RAILGUN: A Unified Convolutional Policy for Multi-Agent Path Finding Across Different Environments and Tasks

Yimin Tang[1*], Xiao Xiong[2*], Jingyi Xi[2], Jiaoyang Li[3], Erdem Bıyık[1], Sven Koenig[4]

***Abstract*— Multi-Agent Path Finding (MAPF), which focuses on finding collision-free paths for multiple robots, is crucial for applications ranging from aerial swarms to warehouse automation. Solving MAPF is NP-hard so learning-based approaches for MAPF have gained attention, particularly those leveraging deep neural networks. Nonetheless, despite the community's continued efforts, all learning-based MAPF planners still rely on decentralized planning due to variability in the number of agents and map sizes. We have developed the first centralized learning-based policy for MAPF problem called RAILGUN. RAILGUN is not an agent-based policy but a map-based policy. By leveraging a CNN-based architecture, RAILGUN can generalize across different maps and handle any number of agents. We collect trajectories from rule-based methods to train our model in a supervised way. In experiments, RAILGUN outperforms most baseline methods and demonstrates great zero-shot generalization capabilities on various tasks, maps and agent numbers that were not seen in the training dataset.**

I. Introduction

Multi-Agent Path Finding (MAPF) is an NP-hard problem [1], [2] which focuses on finding collision-free paths for multiple agents to move from start locations to their goal locations in a known environment while optimizing a specified cost function. This problem could be adapted to many realistic scenarios from aerial swarms to warehouse automation which are multi-billion dollar industries. Many algorithms have been proposed to solve this problem or its variants, such as Conflict-Based Search (CBS) [3], M^* [4], LaCAM [5] and MAPF-LNS2 [6].

As neural networks demonstrate their powerful capabilities in various fields of computer science [7], [8], [9], learning-based MAPF solvers have also garnered significant attention [10]. Currently, all learning-based MAPF solvers adopt decentralized approaches, where each agent takes surrounding local information as input, typically represented as a field-of-view (FOV). These decentralized policies determine each agent's action, either simultaneously or sequentially, at the current timestep based on the agent's FOV input. Many decentralized methods have been proposed, such as PRIMAL [11], MAPPER [12], MAGAT [13], SCRIMP [14], and MAPF-GPT [15]. These methods primarily rely on imitation learning (IL) and reinforcement learning (RL) and often incorporate additional components, such as inter-agent communication, to enhance performance. It is important to note these approaches focus on individual agents and attempt to generate actions based on agent-specific features, which typically do not include global state information. Furthermore, as features are based on the agent itself, these approaches inherently allow the number of agents to vary.

On the other hand, centralized approaches bring several benefits, such as the ability to coordinate the movements of multiple agents. However, the literature lacks centralized MAPF algorithms that are *learning-based*, since it is challenging to train a centralized neural network that can handle variability in both the number of agents and map sizes.

We present the first centralized learning-based method for MAPF, called RAILGUN, which generates actions based on maps rather than individual agents. The core idea of RAILGUN is to generate a directed graph in which each node has exactly one outgoing edge at every timestep. This design enables our method to handle any number of agents on the map. Additionally, we use a convolutional neural network (CNN) as the model backbone which produces outputs of the same dimensions as the input features. This allows RAILGUN to accommodate maps of varying sizes. In summary, our contributions are as follows:

- We propose the first centralized learning-based MAPF algorithm, RAILGUN, which generates actions for map grid cells rather than for individual agents.
- We design a CNN-based network enabling RAILGUN to handle maps of different sizes.
- Through experiments in diverse test settings, we demonstrate that RAILGUN, trained on data from one map type, generalizes effectively to new types of maps and testing scenarios, and outperforms most baseline methods in POGEMA [16] benchmark.

II. Problem Definition

The MAPF problem is defined as follows: Let $I = \{1, 2, \cdots, N\}$ denote a set of N agents. $G = (V, E)$ represents an undirected graph, where each vertex $v \in V$ represents a possible location of an agent in the workspace, and each edge $e \in E$ is a unit-cost edge between two vertices that moves an agent from one vertex to the other. In this paper, we focus on 2D grid maps with connections in four directions. Self-loop edges are also allowed, which represent "wait-in-place" actions. Each agent $i \in I$ has a start location $s_i \in V$ and a goal location $g_i \in V$. It also holds that $s_i \neq s_j$ and $g_i \neq g_j$ when $i \neq j \ \forall i, j \in I$. Our task is to plan a collision-free path for each agent i from s_i to g_i.

*Equal contribution

[1]Thomas Lord Department of Computer Science, University of Southern California, yimintan@usc.edu, biyik@usc.edu

[2]Independent Researcher, xiaoxiong.xx21@gmail.com, flotherxi@gmail.com

[3]Carnegie Mellon University, jiaoyanl@andrew.cmu.edu

[4]University of California, Irvine, sven.koenig@uci.edu

Each action of agents, either waiting in place or moving to an adjacent vertex, takes one time unit. Let $v_t^i \in V$ be the location of agent i at timestep t. Let $\pi_i = [v_0^i, v_1^i, ..., v_{T^i}^i]$ denote a path of agent i from its start location v_0^i to its target $v_{T^i}^i$. We assume that agents rest at their targets after completing their paths, i.e., $v_t^i = v_{T^i}^i, \forall t > T^i$. The cost of agent i's path is T^i. We refer to the path with the minimum cost as the shortest path.

We consider two types of agent-agent collisions. The first type is *vertex collision*, where two agents i and j occupy the same vertex at the same timestep. The second type is *edge collision*, where two agents move in opposite directions along the same edge simultaneously. We use (i, j, t) to denote a vertex collision between agents i and j at timestep t or an edge collision between agents i and j at timestep t to $t+1$. The requirement of being collision-free implies the targets assigned to the agents must be distinct from each other. We use *SoC (flowtime)* $\sum_{i=1}^{N} T^i$ as the cost function.

The objective of the MAPF problem is to find a set of paths $\{\pi_i \mid i \in I\}$ for all agents such that, for each agent i:

1) Agent i starts from its start location (i.e., $v_0^i = s_i$) and stops at its target location g_j (i.e., $v_t^i = g_j, \forall t \geq T^i$).
2) Every pair of adjacent vertices on path π_i is connected by an edge, i.e., $(v_t^i, v_{t+1}^i) \in E, \forall t \in \{0, 1, \ldots, T^i\}$.
3) $\{\pi_i \mid i \in I\}$ is collision-free.

III. Related Work

A. Multi-Agent Path Finding (MAPF)

MAPF has been proved an NP-hard problem with optimality [2]. It has inspired a wide range of solutions for its related challenges. Decoupled strategies, as outlined in [17], [18], [19], approach the problem by independently planning paths for each agent before integrating these paths. In contrast, coupled approaches [20], [21] devise a unified plan for all agents simultaneously. There also exist dynamically coupled methods [3], [22] that consider agents planning independently at first and then together only when needed for resolving agent-agent collisions. Among these, Conflict-Based Search (CBS) algorithm [3] stands out as a centralized and optimal method for MAPF, with several bounded-suboptimal variants such as ECBS [23] and EECBS [24]. Some suboptimal MAPF algorithms, such as Prioritized Planning (PP) [25], [17], PBS [26], LaCAM [5] and their variant methods [27], [6], [28] exhibit better scalability and efficiency. However, these search-based algorithms always face the problem of search space dimensionality explosion as the problem size increases, making it difficult to produce a valid solution within a limited time. Learning-based methods can overcome the dimensionality issue by learning from large amounts of data and addressing the trade-off between low-cost paths and scalability.

B. Lifelong MAPF

Compared to the MAPF problem, Lifelong MAPF (LMAPF) continuously assigns new target locations to agents once they have reached their current targets. In LMAPF, agents do not need to arrive at their targets simultaneously. There are three main approaches to solving LMAPF: solving the problem as a whole [29], using MAPF methods but replanning all paths at each specified timestep [30], [31], and replanning only when agents reach their current targets and are assigned new ones [32], [33]. Some algorithms consider the offline setting in LMAPF, where all tasks are known in advance. Examples include CBSS [34], which applies Traveling Salesman Problem (TSP) methods to plan task orders, and a four-level hierarchical planning algorithm [35] that incorporates MILP and CBS. However, these LMAPF methods also face the same scalability problem as MAPF methods.

C. Learning-based MAPF

Given the huge success of deep learning, many learning-based MAPF methods have been proposed. Compared to search-based algorithms, these methods can usually complete planning in short time and automatically learn heuristic functions. Some of these methods focus on modifying edge weights in the map, such as the congestion model [36], which is a data-driven approach that predicts agents' movement delays and uses these delays as movement costs, or Online GGO [37], which optimizes edge weights for Lifelong MAPF. However, these methods split MAPF planning into multiple stages, which can lead to a larger optimization search space if one considers both edge-weight design and the MAPF solver simultaneously.

Most other methods focus on the solver side, using imitation learning (IL), reinforcement learning (RL), or both. One early learning-based method for MAPF is PRIMAL [11] which is trained by RL and IL. It is a decentralized algorithm that relies on an FOV around an agent to generate the actions of that agent. MAPF-GPT [15] is a GPT-based model for MAPF problems, trained by IL on a large dataset. Other approaches incorporate communication mechanisms in a decentralized manner, such as GNN [38] and MAGAT [13], which employ Graph Neural Networks (GNNs) for communication, and SCRIMP [14], which uses a global communication mechanism based on transformers.

However, all existing learning-based solvers focus on the agents themselves, forcing researchers to design features of agents. This makes it challenging, if not impossible, to develop a centralized policy that can handle varying numbers of agents and map sizes. Our method is the first centralized MAPF solver to overcome the challenge of feature design and to integrate edge-weight design ideas [39], [40] into a neural-network-based solver.

IV. Method

In this section, we introduce our RAILGUN method. First, we discuss why it is difficult to design a learning algorithm for centralized MAPF where policies are agent-based. When focusing on generating actions based on agent features, we need to provide a neural network with at least the agent's start location, goal location, and additional features, amounting to k scalar variables ($k \geq 4$) for one agent. Then the total number of features is at least kN. Consider that the

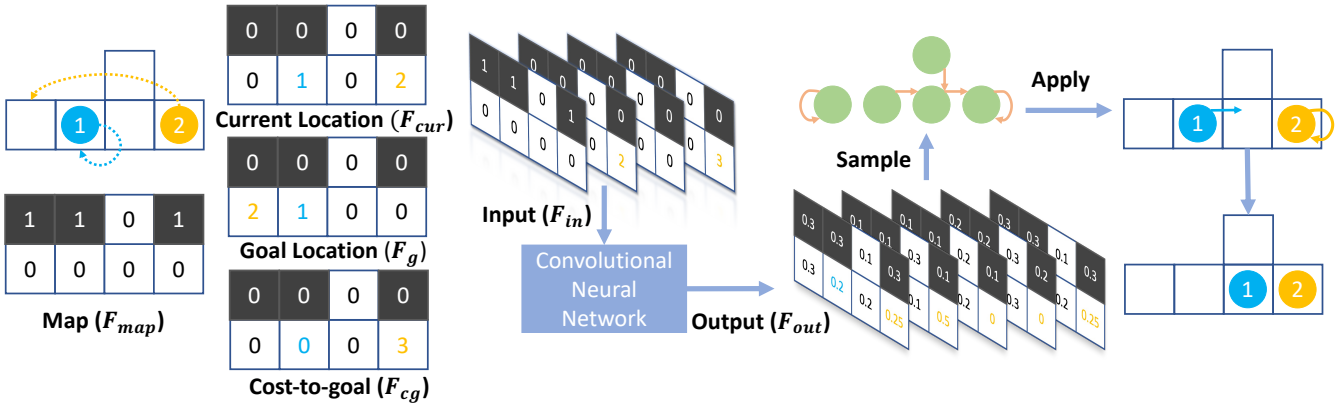

Fig. 1: RAILGUN Inference Overview: On the left side, there is one current state along with all related input features of size $(n, m, 1)$. These features are then stacked along the last channel to construct the input feature F_{in} of size (n, m, k). In this example, we have $n = 2$, $m = 4$, and $k = 4$. On the right side, the input feature F_{in} is fed into a CNN-based neural network, which outputs action probabilities F_{out} of size $(n, m, 5)$. We sample from F_{out} to obtain actual actions and then apply the corresponding actions to each agent.

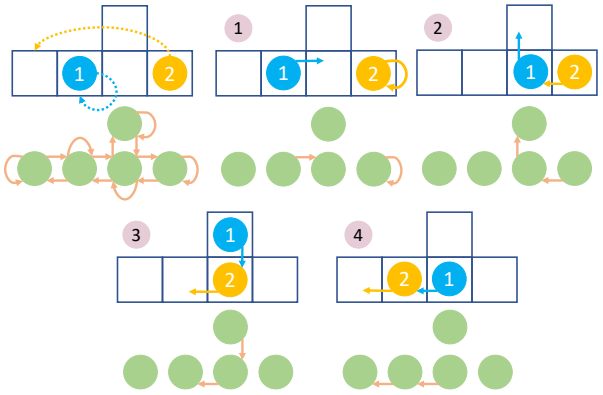

Fig. 2: This is an example of how an agent-based solution relates to a series of specialized graphs. The upper-left figure illustrates a testcase we aim to solve, along with a graph where green nodes and orange edges represent map connectivity. The other figures show a valid MAPF solution for this testcase, where agent 1 should yield to agent 2. At each timestep, each node in the connectivity graph has only one outgoing edge. Here, we draw edges only for nodes occupied by agents, as the outgoing edge for other nodes could be any of the available edges.

maximum number of agents could be $N = |V| \approx nm$, where n and m are the 2D map dimensions. If we want to handle all possible numbers of agents on a specific map, the total feature size would be knm. This dependence on map size means that we cannot create a policy to cover all different maps if we construct the features agent by agent. That is why there is no centralized learning-based solver and all learning-based MAPF solvers adopt a decentralized approach with a limited FOV for each agent [11], [12], [13], [14], [15].

Our insight is that in a valid MAPF solution, there will be no collision, which means there can be at most one agent in each map grid cell in each timestep. At any timestep, each agent chooses one of the five edges of its grid cell as its action. Therefore, if we remove all edges that the agents do not use at each timestep, we find that a valid MAPF solution can be viewed as a series of specialized graphs. As shown in Figure 2, these specialized graphs have exactly one edge in every occupied grid cell. Once such a directed graph is given, no MAPF solver is needed, as there is only one possible transition at each timestep. The sequence of these specialized graphs then constitutes a valid MAPF solution.

After converting the agent-based solution into a representation as a series of specialized graphs, we use a CNN network to address the challenge of generating these specialized graphs and generalizing across different maps, which we discussed in the previous paragraphs. In this paper, we use standard U-Net architecture [41] for RAILGUN, where the input feature is F_{in} with size (n, m, k) and the output feature is F_{out} with size $(n, m, 5)$. Here, k represents the number of feature channels based on the feature design, and (n, m) represents the map size. As for network structure and feature selection, please refer to Appendix.

As an example shown in Figure 1, to encode an agent's current location as a feature, we construct a tensor F_{cur} with size $(n, m, 1)$. In this tensor, $F_{cur}[i][j] = idx$ if the agent idx is at position (i, j) in the map; otherwise, $F_{cur}[i][j] = 0$. Stacking all such feature tensors along the last dimension forms F_{in} with size (n, m, k). $F_{out}[i][j]$ represents the probability distribution over all possible actions at grid cell position (i, j). We use 5 channels because each agent can take one of up to five different actions at each timestep. Thus, if an agent is located at grid cell (i, j), its action probabilities are stored in $F_{out}[i][j]$.

V. Experiments & Results

A. Training and Testing Settings

We use POGEMA [16] benchmark to test our method. POGEMA includes several different metrics, allowing a fair multi-fold comparison. For data collection, our training data is primarily generated by LaCAM-v1 [5]. The model is trained with cross-entropy loss and a batch size of 256. We utilize the AdamW optimizer [42] with β values set to $(0.9, 0.999)$ and a weight decay of 10^{-3}. The training process achieves convergence in only six hours, leveraging the power of four NVIDIA A100 GPUs.

For training data, we randomly generate 180 maze maps with 32×32 size, each with varying obstacle densities and maze shape. For each map, we randomly generate {2, 5, 20,

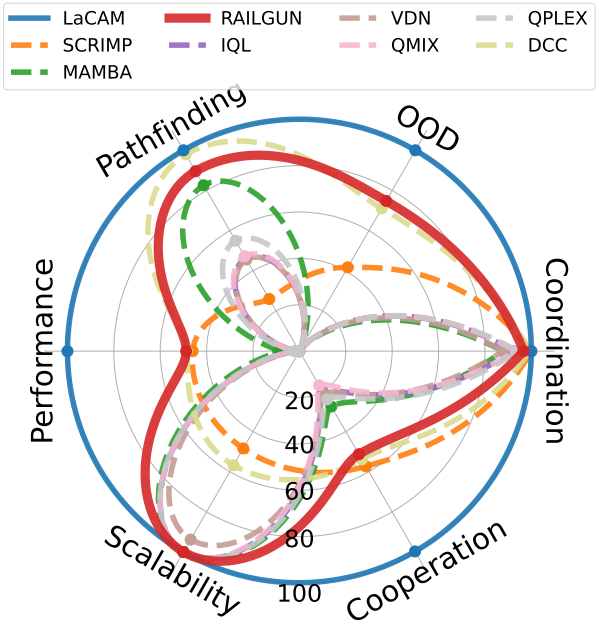

Fig. 3: MAPF: RAILGUN outperforms most baseline models in Scalability, Performance, Pathfinding, OOD and Coordination. Performance, OOD, Pathfinding and Cooperation represents solution SoC/throuput quality. Scalability represents runtime respect to agent numbers. Coordination is the probability of invalid actions from learning-based methods.

40, 60} MAPF scenarios with {16, 32, 64, 96, 128} agents respectively, for a total of 127 scenarios for each map. We use LACAM-v1 [5] to compute the reference paths for all scenarios as the training data.

All experiments were conducted on a system running Ubuntu 22.04.1 LTS equipped with an AMD Intel i9-12900K CPU, 128GB RAM and NVIDIA RTX 3080. For the testing phase, the POGEMA benchmark provides a total of 3,376 test cases featuring six different types of maps shown in Figure 5, varying numbers of agents, and different map sizes.

B. Testing Results

Figure 3 presents the performance metrics for RAILGUN and the baseline methods. The learning-based methods include VDN [43], QPLEX [44], SCRIMP [14], IQL [45], QMIX [46], DCC [47], MAMBA [48], Switcher [49], Follower [50], and MATS-LP [51]. All these baseline methods are decentralized methods. LaCAM-v3 [28] and RHCR [52] serve as the search-based algorithm baselines in MAPF and LMAPF problems.

In Figure 3, we observe that RAILGUN achieves high scores across all six metrics and delivers the best performance in four metrics compared to other learning-based methods. RAILGUN attains the highest score in the Scalability metric because it generates specific directed graphs at each timestep, ensuring that runtime depends only on map size rather than the number of agents in theory. However, even though RAILGUN outperforms or matches the scores of other learning-based methods in most areas, it still exhibits a significant gap with LaCAM in SoC-related metrics.

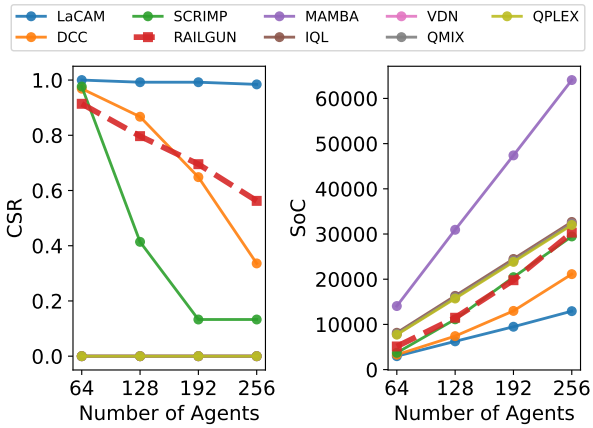

Fig. 4: MAPF testing on Cities-tiles: CSR (the success rate at which all agents reach their goal locations; higher is better), SoC (Sum of all agent arrival time; lower is better).

This outcome is expected, as RAILGUN is trained on data generated by LaCAM-v1, and LaCAM-v1 is not designed to achieve the best SoC performance. Mimicking LaCAM-v1 is the top priority of RAILGUN rather than producing a valid solution with the lowest SoC. RAILGUN also achieves good results in LMAPF task which is a zero-shot task for RAILGUN shown in Appendix.

Figure 4 presents detailed CSR (see caption) and SoC. Even for unseen maps (Cities-tiles) and larger agent numbers (192 and 256), RAILGUN outperforms other learning-based methods, achieving up to 60% CSR. This also shows RAILGUN's strong zero-shot generalization ability in new maps and new agent numbers. Furthermore, we observe that DCC attains a better SoC, despite having a lower CSR. This indicates that in DCC, only a few agents fail to reach their goal locations and the path lengths are shorter than those produced by RAILGUN, highlighting that generating valid solutions is a higher priority for RAILGUN.

VI. Conclusion and Future Work

In this paper, we propose the first centralized learning-based method, RAILGUN, for the MAPF problem. We found that, rather than predicting actions for individual agents, predicting edge directions for each map grid cell overcomes the difficulties associated with variable input feature dimensions. This finding allows RAILGUN to employ a CNN-based architecture capable of handling maps of any size and any number of agents. In our experiments, RAILGUN demonstrates strong performance across all six metrics in the POGEMA benchmark. Furthermore, its excellent generalization abilities enable it to handle unseen maps, varying agent numbers, and even other tasks such as the LMAPF problem. In future work, we plan to collect higher-quality data to train RAILGUN as a foundation model and apply RL with a task-specific cost function to fine-tune RAILGUN on specific tasks, agent numbers, and map shapes, thereby improving solution quality and success rate in real-world applications.

References

[1] R. Stern, N. Sturtevant, A. Felner, S. Koenig, H. Ma, T. Walker, J. Li, D. Atzmon, L. Cohen, T. Kumar *et al.*, "Multi-agent pathfinding: Definitions, variants, and benchmarks," in *Proceedings of the International Symposium on Combinatorial Search (SoCS)*, vol. 10, no. 1, 2019.

[2] J. Yu and S. LaValle, "Structure and intractability of optimal multi-robot path planning on graphs," in *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, vol. 27, no. 1, 2013, pp. 1443–1449.

[3] G. Sharon, R. Stern, A. Felner, and N. R. Sturtevant, "Conflict-based search for optimal multi-agent pathfinding," *Artificial Intelligence*, vol. 219, pp. 40–66, 2015.

[4] G. Wagner and H. Choset, "M*: A complete multirobot path planning algorithm with performance bounds," in *2011 IEEE/RSJ international conference on intelligent robots and systems*. IEEE, 2011.

[5] K. Okumura, "Lacam: Search-based algorithm for quick multi-agent pathfinding," in *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, vol. 37, no. 10, 2023, pp. 11 655–11 662.

[6] J. Li, Z. Chen, D. Harabor, P. J. Stuckey, and S. Koenig, "Mapf-lns2: fast repairing for multi-agent path finding via large neighborhood search," in *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, vol. 36, no. 9, 2022, pp. 10 256–10 265.

[7] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 25, 2012.

[8] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot *et al.*, "Mastering the game of go with deep neural networks and tree search," *nature*, vol. 529, no. 7587, 2016.

[9] J. Achiam, S. Adler, S. Agarwal, L. Ahmad, I. Akkaya, F. L. Aleman, D. Almeida, J. Altenschmidt, S. Altman, S. Anadkat *et al.*, "Gpt-4 technical report," *arXiv preprint arXiv:2303.08774*, 2023.

[10] J.-M. Alkazzi and K. Okumura, "A comprehensive review on leveraging machine learning for multi-agent path finding," *IEEE Access*, 2024.

[11] G. Sartoretti, J. Kerr, Y. Shi, G. Wagner, T. S. Kumar, S. Koenig, and H. Choset, "Primal: Pathfinding via reinforcement and imitation multi-agent learning," *IEEE Robotics and Automation Letters*, vol. 4, no. 3, pp. 2378–2385, 2019.

[12] Z. Liu, B. Chen, H. Zhou, G. Koushik, M. Hebert, and D. Zhao, "Mapper: Multi-agent path planning with evolutionary reinforcement learning in mixed dynamic environments," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2020, pp. 11 748–11 754.

[13] Q. Li, W. Lin, Z. Liu, and A. Prorok, "Message-aware graph attention networks for large-scale multi-robot path planning," *IEEE Robotics and Automation Letters*, vol. 6, no. 3, pp. 5533–5540, 2021.

[14] Y. Wang, B. Xiang, S. Huang, and G. Sartoretti, "Scrimp: Scalable communication for reinforcement-and imitation-learning-based multi-agent pathfinding," in *2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2023, pp. 9301–9308.

[15] A. Andreychuk, K. Yakovlev, A. Panov, and A. Skrynnik, "Mapf-gpt: Imitation learning for multi-agent pathfinding at scale," *arXiv preprint arXiv:2409.00134*, 2024.

[16] A. Skrynnik, A. Andreychuk, A. Borzilov, A. Chernyavskiy, K. Yakovlev, and A. Panov, "Pogema: A benchmark platform for cooperative multi-agent navigation," 2024. [Online]. Available: https://arxiv.org/abs/2407.14931

[17] D. Silver, "Cooperative pathfinding," in *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE)*, vol. 1, no. 1, 2005, pp. 117–122.

[18] K.-H. C. Wang and A. Botea, "Fast and memory-efficient multi-agent pathfinding," in *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*, 2008, pp. 380–387.

[19] R. J. Luna and K. E. Bekris, "Push and swap: Fast cooperative path-finding with completeness guarantees," in *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, 2011.

[20] T. Standley, "Finding optimal solutions to cooperative pathfinding problems," in *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, vol. 24, no. 1, 2010, pp. 173–178.

[21] T. Standley and R. Korf, "Complete algorithms for cooperative pathfinding problems," in *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, 2011, pp. 668–673.

[22] G. Wagner and H. Choset, "Subdimensional expansion for multirobot path planning," *Artificial intelligence*, vol. 219, pp. 1–24, 2015.

[23] M. Barer, G. Sharon, R. Stern, and A. Felner, "Suboptimal variants of the conflict-based search algorithm for the multi-agent pathfinding problem," in *Proceedings of the International Symposium on Combinatorial Search (SoCS)*, 2014.

[24] J. Li, W. Ruml, and S. Koenig, "Eecbs: A bounded-suboptimal search for multi-agent path finding," in *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, vol. 35, no. 14, 2021.

[25] M. Erdmann and T. Lozano-Perez, "On multiple moving objects," *Algorithmica*, vol. 2, pp. 477–521, 1987.

[26] H. Ma, D. Harabor, P. J. Stuckey, J. Li, and S. Koenig, "Searching with consistent prioritization for multi-agent path finding," in *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, vol. 33, no. 01, 2019, pp. 7643–7650.

[27] S.-H. Chan, R. Stern, A. Felner, and S. Koenig, "Greedy priority-based search for suboptimal multi-agent path finding," in *Proceedings of the International Symposium on Combinatorial Search (SoCS)*, vol. 16, no. 1, 2023, pp. 11–19.

[28] K. Okumura, "Engineering lacam*: Towards real-time, large-scale, and near-optimal multi-agent pathfinding," *arXiv preprint*, 2023.

[29] V. Nguyen, P. Obermeier, T. Son, T. Schaub, and W. Yeoh, "Generalized target assignment and path finding using answer set programming," in *Proceedings of the International Symposium on Combinatorial Search (SoCS)*, vol. 10, no. 1, 2019, pp. 194–195.

[30] J. Li, A. Tinka, S. Kiesel, J. W. Durham, T. S. Kumar, and S. Koenig, "Lifelong multi-agent path finding in large-scale warehouses," in *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, vol. 35, no. 13, 2021, pp. 11 272–11 281.

[31] K. Okumura, M. Machida, X. Défago, and Y. Tamura, "Priority inheritance with backtracking for iterative multi-agent path finding," *Artificial Intelligence*, vol. 310, p. 103752, 2022.

[32] M. Čáp, J. Vokřínek, and A. Kleiner, "Complete decentralized method for on-line multi-robot trajectory planning in well-formed infrastructures," in *Proceedings of the International Conference on Auto- mated Planning and Scheduling (ICAPS)*, vol. 25, 2015, pp. 324–332.

[33] F. Grenouilleau, W.-J. Van Hoeve, and J. N. Hooker, "A multi-label a* algorithm for multi-agent pathfinding," in *Proceedings of the International Conference on Auto- mated Planning and Scheduling (ICAPS)*, vol. 29, 2019, pp. 181–185.

[34] Z. Ren, S. Rathinam, and H. Choset, "CBSS: A new approach for multiagent combinatorial path finding," *IEEE Transactions on Robotics*, vol. 39, no. 4, pp. 2669–2683, 2023.

[35] K. Brown, O. Peltzer, M. A. Sehr, M. Schwager, and M. J. Kochenderfer, "Optimal sequential task assignment and path finding for multi-agent robotic assembly planning," in *Proceedings of IEEE International Conference on Robotics and Automation (ICRA)*, 2020.

[36] G. Yu and M. T. Wolf, "Congestion prediction for large fleets of mobile robots," in *Proceedings of IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2023, pp. 7642–7649.

[37] H. Zang, Y. Zhang, H. Jiang, Z. Chen, D. Harabor, P. J. Stuckey, and J. Li, "Online guidance graph optimization for lifelong multi-agent path finding," in *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, 2025.

[38] 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)*. IEEE, 2020, pp. 11 785–11 792.

[39] Y. Zhang, H. Jiang, V. Bhatt, S. Nikolaidis, and J. Li, "Guidance graph optimization for lifelong multi-agent path finding," *arXiv preprint arXiv:2402.01446*, 2024.

[40] Z. Chen, D. Harabor, J. Li, and P. J. Stuckey, "Traffic flow optimisation for lifelong multi-agent path finding," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 38, no. 18, 2024.

[41] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," in *Medical image computing and computer-assisted intervention–MICCAI 2015: 18th international conference, Munich, Germany, October 5-9, 2015, proceedings, part III 18*. Springer, 2015, pp. 234–241.

[42] I. Loshchilov and F. Hutter, "Decoupled weight decay regularization," *arXiv preprint arXiv:1711.05101*, 2017.

[43] P. Sunehag, G. Lever, A. Gruslys, W. M. Czarnecki, V. Zambaldi, M. Jaderberg, M. Lanctot, N. Sonnerat, J. Z. Leibo, K. Tuyls *et al.*, "Value-decomposition networks for cooperative multi-agent learning based on team reward," in *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 2018.

[44] J. Wang, Z. Ren, T. Liu, Y. Yu, and C. Zhang, "Qplex: Duplex dueling multi-agent q-learning," in *International Conference on Learning Representations*, 2020.

[45] M. Tan, "Multi-agent reinforcement learning: Independent vs. cooperative agents," in *Proceedings of the tenth international conference on machine learning*, 1993, pp. 330–337.

[46] T. Rashid, M. Samvelyan, C. S. De Witt, G. Farquhar, J. Foerster, and S. Whiteson, "Monotonic value function factorisation for deep multi-agent reinforcement learning," *Journal of Machine Learning Research*, vol. 21, no. 178, pp. 1–51, 2020.

[47] Z. Ma, Y. Luo, and J. Pan, "Learning selective communication for multi-agent path finding," *IEEE Robotics and Automation Letters*, vol. 7, no. 2, pp. 1455–1462, 2021.

[48] V. Egorov and A. Shpilman, "Scalable multi-agent model-based reinforcement learning," in *Proceedings of the 21st International Conference on Autonomous Agents and Multiagent Systems*, 2022.

[49] A. Skrynnik, A. Andreychuk, K. Yakovlev, and A. I. Panov, "When to switch: planning and learning for partially observable multi-agent pathfinding," *IEEE Transactions on Neural Networks and Learning Systems*, 2023.

[50] A. Skrynnik, A. Andreychuk, M. Nesterova, K. Yakovlev, and A. Panov, "Learn to follow: Decentralized lifelong multi-agent pathfinding via planning and learning," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 38, 2024, pp. 17 541–17 549.

[51] A. Skrynnik, A. Andreychuk, K. Yakovlev, and A. Panov, "Decentralized monte carlo tree search for partially observable multi-agent pathfinding," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 38, 2024, pp. 17 531–17 540.

[52] J. Li, A. Tinka, S. Kiesel, J. W. Durham, T. K. S. Kumar, and S. Koenig, "Lifelong multi-agent path finding in large-scale warehouses," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 35, no. 13, pp. 11 272–11 281, May 2021. [Online]. Available: https://ojs.aaai.org/index.php/AAAI/article/view/17344

VII. Appendix

A. Model Architecture

We use U-Net as our model backbone mainly due to its success in diffusion models and its ability to output feature maps that are the same size as inputs. We employ the standard U-Net architecture, which consists of a total of five layers. The encoder begins with an initial layer containing 64 channels. At each subsequent layer, the number of channels is doubled while the size of the feature maps is halved. In the decoder, this process is reversed, with the number of channels halved and the spatial resolution doubled at each layer. Notably, bilinear interpolation is not employed in the decoder; instead, we use deconvolution as in original U-Net. At the final layer, the number of channels is reduced to 5, corresponding to the maximum number of possible actions. We should also note that since U-Net uses CNN layers in the encoder, which progressively reduce the spatial dimensions of the feature maps, there is a minimum required input size to ensure valid downsampling operations. For small maps, padding is needed. The resulting RAILGUN model contains approximately 30 million FP32 parameters.

B. Feature Selection

As shown in Figure 1, we construct the input features from multiple components. We employ five types of features: the map, current locations, goal locations, cost-to-goal, and gradients of cost-to-goal (the last feature is not shown in Figure 1 due to space constraints). For the map feature, we use 1 to represent non-traversable grid cells and 0 to represent traversable grid cells. For the current and goal locations, we use the agent's index to indicate which agent occupies a grid cell; otherwise, the grid cell is set to 0. We also attempted encoding agent indices as binary vectors; however, this produces excessively large input features, rendering the model too large to train.

We also use the precomputed shortest path cost as the cost-to-goal feature for each agent which is a widely used feature in learning-based methods, as shown in Figure 1. The gradients of the cost-to-goal, represents the potential direction of next action, are determined by the changes of the cost-to-goal distances. Specifically, we define the changes in cost-to-goal distances from an agent's current cell (i, j) to its adjacent cells as $\delta_{\text{left}}, \delta_{\text{right}}, \delta_{\text{up}}, \delta_{\text{down}}$. $\delta < 0$ indicates that the agent is approaching the goal location; vice versa. The resulting direction, denoted by $\mathbf{g}_{ij} = (\Delta x_{ij}, \Delta y_{ij})$, consists of horizontal and vertical components. For the horizontal component, Δx_{ij} is computed as shown below:

$$\Delta x_{ij} = \begin{cases} 0 & \text{if } \delta_{\text{left}} \geq 0 \text{ and } \delta_{\text{right}} \geq 0, \\ 1 & \text{if } \delta_{\text{left}} \geq 0 \text{ and } \delta_{\text{right}} < 0, \\ -1 & \text{if } \delta_{\text{left}} < 0 \text{ and } \delta_{\text{right}} \geq 0, \\ \text{random}(\pm 1) & \text{if } \delta_{\text{left}} < 0 \text{ and } \delta_{\text{right}} < 0, \end{cases}$$

and similarly for the vertical component.

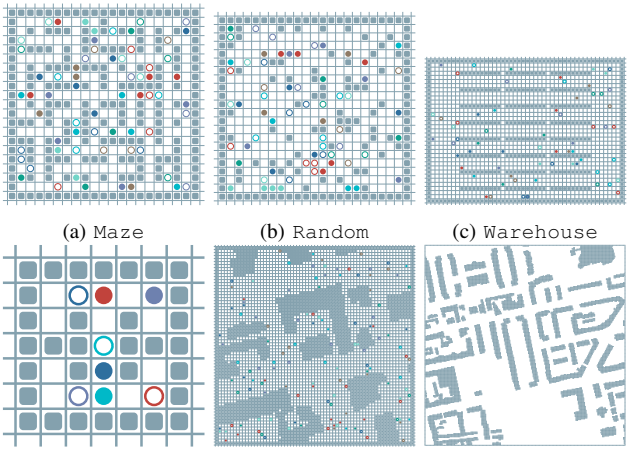

Fig. 5: Examples of POGEMA-tested maps. The six metrics—*Performance*, *Coordination*, *Scalability*, *Cooperation*, *OOD*, and *Pathfinding*—are evaluated on the following map sets: Maze, Random, Maze, Random, Warehouse, Puzzle, Cities-tiles, and Cities. Note that Cities-tiles are 64×64 areas selected from larger Cities maps with dimensions of 256×256.

C. POGEMA Test

POGEMA use six metrics, namely, *Performance*, *Coordination*, *Scalability*, *Coopeartion*, *Out-of-Distribution (OOD)* and *Pathfinding*. The relevant equations are as follows:

$$\text{Performance} = \begin{cases} SoC_{best}/SoC & \text{if MAPF solved} \\ 0 & \text{if MAPF not solved} \\ \frac{\text{throughput}}{\text{throughput}_{best}} & \text{if LMAPF} \end{cases}$$

$$\text{OOD/Cooperation} = \begin{cases} SoC_{best}/SoC & \text{if MAPF solved} \\ 0 & \text{if MAPF not solved} \\ \frac{\text{throughput}}{\text{throughput}_{best}} & \text{if LMAPF} \end{cases}$$

Performance, *OOD* and *Cooperation* metrics primarily evaluate solution quality and success rate on different maps. SoC_{best} represents the best SoC performance achieved among all tested algorithms.

$$\text{Scalability} = \frac{\text{runtime}(\text{agents}_1)/\text{runtime}(\text{agents}_2)}{|\text{agents}_1|/|\text{agents}_2|}$$

$$\text{Coordination} = 1 - \frac{\text{\# of collisions}}{|\text{agents}| \times \text{episode\_length}}$$

$$\text{Pathfinding} = \begin{cases} \text{SoC}/\text{SoC}_{best} \\ 0 \text{ if path not found} \end{cases}$$

Scalability is the ratio of algorithm runtimes with different agent numbers with $|\text{agent}_1| < |\text{agent}_2|$, providing a measure of how the algorithm's runtime scales as the agent number changes and higher is better. *Coordination* focuses on invalid action frequency produced by learning-based methods. *Pathfinding* indicates the ability of learning-based methods to find the shortest path for a single agent.

D. Additional Testing Results

In Table I, we present the CSR, SoC, and Makespan metrics (see caption) of different algorithms for the Warehouse map. We observe a similar pattern where RAILGUN

Algorithm \ Agents	CSR						SoC (x1000)						Makespan					
	32	64	96	128	160	192	32	64	96	128	160	192	32	64	96	128	160	192
LaCAM	1.00	1.00	1.00	1.00	1.00	1.00	0.98	1.97	3.00	4.07	5.16	6.32	55.34	58.50	60.50	61.59	62.77	64.04
RAILGUN	**1.00**	**0.98**	**0.75**	**0.23**	**0.02**	-	1.16	2.79	5.33	8.93	12.77	17.18	**60.32**	**76.34**	**105.36**	**124.14**	**127.70**	-
DCC	0.95	0.86	0.73	0.12	-	-	**1.10**	**2.62**	**4.88**	**7.82**	**11.08**	**14.85**	66.06	88.09	111.23	126.75	-	-
SCRIMP	0.83	0.17	-	-	-	-	1.31	3.24	6.97	11.96	16.17	20.32	85.48	122.55	-	-	-	-
MAMBA	-	-	-	-	-	-	2.78	7.06	11.29	15.49	19.58	23.81	-	-	-	-	-	-
IQL	-	-	-	-	-	-	4.08	8.15	12.24	16.34	20.45	24.56	-	-	-	-	-	-
VDN	-	-	-	-	-	-	3.55	7.44	11.73	16.00	20.21	24.40	-	-	-	-	-	-
QMIX	-	-	-	-	-	-	3.67	7.56	11.64	15.85	20.06	24.28	-	-	-	-	-	-
QPLEX	-	-	-	-	-	-	3.79	7.68	11.69	15.82	20.03	24.24	-	-	-	-	-	-

TABLE I: MAPF Scores on Warehouse: Makespan is the latest agent arrival time. Bold text represents the best score except for LaCAM. "-" represents 0 in CSR and 128 in Makespan.

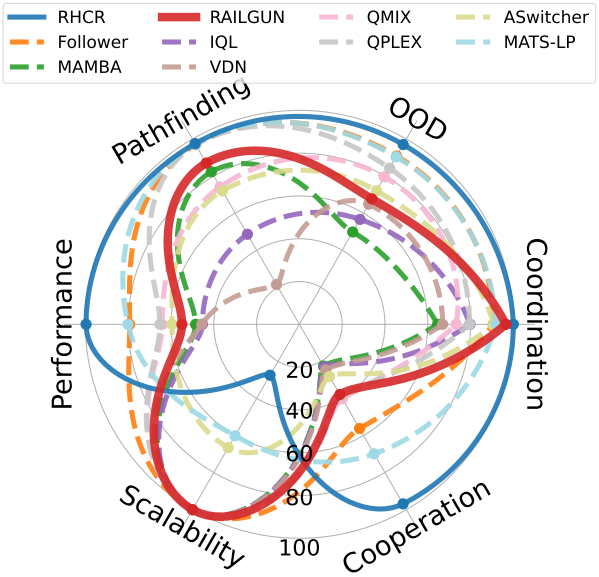

Fig. 6: LMAPF: RAILGUN still have good zero-shot LMAPF performace in Pathfinding, Coordination, Cooperation and Scalability just training on MAPF dataset.

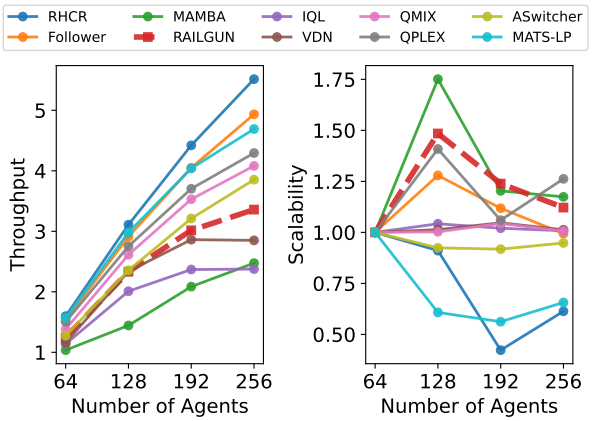

Fig. 7: LMAPF Throughput and Scalability Performance on Cities-tiles: Scalability is calculated by previous average per agent runtime divide by current one.

achieves a higher CSR score but a lower SoC compared to DCC. However, when considering Makespan, RAILGUN outperforms DCC. This indicates that RAILGUN is capable of finding relatively short solutions. As Makespan reflects the latest arrival time, many agents arriving before the last ones contribute to a higher SoC. This may be due to congestion situations, where many agents have a dead lock, and RAILGUN requires additional time to resolve the congestion. For LMAPF, as shown in Figure 7, RAILGUN's throughput score is better than those of VDN, IQL, and MAMBA. Although it does not achieve the best throughput score overall, its scalability is impressive. Figure 7 also demonstrates that as the number of agents increases, the average runtime per agent decreases.

As shown in Figure 6, when testing on LMAPF, a completely zero-shot task for RAILGUN, RAILGUN achieves only moderate scores in throughput-related metrics since none of the training data was optimized for throughput. However, this zero-shot test also demonstrates RAILGUN's strong generalization ability across different tasks. RAILGUN also attains high scores in Pathfinding, Coordination, and Scalability. These strengths and weaknesses suggest that, although RAILGUN's overall solution quality remains an issue, it can produce valid solutions in a diverse set of scenarios. Thus, we believe using a dataset optimized for the cost function of interest, combined with applying a task-specific reward function for fine-tuning via RL after the SL process, will help improve the overall solution quality.