicit mechanism for coordination, potentially leading to more robust multi-robot behaviors in complex scenarios.

3. Methods

In this section, we present the MARL algorithms used in training and discuss the design details of our CNN and GNN policies.

3.1. Multi-Agent Training Algorithms

To train multi-agent policies, IPPO and MAPPO have shown to be sample efficient, capable of learning complex behaviors, and competitive with other state-of-the-art multi-agent DRL algorithms. Both of these algorithms extend PPO [14], an actor-critic algorithm, which optimizes the long-term reward received under a given policy and a value function estimator, while balancing exploitation and exploration.

IPPO is the most straightforward implementation of PPO for multiple agents, where each agent has its own actor and its own critic. A given actor and critic receive their agent's state information and produce an action and a value of that state, respectively. To train a given actor and critic, an agent generates state, action, and reward data by interacting with an environment. Using these experiences, the actor and critic learn actions and states, respectively, that lead to high rewards. Formally, the following loss function is optimized:

$$L = \min\left(r_t A(s,a), \text{clip}\left(r_t, 1-\epsilon, 1+\epsilon\right) A(s,a)\right), \quad (1)$$

where $r_t = \frac{\pi(a|s)}{\pi_{old}(a|s)}$, $\epsilon \in [0, 1)$, and $A(s,a)$ is the advantage function of a given state and action pair.

MAPPO can either learn an actor policy for each agent or share a policy for all agent, while always sharing a critic across all the agents. The critic receives every agent's state information and produces a value for the entire system. This shared value estimate captures how the well all of the agents are contributing to the overall objective, whereas an IPPO critic only captures how well the current agent is doing. Every agent generates its own data to train its actor, but all agent data is used for training the critic. MAPPO optimizes

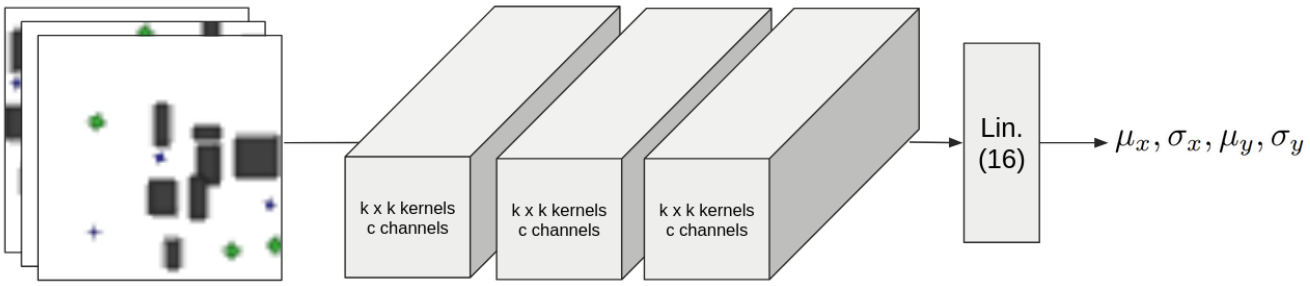

Figure 2. An example of a CNN policy used for producing actions for each robot. There are three CNN layers with c channels and a linear layer used to predict actions (force distributions for 2D robot motions).

the same loss function as IPPO, see Equation 1, but the advantage estimate is different, since the critic uses all agent's states to produce a value. As we consider homogeneous agents with equivalent state and action spaces in our experimental domain, we elect to share the actor policy among all agents for MAPPO. In this case, the actor is trained using data generated by all agents, which helps to accelerate training.

3.2. CNN Architecture

To parameterize a control policy using a CNN, image-based observations were used as input. Each robot received a 64 × 64 × 3 RGB image from its local perspective, capturing nearby tasks, obstacles, and other robots. This image was passed through a three-layer convolutional neural network, with each layer applying a kernel, followed by a *tanh* activation. Stride-based convolution for downsampling was used in the first two convolution layers to reduce the spatial resolution while increasing the receptive field. See the full policy architecture in Figure 6.

The final convolutional output is passed through a fully connected layer with 16 hidden units to produce the action distribution parameters. The network predicts the mean and standard deviation for each action dimension, $\mu_x, \sigma_x, \mu_y, \sigma_y$, using a *tanh* activation to ensure the output remains bounded within the action space.

Separate networks were used for the actor and critic, though both shared the same high-level architecture. The actor used only local visual input to predict actions, while the critic estimated the state value. Because the image observations are robot-centric, this architecture allows decentralized execution.

MAPPO was used to train the CNN policy since each agent only had a local observation. Using MAPPO enables the critic to judge the value of all robot states, allowing for cooperative behavior to be learned while maintaining decentralized policy execution.

3.3. GNN Architecture

To parameterize a multi-robot policy using a GNN, the states, which are used as input, must be defined. The state used for the GNN policy is the current robot's location and velocity along with the relative distance of the current robot to all other robots, obstacles, and tasks. Using this information, a graph can be constructed, where each node holds its relative distance to the current robot. To incorporate information across nodes, an aggregation function is used, see Figure 3. This work considers three different aggregation functions.

The first aggregation function, graphical convolution (GC), sums the scaled edge weight of each neighboring node and multiplies that by learned weights [8]. Graphical convolution is the simplest method for incorporating other nodes' information and, as such, can suffer in performance for complex tasks. The next aggregation function, graph attention (GAT), uses learned weights alongside an attention weight, computed across neighboring nodes' features [4]. The attention enables the GNN to learn what information from other nodes is relevant to the current node. The last aggregation function considered, transformer convolution (TC), incorporates multi-head attention alongside graph convolution [15].

Once information has been aggregated, the current robot's node features are updated. The final actions for the current robot, identical to those of the CNN policy, are predicted using the updated node features. Each robot uses the GNN, with its current relative distances to other robots, obstacles, and tasks, to predict its actions. IPPO was used to train the GNN policy because each robot's state already included information about all other robots, tasks, and obstacles.

4. Experiments

In this section, we describe our simulation and training environment and experimental designs for evaluating the CNN-based and GNN-based policies.

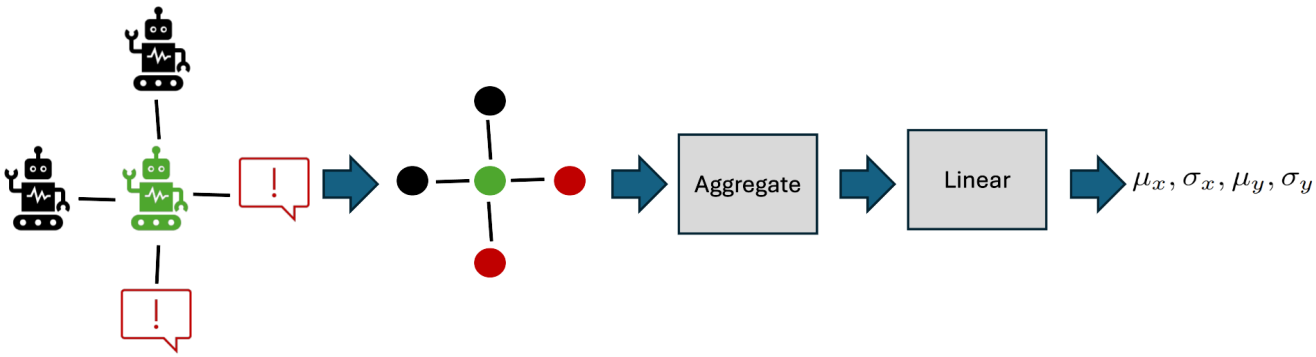

Figure 3. An example of a GNN policy used for predicting actions for the center robot. The GNN aggregates information across robot and task nodes while a linear layer predicts the final actions.

4.1. Training Environment

Our motivating experimental domain considers robots charged with visiting multiple points of interest in an environment cluttered with impassible obstacles to do data collection. Common domains include lunar or marine exploration, where robots are charged with collecting data from numerous regions. In these domains, the best behavior is for robots to distribute themselves throughout the environment, to avoid duplicate data collection visits, which provide no additional reward.

We trained and evaluated our models on a custom Vectorized Multi-Agent Simulator (VMAS) [1] environment using the BenchMARL multi-agent training library [2]. The vectorized nature of VMAS allows efficient parallel simulation of multiple environments, enabling efficient training of MARL policies. The BenchMARL library contains reconfigurable MARL algorithms including MAPPO and IPPO and supports a diverse range of policy models, including CNNs and GNNs. BenchMARL is designed to easily integrate with multi-agent simulators like VMAS.

Figure 4 shows a visual rendering of an example simulation environment. The environment contains randomly-distributed mobile robots (blue circles), task locations (green circles), and obstacles (gray blocks). The environment is designed to allow for a variable number of robots, tasks, and obstacles. At each simulation step, robots collect local observations, pass these observations through their control policies, receive actions to take, execute the actions in the environment, and receive a reward.

The objective of the agents is to reach the waypoints represented by the tasks while avoiding collisions with obstacles. As such, we provide a reward function that rewards a robot for reaching and completing tasks R_{comp}, approaching tasks R_{dist}, and avoiding collisions with obstacles R_{coll}:

$$R = \alpha_1 R_{comp} + \alpha_2 R_{dist} - \alpha_3 R_{coll}, \qquad (2)$$

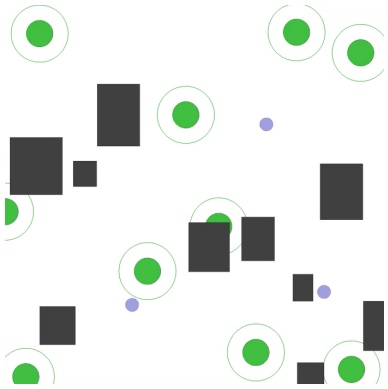

Figure 4. Example simulation environment with 3 robots (blue), 10 tasks (green), and 10 obstacles (black)

where the α reward scaling parameters were tuned via empirical evaluation. We note that R_{comp} is a high-value sparse reward that can only be obtained once per task, and only by the first robot to visit it. R_{dist} and R_{coll} are designed to provide agents with dense reward feedback.

For each experiment, policies are trained for up to 1000 epochs to observe the reward progression.

4.2. CNN Experiments

We conducted experiments to evaluate the CNN model and training parameters for the multi-robot task allocation problem. The primary goal of these experiments was twofold: first, to identify the best-performing RL training parameters for the MAPPO setup of our environment; and second, to evaluate different network configurations in terms of the number of channels and kernel sizes.

To determine the best RL training setup, we focused on three key parameters: minibatch iterations, minibatch size, and learning rate. For this set of experiments, we fixed the CNN architecture with 8 channels, a 3×3 kernel size, and input padding across all three convolutional layers. The first

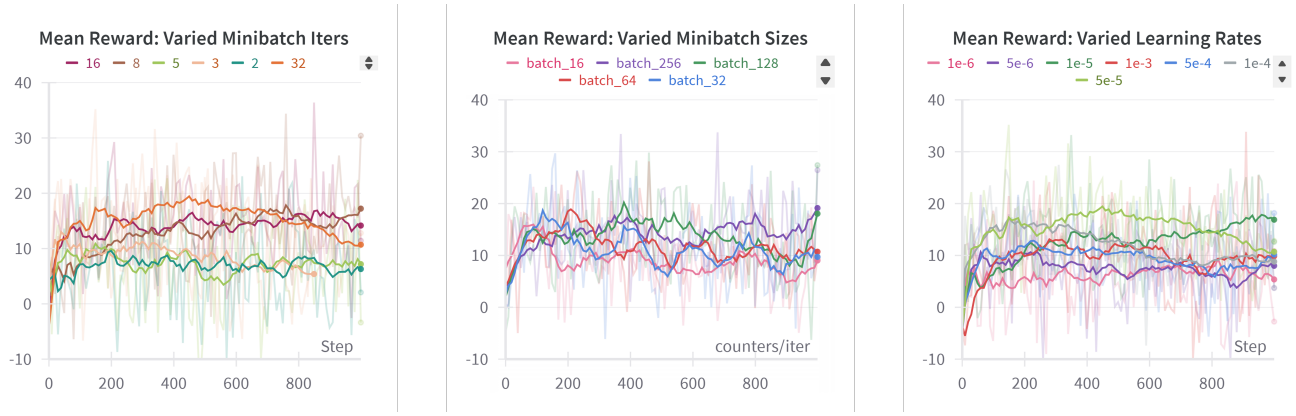

Figure 5. Training performance of the CNN policy under different hyperparameter settings. (Left) Mean episode reward across training steps for varying numbers of minibatch iterations. (Center) Comparison of minibatch sizes and their effect on reward progression. (Right) Performance under different learning rates. Larger minibatch sizes and learning rates around $5e^{-5}$ yielded the most stable and improved learning outcomes.

two layers use a stride of two to downsample the input image. For the minibatch iteration experiments, we fixed the learning rate at $5e^{-5}$ and the minibatch size at 128, varying the number of iterations from 2 to 32. For the minibatch size experiments, we used the same learning rate and 32 minibatch iterations, varying the batch size from 16 to 256 samples. Lastly, for the learning rate experiments, we used 32 minibatch iterations with 128 samples per batch and varied the learning rate from $1e^{-6}$ to $1e^{-3}$.

From these experiments, we identified the optimal RL training parameters as 32 minibatch iterations, a minibatch size of 128, and a learning rate of $5e^{-5}$. With this configuration fixed, we then explored CNN architecture design choices, focusing on kernel size and the number of channels per layer. We tested three kernel sizes of $[3, 5, 7]$, to evaluate the effect of increasing receptive fields. For each kernel size, we experimented with different channel configurations across the three layers. Specifically, we evaluated four network setups: 1) $[8, 8, 8]$, 2) $[16, 16, 16]$, 3) $[16, 16, 8]$, and 4) $[64, 32, 16]$.

4.3. GNN Experiments

We conducted trainings to evaluate GNN aggregation strategies and network architectures. Inheriting the best-performing RL training parameters from the evaluation conducted for the CNN-based policy, we first evaluated our three GNN message passing strategies: GC, GAT, and TC [8, 4, 15].

Next, we selected the best-performing message passing strategy, TC, to evaluate multiple network architectures. Since the initial tests complete only one round of message passing, we sought to provide more opportunities for information exchange by increasing the number of GNN layers. For this test, we first evaluated a policy with four GNN layers with varying dimensions for each layer. For example, a 32-16-8 GNN encodes a 32x1 dimensional node feature vector in the first layer, a 16x1 feature vector in the second layer, a 8x1 feature vector in the third aggregation layer, and a $|A|$x1 feature vector for the last layer, where $|A|$ is the size of the action space, four in our case.

Additionally, we tested a three-layer GNN with a final two-layer MLP to provide the network with additional opportunity to extract important features from the GNN output feature vector. The dimensions for each GNN layer were varied identically to the previous experiment. We held the number of cells in the MLP hidden layer to 8.

5. Results

We performed MARL trainings in our experimental domain and present our results and discussion in the following sections.

5.1. CNN Results

Figure 5 shows the mean reward at evaluation for the RL training parameter experiments, plotted against training epochs. In the left-most plot of Figure 5, we present the results of the minibatch iteration experiments. With fewer iterations, such as 2, 3, and 5, the reward curves remain lower compared to higher iteration counts. This indicates that the policy benefits from a greater number of minibatch iterations. Higher values like 8, 16, and 32 show improved rewards with similar convergence regions. Notably, 32 iterations yield a faster and generally higher reward peak, although the reward tends to dip as training continues. Overall, increasing minibatch iterations up to 32 supports better learning, with trade-offs of training stability and convergence speed.

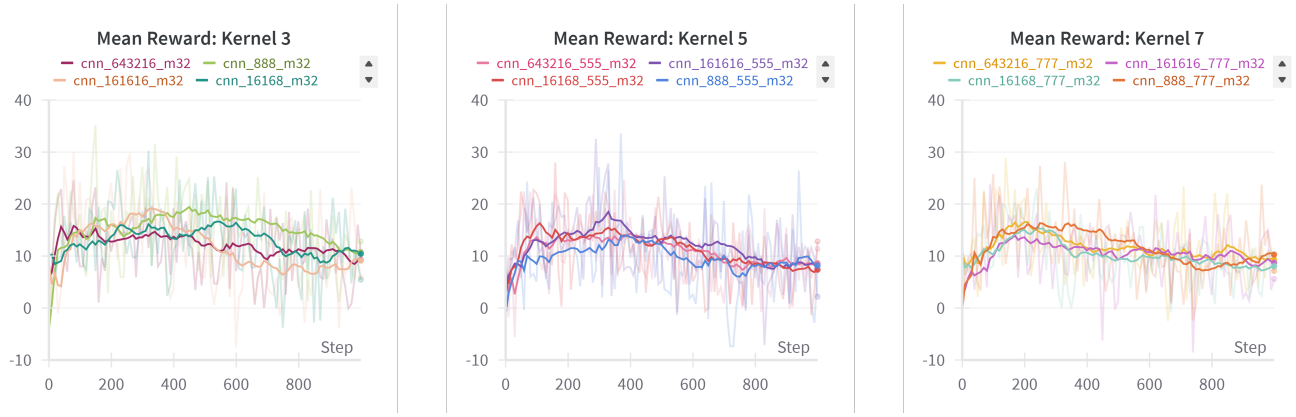

Figure 6. Evaluation of CNN architectural variations with different kernel sizes and channel configurations. (Left) Networks with 3×3 kernels. (Center) Networks with 5×5 kernels. (Right) Networks with 7×7 kernels. All models use three convolutional layers with varying channel depth (e.g., 64-32-16, 16-16-16, 8-8-8, and 16-16-16). Kernel size had a limited impact on performance, with 3×3 kernels slightly outperforming larger kernels in terms of stability and average reward.

The middle plot in Figure 5 shows the results for varying minibatch sizes. Here, the overall reward curves remain close across different batch sizes, with only minor variations. The primary effect is seen in training speed and stability. Smaller batches lead to slightly faster training but introduce more instability, while larger batches stabilize updates at the cost of slower wall-clock training time. We note that a minibatch size of 128 is a balanced choice, offering a good trade-off between training speed and stability.

Finally, the right-most plot in Figure 5 presents results from varying the learning rate. Extremely low or high learning rates result in overall lower rewards and less effective learning. Learning rates of $1e^{-5}$ and $5e^{-5}$ perform best, with $5e^{-5}$ showing faster learning but also a reward drop as training progresses. Based on these results, we select 32 minibatch iterations, a batch size of 128, and a learning rate of $5e^{-5}$ as the RL training parameters for the CNN network experiments.

Figure 6 shows the reward curves for different CNN kernel sizes and channel configurations. Across the three plots, we observe minimal differences in reward curves as the kernel size increases from 3 to 7. Larger kernels, which offer greater receptive fields, slightly reduce the maximum reward achieved, whereas the smaller 3×3 kernel more frequently reaches the peak reward of 30. Within each kernel size group, we compare four network configurations with varying channel setups. No configuration consistently outperforms the others across kernel sizes, with reward values remaining in a similar range across setups. However, the $[8, 8, 8]$ architecture with a 3×3 kernel shows slightly better performance overall and has the added benefit of being lightweight and faster to train.

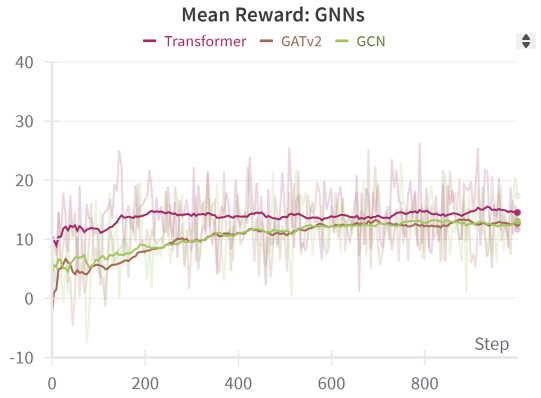

Figure 7. Mean episodic reward evaluated during training for GNN aggregation function (Transformer: TC, GATv2: GAT, GCN: GC) test.

5.2. GNN Results

Figure 7 presents the mean reward obtained from evaluations throughout training for single-layer GNN-based policies using TC, GAT, and GC aggregation methods. While GAT and GC see similar performance, the TC approach consistently outperforms the other two methods, ultimately achieving a mean reward of 14.54 compared to 12.68 and 13.07 for GAT and GC, respectively. With these results, we selected the TC method for message passing in our additional GNN experiments.

Figure 8 shows the mean training rewards gathered for multiple 3-layer GNN configurations. We present the results of these configurations along the baseline single-layer TC results from Figure 7. In this test, the baseline method saw the best performance at 14.54, followed by mean re-

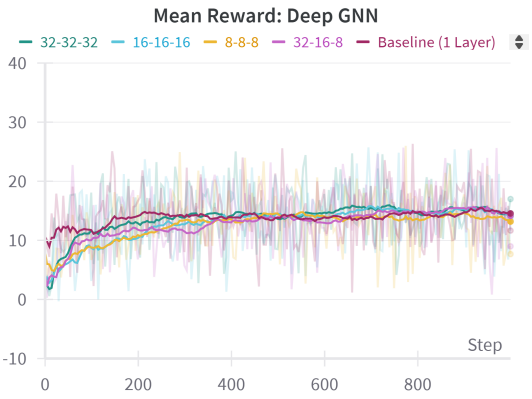

Figure 8. Mean episodic reward evaluated during training for multi-layer GNN node feature test. Labels N1-N2-N3 represent the encoded node feature dimension in each GNN layer.

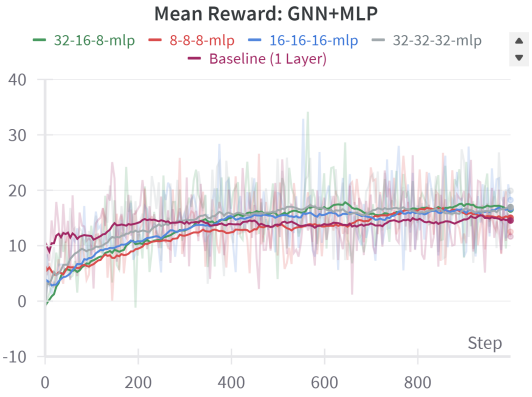

Figure 9. Mean episodic reward evaluated during training for multi-layer GNN with MLP final processing test.

wards of 14.53, 14.16, 13.58, and 13.18 for network dimensions 32-32-32, 32-16-8, 16-16-16, and 8-8-8, respectively. These competitive results indicate that network depth may not be a key performance driver for GNNs in this experimental domain. Rather, richness of node feature encodings may play a larger role, with higher-resolution encodings of 32-32-32 outperforming the 8-8-8 network.

Figure 9 displays the mean training rewards for the GNN+MLP tests. Unlike the previous results, the single-layer baseline performs the worst in terms of mean reward, 14.54, while the best-performing model has a dimension of 32 for all GNN layers, receiving an average reward of 19.91. This result indicates that there is value in a final processing stage after the GNN aggregation steps. Similarly to the trends found in Figure 8, as the number of dimensions increases in the GNN layers, the performance increases.

5.3. Discussion

Comparing the CNN and GNN-based policies, it can be seen that the GNN policy appears to require less fine-tuning to obtain a competitive reward. Additionally, the training curves of the GNNs exhibit a smoother increase, while the CNN policies tend to surge early and decrease with additional training, perhaps due to overfitting. Alternatively, these differences may be due to the fact that the GNN has simpler data to learn from, whereas the CNN must learn to predict actions by processing an entire image. The best models for each respective policy type (the [8,8,8] model for the CNN-based policy and the 32-16-8-MLP model for the GNN-based policy) have similar average rewards, indicating that with tuned parameters, both policies can perform equally well.

While both CNN-based and GNN-based policies show the ability to learn control actions that move robots toward tasks, qualitative analysis of each policy type suggests that robots are not learning to coordinate with each other. We observe a sort of "flocking" behavior when rolling out both policies in which robots that are near each other tend to move toward the same target location. In this scenario, the optimal behavior would be for the agents to move toward different targets to receive a higher R_{comp}.

We identify two potential causes for this suboptimal behavior. First, nearby agents have similar observations, and as such their policies are likely to provide similar action outputs unless they place an adequate amount of weight on potentially subtle differences in their respective observations. Second, the training configuration may have encouraged agents to learn that moving toward a task, even if another agent is also approaching that task, is more valuable than breaking off to go in a different direction. This issue could be addressed with additional reward shaping that encourages agents to spread apart, or by placing even greater value on individual task completions.

6. Conclusions

In this work, we evaluated two different neural network architectures for coordinated multi-robot control. The first architecture used a CNN to predict a robot's action using a local image. We tested various training and CNN parameters to explore what factors affected the mean reward gained by the team. Empirical evidence shows that changing the number of minibatch iterations, batch size, and learning rate had noticeable effects on mean reward, while changing network parameters had little effect. The second architecture used a GNN to predict a robot's action given the robot's position, velocity, and relative distances to other robots, tasks, and obstacles. We found that the GNN using Transformer Convolution as an aggregation function produced the highest mean rewards. Using the Transformer Convo-

lution method, we explored how different dimensions of intermediate GNN layers affected mean reward and found that higher-dimensional layers lead to better performance.

While both methods successfully guided robots to complete tasks, we found that neither the explicit coordination offered by the GNN or the implicit coordination expected of the CNN was achieved. This result indicates that accomplishing both control and coordination is too much to ask of these isolated policy configurations, and that future robotic systems should prioritize the use of specialized policies for specialized roles.

With this in mind, future work will investigate incorporating the CNN and GNN networks into a single policy while isolating each network's role. The CNN could, for example, be used solely for extracting nearby obstacles from image data. The GNN then could handle only agents and tasks, and the outputs of both networks could then be input into a control policy.

References

[1] M. Bettini, R. Kortvelesy, J. Blumenkamp, and A. Prorok. Vmas: A vectorized multi-agent simulator for collective robot learning. In *International Symposium on Distributed Autonomous Robotic Systems*, pages 42–56. Springer, 2022.

[2] M. Bettini, A. Prorok, and V. Moens. Benchmarl: Benchmarking multi-agent reinforcement learning. *Journal of Machine Learning Research*, 25(217):1–10, 2024.

[3] J. Blumenkamp, S. Morad, J. Gielis, Q. Li, and A. Prorok. A framework for real-world multi-robot systems running decentralized gnn-based policies. In *2022 International Conference on Robotics and Automation (ICRA)*, pages 8772–8778. IEEE, 2022.

[4] S. Brody, U. Alon, and E. Yahav. How attentive are graph attention networks?, 2022.

[5] C. S. De Witt, T. Gupta, D. Makoviichuk, V. Makoviychuk, P. H. Torr, M. Sun, and S. Whiteson. Is independent learning all you need in the starcraft multi-agent challenge? *arXiv preprint arXiv:2011.09533*, 2020.

[6] M. Goarin and G. Loianno. Graph neural network for decentralized multi-robot goal assignment. *IEEE Robotics and Automation Letters*, 2024.

[7] J. Heinrich and D. Silver. Deep reinforcement learning from self-play in imperfect-information games, 2016.

[8] T. N. Kipf and M. Welling. Semi-supervised classification with graph convolutional networks, 2017.

[9] M. Konishi, T. Sasaki, and K. Cai. Efficient safe control via deep reinforcement learning and supervisory control–case study on multi-robot warehouse automation. *IFAC-PapersOnLine*, 55(28):16–21, 2022.

[10] Q. Li, F. Gama, A. Ribeiro, and A. Prorok. Graph neural networks for decentralized multi-robot path planning. In *2020 IEEE/RSJ international conference on intelligent robots and systems (IROS)*, pages 11785–11792. IEEE, 2020.

[11] D. Mansfield and A. Montazeri. A survey on autonomous environmental monitoring approaches: towards unifying active sensing and reinforcement learning. *Frontiers in Robotics and AI*, 11:1336612, 2024.

[12] J. Peng, H. Viswanath, K. Tiwari, and A. Bera. Graph-based decentralized task allocation for multi-robot target localization, 2023.

[13] J. P. Queralta, J. Taipalmaa, B. C. Pullinen, V. K. Sarker, T. N. Gia, H. Tenhunen, M. Gabbouj, J. Raitoharju, and T. Westerlund. Collaborative multi-robot search and rescue: Planning, coordination, perception, and active vision. *Ieee Access*, 8:191617–191643, 2020.

[14] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms, 2017.

[15] Y. Shi, Z. Huang, S. Feng, H. Zhong, W. Wang, and Y. Sun. Masked label prediction: Unified message passing model for semi-supervised classification, 2021.

[16] L. Tai and M. Liu. Mobile robots exploration through cnn-based reinforcement learning. *Robotics and biomimetics*, 3:1–8, 2016.

[17] Q. Tan, T. Fan, J. Pan, and D. Manocha. Deepmnavigate: Deep reinforced multi-robot navigation unifying local global collision avoidance, 2020.

[18] C.-Y. Tsai, H. Nisar, and Y.-C. Hu. Mapless lidar navigation control of wheeled mobile robots based on deep imitation learning. *IEEE Access*, 9:117527–117541, 2021.

[19] Z. Wang, C. Liu, and M. Gombolay. Heterogeneous graph attention networks for scalable multi-robot scheduling with temporospatial constraints. *Autonomous Robots*, 46(1):249–268, 2022.

[20] C. Yu, A. Velu, E. Vinitsky, J. Gao, Y. Wang, A. Bayen, and Y. Wu. The surprising effectiveness of ppo in cooperative multi-agent games. *Advances in neural information processing systems*, 35:24611–24624, 2022.