

Deploying Ten Thousand Robots: Scalable Imitation Learning for Lifelong Multi-Agent Path Finding

He Jiang^{1*}, Yutong Wang^{2*}, Rishi Veerapaneni¹, Tanishq Duhan², Guillaume Sartoretti², Jiaoyang Li¹

Abstract—Lifelong Multi-Agent Path Finding (LMAPF) is a variant of MAPF where agents are continually assigned new goals, necessitating frequent re-planning to accommodate these dynamic changes. Recently, this field has embraced learning-based methods, which reactively generate single-step actions based on individual local observations. However, it is still challenging for them to match the performance of the best search-based algorithms, especially in large-scale settings. This work proposes an imitation-learning-based LMAPF solver that introduces a novel communication module and systematic single-step collision resolution and global guidance techniques. Our proposed solver, Scalable Imitation Learning for LMAPF (SILLM), inherits the fast reasoning speed of learning-based methods and the high solution quality of search-based methods with the help of modern GPUs. Across six large-scale maps with up to 10,000 agents and varying obstacle structures, SILLM surpasses the best learning- and search-based baselines, achieving average throughput improvements of 137.7% and 16.0%, respectively. Furthermore, SILLM also beats the winning solution of the 2023 League of Robot Runners, an international LMAPF competition sponsored by Amazon Robotics. Finally, we validated SILLM with 10 real robots and 100 virtual robots in a mockup warehouse environment.

I. INTRODUCTION

Multi-Agent Path Finding (MAPF) [1] is the problem of finding collision-free paths on a given graph for a set of agents, each assigned a start and goal location. Lifelong MAPF (LMAPF) [2] extends MAPF by continually assigning new goals to agents that reach their current ones. The main target of LMAPF is to maximize the throughput, which is defined as the average number of goals reached by all agents per time step. LMAPF has wide applications in the real world, such as automated warehouses, traffic systems, and virtual games. For example, Amazon fulfillment centers were reported to have more than 4,000 robots deployed in 2022 [3]. With growing demands in our daily life, larger autonomous systems are expected to be deployed in the near future. Therefore, more and more scalable search-based solvers have been developed for LMAPF in recent years [4], [5], [6], [7]. Existing search-based solvers [6], [7] can nowadays scale to thousands of agents.

Since learning-based LMAPF solvers¹ are expected to be more decentralized and scalable, numerous studies on

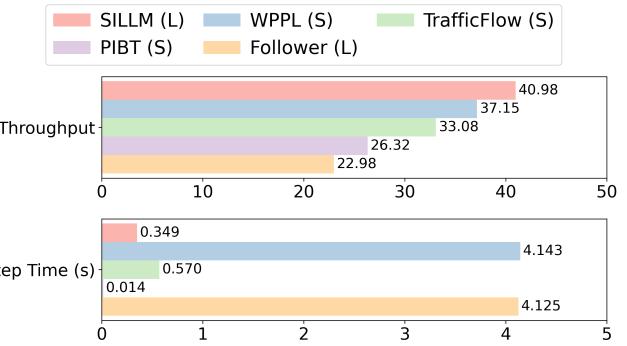


Fig. 1: Comparison of mean throughput and mean planning time per step between our solver, SILLM, with other state-of-the-art search- and learning-based solvers on 6 maps with 10,000 agents. The (L) and (S) in the legend denote learning-based and search-based solvers. Details are given in Table III.

learning have been conducted following the pioneering work PRIMAL [8] in 2019. However, most learning-based solvers have only been tested on small-scale instances involving tens to hundreds of agents [8], [9], [10], [11], [12].

Additionally, most learning papers emphasize the scalability of their solvers compared to optimal or bounded-suboptimal search-based solvers [13], [14], often showing positive results. This is largely because these search-based solvers struggle with computational complexity, as solving MAPF optimally is NP-hard. However, when compared to scalable suboptimal search-based solvers like PIBT [15], [5] (first introduced in 2019), most state-of-the-art learning-based solvers, such as PICO [9] or SCRIMP [10], would actually fail to beat them. Only very recently has a learning method Follower [12] been shown to outperform PIBT in throughput on small maps with up to 192 agents. Indeed, recent scalable search-based solvers based on PIBT, such as TrafficFlow [6] and WPPL [7], can act as even stronger baselines. Thus, our objective is to not only outperform existing learning-based methods but to outperform existing search-based methods as well.

In this paper, we propose Scalable Imitation Learning for LMAPF (SILLM), a learning-based solver that can manage up to 10,000 agents. Unlike prior works, we imitate a state-of-the-art scalable LMAPF solver on scenarios consisting of hundreds of agents [16], [7]. We further design a novel communication architecture, the Spatially Sensitive Communication (SSC) module, that emphasizes spatial reasoning to better learn highly cooperative actions seen in efficient LMAPF solutions. Finally, we integrate heuristic search

*The first two authors contribute equally.

¹These authors are with the Robotics Institute, Carnegie Mellon University, USA. {hejiangrivers,vrishi,jiaoyangli}@cmu.edu.

²These authors are with the Department of Mechanical Engineering, National University of Singapore, Singapore. {yutong-wang,e1280621}@u.nus.edu, guillaume.sartoretti@nus.edu.sg

¹Since solvers for MAPF can be easily adapted for LMAPF, we do not differentiate between MAPF and LMAPF in the rest of this paper, unless necessary.

improvements in safeguarding single-step collisions [17] and global guidance heuristics [7], [6].

Our experimental results show that SILLM outperforms state-of-the-art learning-based and search-based baselines on six large maps from common benchmarks with varying obstacle structures, achieving average throughput improvements of 137.7% and 16.0%, respectively, in tasks with 10,000 agents. We further demonstrate that SILLM is even able to outperform WPPL [7], the winning solution of the 2023 League of Robot Runners [18], which focuses on solving LMAPF challenges with up to 10,000 agents. Notably, with the aid of GPUs, SILLM takes less than 1 second to plan for 10,000 agents at each time step. Additionally, we validated SILLM with 10 real robots and 100 virtual robots in a mockup warehouse environment, further showcasing its potential for real-world applications. Our work highlights the effectiveness of learning-based methods for large-scale LMAPF instances and offers a comprehensive framework to unlock the full power of learning in future research.

II. BACKGROUND

The LMAPF problem includes a 4-connected grid graph $G = (V, E)$ and a set of n agents $A = \{a_1, \dots, a_n\}$, each with a unique start location. The vertices V of the graph G correspond to locations (namely unblocked grid cells), and the edges E correspond to connections between neighboring locations. Time is discretized into timesteps. At each timestep, an agent can move to an adjacent location by one of four move actions (up, down, left, right) or wait at its current location. During execution, we disallow vertex collisions and edge collisions. A vertex collision occurs when two agents occupy the same location at the same timestep. An edge collision occurs when two agents traverse the same edge in the opposite directions at the same timestep.

LMAPF requires real-time planning as agents are continuously assigned new goals. It assumes that an external task assigner handles goal assignments, with each agent knowing its next goal only upon reaching its current one. An LMAPF instance terminates after a predetermined number of timesteps. The objective is to plan collision-free paths for all agents while maximizing throughput, i.e., the average number of goals reached per timestep.

III. RELATED WORK

PRIMAL [19] is the pioneering work that adopts learning to solve the MAPF problem, which exploits the homogeneity in MAPF to train a decentralized policy shared by all agents. Subsequent research has focused on improving it from four main directions: enabling communication between agents [20], [10], incorporating global guidance into agents' field of view (FoV) [21], [12], enhancing collision resolution efficiency [10], and imitating search-based MAPF algorithms [8], [22]. In Table I, we summarize the learning-based solvers proposed in recent years that are commonly used as baselines, and in each subsection of Section IV, we explain the improvements we made to each technique.

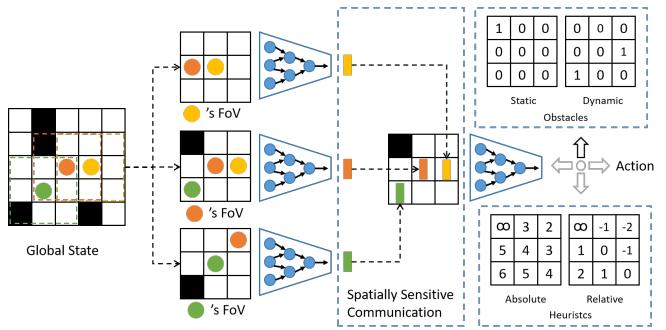


Fig. 2: Core network structure. The global state has all static obstacles (black squares) and agents (colored circles). As an example, an agent's FoV is of size 3×3 . The orange agent's 4 channels are shown in the right upper and bottom corners.

In addition to learning-based solvers, we are also interested in the recent progress in search-based methods. RHCR [4] is one of the most widely used baselines for LMAPF, which introduces the planning window to reduce the computation. It has a high solution quality but is still relatively slow. As a result, it can only manage hundreds of agents. On the other hand, PIBT [5] is a greedy single-step planner, which is very fast and scalable. It can coordinate thousands of agents but has a relatively low solution quality. TrafficFlow [6] incorporates traffic information to help PIBT avoid congestion and, thus, improves its solution quality. As a competitive approach, WPPL [7] is the winning solution of the 2023 League of Robot Runner Competition [18], which exploits PIBT to generate an initial windowed plan and then applies windowed MAPF-LNS [16] to refine the plan.

IV. METHODS

In this section, we first describe the network structure of SILLM, including the Spatially Sensitive Communication (SSC) module, in Section IV-A. Then, we discuss three types of global guidance in Section IV-B. Finally, we introduce the inference and training procedure in Sections IV-C and IV-D.

A. Neural Policy with Spatially Sensitive Communication

Communication is an important aspect of neural network design in multi-agent systems. As shown in the second row of table I, existing works mostly adopted attention-based communication (ABC) to aggregate information from neighboring agents, which only implicitly reason with the spatial information. However, we argue that precise spatial information benefits local collision avoidance, as similar designs, such as the conflict avoidance table [27], are frequently applied in search-based solvers. Therefore, we propose a spatial sensitive communication (SSC) module that explicitly preserves the spatial relationship between agents when aggregating information.

Figure 2 illustrates the core structure of our neural network. The neural network comprises two Convolutional Neural Network (CNN) modules and a Spatially Sensitive Communication (SSC) module. Since all agents are homogeneous in LMAPF, they share the same network weights.

TABLE I: Overview of learning-based solvers. The "Max #Agent" is based on experiments reported in the papers. Some methods allow communication between agents: "ABC" denotes attention-based communication, and "SSC" denotes our spatially sensitive communication. All methods use agents' FoV as inputs, while some also incorporate global guidance, such as a path calculated by A*, or determining whether the movement brings the agent closer to the goal using the Backward Dijkstra (BD). Our method, SILLM, uses three types of global guidance: BD, Static Guidance (SG), and Dynamic Guidance (DG), which are explained in Section IV-B. At each timestep, collisions are resolved by either keeping agents in their previous locations (Freeze) or allowing them to re-select new actions (Reselect). All methods are trained via imitation learning (IL), reinforcement learning (RL), or both (Mixed). IL uses search-based solvers as the imitation objective.

	PRIMAL 2019 [19]	MAPPER 2020[23]	PRIMAL2 2021 [8]	MAGAT 2021 [22]	DHC 2021 [20]	DCC 2021 [24]	SACHA 2023 [25]	RDE 2023 [26]	SCRIMP 2023 [10]	FOLLOWER 2024 [12]	SILLM (Ours)
Max #Agents	1,024	150	1,024	1,000	64	128	64	70	128	256	10,000
Communication	None	None	None	ABC	ABC	ABC	ABC	ABC	ABC	None	SSC
Global Guidance	None	Path	Path	None	Movements	Movements	Movements	Movements	Movements	Path	BD,SG,DG
Collision Resolution	Freeze	Freeze	Freeze	Freeze	Freeze	Freeze	Freeze	Freeze	Reselect	Freeze	Reselect
Training Approach	Mixed	RL	Mixed	IL	RL	RL	RL	RL	Mixed	RL	IL
Imitation Objective	ODrM*	None	ODrM*	ECBS	None	None	None	None	ODrM*	None	W-MAPF-LNS

Each agent a_i has a local Field of View (FoV) of size $V_h \times V_w$ (set to $V_h = V_w = 11$ in our experiments), centered at the agent's position, with a total of five observation channels. These channels are divided into o_1^i and o_2^i . o_1^i , with a size of $4 \times V_h \times V_w$, consists of four channels representing the locations of obstacles and other agents, as well as absolute and relative heuristic values (these will be explained in Section IV-B). o_2^i , with a size of $3 \times V_h \times V_w$, consists of three channels representing the locations of obstacles, other agents, and the goal of agent a_i .

We first use a CNN module to convert o_1^i into a 32-channel feature vector f_i . Then, a_i gathers the feature vectors of all neighboring agents within its local FoV through communication.² Subsequently, the SSC module generates a matrix o_3^i of shape $32 \times V_h \times V_w$, filled with -1, and inserts the gathered feature vectors into the corresponding agents' relative positions in a_i 's FoV (illustrated in Figure 2). o_3^i is then element-wise added to a matrix of the same shape, obtained by processing o_2^i through a Conv2d layer with a kernel size of 1. Finally, the resulting matrix undergoes further feature extraction through another CNN module and is decoded into the probabilities of five actions.

B. Providing Global Guidance with Heuristics

Many learning-based works incorporate global guidance to help agents move toward their goals more easily, as summarised in the third row of Table I. Some works try to follow a specific shortest path [21], [23]. However, since the shortest path might not be unique, other works encode the shortest distance from every location in FOV to the goal, computed by running a Dijkstra's algorithm backward from the goal [20], [10], [26], [25], [11]. A recent paper, Follower [12], tries to follow a shortest path that considers local traffic and is replanned at each timestep. However, no existing work reduces global traffic, which is important for systems with large agent numbers.

We systematically study three types of global guidance, represented as heuristics. The first type of heuristics is the

²The communication range can be smaller than the FoV, but for simplicity, we set them to be the same in the experiments.

aforementioned **Backward Dijkstra (BD)** heuristics applied in previous learning works, which tell agents the shortest distances to their goals. Further, we evaluate two other heuristics that were not applied in the previous learning works but effectively reduce global traffic.

The second type of heuristics is still Backward Dijkstra heuristics but based on specially designed edge costs that reduce traffic offline. We call it **Static Guidance (SG)**. In addition to telling the distances, SG heuristics encourage agents to move in the same direction, avoiding head-on collisions. Specifically, we adopt the classic crisscross highway [28], where the encouraged directions are alternating row by row and column by column. In practice, the cost of edge in the encouraged directions is 1, and the cost of edge in the discouraged directions is 100,000 for the warehouse or sortation maps and 3 for other maps. The cost of other edges are 2 by default. More advanced edge cost designs are also possible. For example, GGO [29] proposes an automatic way to optimize edge costs to maximize throughput.

The last type of heuristics is **Dynamic Guidance (DG)**, which further encodes dynamic traffic information, encouraging agents to move along short paths while avoiding congestion. We adopt the implementation in TrafficFlow [6], which plans a guide path for each agent and then asks agents to follow their guide paths as much as possible. When planning a guide path for an agent, TrafficFlow first counts the global traffic information based on the guide paths of other agents. Specifically, it counts the number of other agents visiting each location and traversing each edge along their guide paths. Then, dynamic edge costs are defined by handcrafted equations to punish the agent for moving to frequently visited locations and moving in the opposite directions of frequently visited edges. Therefore, the planned guide path could avoid potentially congested regions.³

Given the guide path of an agent a_i whose current goal is g_i , the heuristic value of location v for a_i is defined as

$$h(v) = SPCost(v, v') + GPCost(v', g_i),$$

³Readers interested in the implementation details are encouraged to refer to the original paper. We use the best variant in the paper.

where v' denotes the closest location to v in the guide path, $SPCost(v, v')$ represents the shortest path cost from v to v' , and $GPCost(v', g_i)$ is the remaining path cost from v' along the guide path to goal g_i .

We encode these three different types of heuristics in a unified way into the observation. Specifically, we denote the heuristic value at location v as $h(v)$ and encode the heuristic values in the FoV by a 2-channel 2D feature map that has the same size as the FoV. The first channel is the absolute heuristic value normalized by the map size, i.e., the feature value at location v_i is $h(v_i)/(M_h+M_w)$, where M_h and M_w are the height and width of the map G . The second channel is the relative difference in heuristic values, calculated by subtracting the heuristic value at the center of the FoV and dividing the FoV size, i.e., the feature value at location v_i is $(h(v_i) - h(v_c))/(V_h + V_w)$, where V_h and V_w are the height and width of the FoV, and v_c is the center of the FoV.

C. Safeguarding Single-Step Execution with CS-PIBT

Our neural policy has one remaining issue during inference: the generated actions may still contain collisions as they are independently sampled by agents. Most earlier works address it by “freezing” the agents’ movements that potentially lead to collisions by replacing their original actions with wait actions [19], as shown in the fourth row of Table I. However, this approach is inefficient as many unnecessary wait actions are introduced. It can also lead to deadlocks sometimes when agents repeatedly select the same conflicting actions. To mitigate this issue, SCRIMP [10] allows agents to reselect actions based on their action probabilities to avoid the original conflicts, but this has trouble scaling to many agents. Collision Sheild PIBT (CS-PIBT) [17] proposes to use PIBT [5] to resolve collisions.

In PIBT, each agent is assigned a priority. At each step, every agent ranks its actions in ascending order of the shortest distance from the resulting location to its goal (i.e., the BD heuristic). An agent always takes its highest-ranked action that does not collide with any higher-priority agent. If no collision-free actions exist for a low-priority agent, PIBT triggers a backtracking process, forcing the higher-priority agent to take its next best action until all agents can take collision-free actions. Given an agent’s action probabilities from the neural network, CS-PIBT converts them into a ranking (by biased sampling) and runs PIBT with this action preference to get 1-step collision-free actions.

We directly applied this idea in our work with slight modifications; unlike CS-PIBT, which safeguards the inference of a policy trained by other methods, this work holistically considers training and inference with it. We adopt a simplified variant of CS-PIBT, which always prioritizes the learned action output by the neural policy and then ranks other actions as the original PIBT. If the learned actions are collision-free, then CS-PIBT will not change them. We still call this safeguarding procedure CS-PIBT but name the combination of the neural policy and CS-PIBT as Learnable PIBT (L-PIBT) from the perspective of learning.

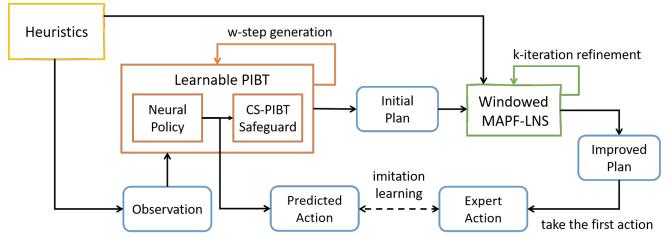


Fig. 3: Data collection procedure in the section IV-D.

D. Imitation Learning From A Scalable Search-Based Solver

We choose imitation learning because it is generally easier to learn team cooperation from mature search-based solvers, while reinforcement learning needs to explore the vast joint action space of multiple agents. A few earlier works have applied imitation learning, and they primarily focus on mimicking weak bounded-suboptimal algorithms, such as ODrM* [14] or ECBS [30], as shown in the last row of Table I. As a result, they are typically constrained to small-scale instances due to the heavy computation in data collection. For instance, PRIMAL2 [8] and SCRIMP [10] were trained on instances with 8 agents, while MAGAT [22] was trained on instances with 10 agents. When applying these solvers to large-scale instances, the significant differences between the original training setup and the actual application scenarios often result in a substantial decline in performance.

To address this issue, we directly imitate an anytime search-based algorithm, Windowed MAPF-LNS (W-MAPF-LNS), the central part of the winning solution WPPL [7]. It is unbounded suboptimal but scales well to large instances due to its planning window and anytime behavior.

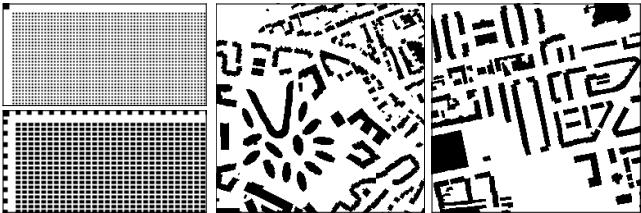
The data collection procedure for imitation learning is shown in Figure 3. Given the observation at the current step T , Learnable PIBT is called w times to generate the initial w -step paths for all agents. Then, we apply W-MAPF-LNS to refine their w -step paths for k iterations. At each iteration, W-MAPF-LNS selects a small group of agents heuristically and tries to improve their w -step paths. Specifically, W-MAPF-LNS tries to optimize agents’ w -step paths for an approximate objective:

$$Obj = \sum_{i=1}^n \left(\sum_{t=T}^{T+w-1} Cost(v_t^i, v_{t+1}^i) + h(v_{T+w}^i) \right),$$

where v_t^i is agent a_i ’s location at step t . The $Cost$ function records the edge cost from one location to a neighbor, and $h(v_{T+w}^i)$ is the heuristic value in Section IV-B, which estimates the future cost from v_{T+w}^i to agent i ’s goal. Notably, the heuristics are consistently used in the observation and objective to make policy learning easy. Empirically, we set $w = 15$ and $k = 5,000$ so that the planning time at each step is less than 1 second during data collection.

Then, we collect the first actions in the refined w -step paths with the current observation for later supervised training of Learnable PIBT. Notably, the imitation learning procedure can be repeated iteratively in a self-bootstrapping manner. After an iteration of supervised training, we ob-

TABLE II: Map visualization with details below. Sortation (top left) and Warehouse (bottom left) maps are from the LMAPF Competition [18], only showing the top left corners. Paris (middle) and Berlin (right) maps are from the MovingAI benchmark [1]. Random1 and Random2 are randomly generated maps with 10% and 20% obstacle densities. We use the underlined characters as the abbreviation for each map. #Locs is the number of unblocked locations and Agent density is defined as the number of agents divided by #Locs.



Name	Small-Scale			Large-Scale		
	Size	#Locs	Agent Density	Size	#Locs	Agent Density
Sortation	33*57	1,564	38.3%	140*500	54,320	18.4%
Warehouse	33*57	1,277	47.0%	140*500	38,586	25.9%
Paris	64*64	3,001	20.0%	256*256	47,096	21.2%
Berlin	64*64	2,875	20.9%	256*256	46,880	21.3%
Random1	64*64	3,670	16.3%	256*256	58,859	17.0%
Random2	64*64	3,221	18.6%	256*256	52,090	19.2%

tained a better Learnable PIBT, which could generate better initial w -step paths. Then, we can potentially obtain better improved w -step paths by W-MAPF-LNS, whose first actions are further used for the supervised learning of Learnable PIBT. Empirically, each iteration collects 15 million action-observation pairs in 50 episodes. We iterate the imitation 12 times and always select the best checkpoint.

V. EXPERIMENTS

Applying imitation learning directly in large-scale settings is possible, but complex engineering is required to deal with the large memory consumption during training. Therefore, we first downscaled the large maps to small ones but kept the original obstacle patterns. We trained the neural policy with 600 agents on each small map (500 timesteps for all maps) but evaluated it with 10,000 agents on the corresponding large map (3200 timesteps for Sortation and Warehouse and 2500 timesteps for others). More details of each map are covered in Table II. We provide more experiments, including the evaluation of different agent numbers and another benchmark used by Follower [12] in the project webpage⁴. The code will be released upon acceptance.

A. Main Results

This subsection compares different variants of our solver, SILLM, with other state-of-the-art search- and learning-based solvers. Specifically, we compare with SCRIMP [10] and Follower [12] for learning and with PIBT [5], RHCR [4], and TrafficFlow [6] for search. In addition, we also compare

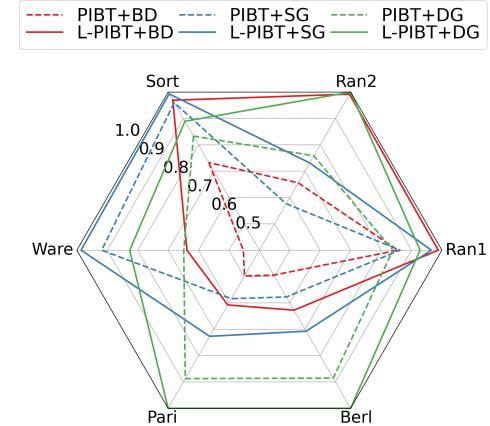


Fig. 4: The comparison of Learnable-PIBT (L-PIBT) and PIBT with different global guidance on large instances. The radar plot shows the score on each instance. The table reports the average score and the average single-step planning time.

SILLM with WPPL [7], the winning solution of the League of Robot Runners competition [18]. To compare different solvers more easily, we compute scores in the same way as the competition. Specifically, for each instance, we collect the best throughput achieved among all the solvers evaluated in this paper. Then, a score between 0 and 1 is computed as the throughput of the solver divided by the best throughput.

First, we compare the performance of PIBT and Learnable PIBT with different global guidance in Figure 4. Learnable PIBT (solid line) always achieves better throughput than its counterpart (dashed line). The increase in the planning time is not negligible but acceptable. Different types of global guidance excel at different kinds of maps in our experiments. For example, Static Guidance, implemented as a criss-cross highway, fits the criss-cross patterns of sortation and warehouse maps better. For random instances, normal Backward Dijkstra actually performs better. The potential reasons are: first, uniformly generated random structures implicitly make agents select more distributed paths; second, with lower agent density and potentially less congestion, normal Backward Dijkstra is a more accurate estimation of the future steps. Dynamic Guidance performs well across all maps and achieves the best results on city-type maps, such as Paris and Berlin, where obstacle patterns are less uniform.

Thus, we select the best performance achieved by Learnable PIBT with one of the global guidance to form the results of our solver, SILLM, in Table III. We also report the performance of using Dynamic Guidance only as SILLM (DG only). Then, we compare SILLM with other search- and learning-based algorithms in Table III. Regarding the score, SILLM outperforms all other learning-based and search-based methods. Specifically, on large instances, SILLM almost doubles the score of the previous state-of-art learning-

⁴<https://diligentpanda.github.io/SILLM/>

TABLE III: The comparison of different algorithms. The left part is the result of downscaled small instances. The right part is the result of the original large instances. We evaluate each instance with 8 runs of different starts and goals. Each column records the mean throughput with standard deviation in the parentheses. The Time (in seconds) and Score refers to the average single-step planning time and the average score. “-” indicates unavailable data due to the excessive planning time. We rank all the algorithms in descending order based on their average score on large-scale instances. The best throughput of each map is marked in bold. Notably, PIBT and TrafficFlow are exactly PIBT+BD and PIBT+DG in Figure 4.

Algorithm	Small maps with 600 agents								Large maps with 10,000 agents							
	Sort	Ware	Pari	Berl	Ran1	Ran2	Time	Score	Sort	Ware	Pari	Berl	Ran1	Ran2	Time	Score
SILLM	16.52 (0.09)	14.28 (0.12)	8.85 (0.11)	7.19 (0.14)	12.73 (0.11)	10.76 (0.12)	0.007	1.00	43.98 (0.07)	42.24 (0.05)	31.00 (0.89)	30.52 (0.98)	53.74 (0.10)	44.37 (0.11)	0.349	0.99
SILLM (DG only)	14.41 (0.16)	11.67 (0.54)	8.34 (0.07)	6.77 (0.28)	11.14 (0.13)	9.18 (0.14)	0.029	0.88	39.38 (0.09)	35.39 (0.26)	31.00 (0.89)	30.52 (0.98)	50.48 (0.14)	44.37 (0.11)	0.721	0.94
WPPL [7]	16.74 (0.06)	13.90 (0.16)	8.91 (0.07)	7.10 (0.20)	13.00 (0.09)	10.58 (0.15)	1.546	0.99	44.28 (0.07)	42.83 (0.04)	23.74 (0.20)	20.35 (0.29)	54.38 (0.09)	37.30 (0.43)	4.143	0.88
TrafficFlow [6]	11.78 (0.20)	9.10 (0.42)	7.44 (0.24)	5.70 (0.43)	10.35 (0.37)	8.48 (0.10)	0.016	0.76	36.89 (0.22)	27.78 (0.88)	27.51 (0.69)	27.03 (0.54)	45.62 (0.84)	33.64 (2.32)	0.570	0.81
PIBT [5]	7.79 (0.36)	4.62 (0.10)	5.59 (0.15)	5.12 (0.35)	10.84 (0.08)	6.80 (0.48)	0.004	0.60	32.44 (0.10)	19.39 (2.04)	15.43 (0.34)	15.09 (0.26)	46.51 (0.08)	29.08 (1.26)	0.014	0.61
Follower [12]	10.71 (0.10)	6.68 (0.31)	7.21 (0.10)	5.60 (0.19)	10.00 (0.07)	8.14 (0.10)	0.051	0.70	33.79 (0.06)	16.26 (0.17)	7.32 (0.32)	9.11 (0.29)	41.39 (0.10)	30.03 (0.82)	4.125	0.52
RHCR [4]	10.56 (0.27)	3.38 (0.24)	6.27 (0.08)	5.19 (0.13)	12.14 (0.09)	7.96 (0.12)	4.829	0.66	-	-	-	-	-	-	-	-
SCRIMP [10]	1.01 (0.10)	0.75 (0.06)	1.42 (0.18)	0.53 (0.05)	6.29 (0.55)	2.08 (0.27)	1.982	0.17	-	-	-	-	-	-	-	-

based method, Follower [12], and also significantly outperforms the state-of-art search-based method, TrafficFlow [6]. Compared to WPPL [7], SILLM performs closely or much better on all the maps and is much faster. Actually, with Dynamic Guidance only, SILLM (DG only) also achieves a better score than any other baselines on large instances. Earlier methods like RHCR [4] and SCRIMP [10] cannot scale to the large scale because of their heavy computation. RHCR and SCRIMP take minutes per step on large instances and are too costly to evaluate, failing to meet real-time requirements. RHCR is slow due to its use of a bounded suboptimal, but slower search method, ECBS [30], for each planning window. SCRIMP’s time consumption stems primarily from its single-step collision avoidance strategy, which requires action resampling and team state value calculation.

B. Ablation Study

We show ablation results in Figure 5. We first compare our spatially sensitive communication (SSC) with attention-based communication (ABC) and no communication (None) trained by imitation learning (IL). SSC consistently outperforms ABC and None, indicating the importance of precise spatial reasoning in LMAPF. We further compare our IL with a reinforcement learning (RL) implementation based on MAPPO [31]. Interestingly, we find our IL outperforms the simple MAPPO. We also tried to apply MAPPO after IL but did not notice any improvement. We believe more sophisticated RL methods are needed, especially those can properly distribute team-level rewards to individuals, which is left for future work.

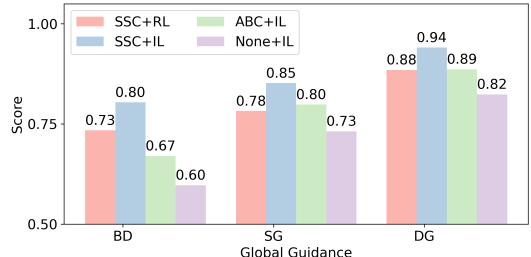


Fig. 5: Ablation studies on large maps.

C. Real-World Mini Example

Due to hardware and software limitations, we validate our solver with 10 real robots and 100 virtual robots in challenging mini warehouse environments with multiple corridors. Details can be found in the video demo.

VI. CONCLUSION

In this work, we show how to scale learning-based solvers to manage a large number of agents within reasonable planning times. Specifically, we design a unique communication module, incorporate efficient single-step collision resolution and different types of global guidance, and use scalable imitation learning with a state-of-the-art LMAPF solver. As a result, our proposed learning-based solver, SILLM, can effectively plan for 10,000 agents in under 1 second for various maps. It outperforms previously best learning- and search-based solvers and validates the potential of applying learning for large-scale LMAPF instances. Future work will explore how to further improve SILLM using RL.

REFERENCES

- [1] R. Stern, N. R. Sturtevant, A. Felner, S. Koenig, H. Ma, T. T. Walker, J. Li, D. Atzmon, L. Cohen, T. K. S. Kumar, R. Barták, and E. Boyarski, "Multi-agent pathfinding: Definitions, variants, and benchmarks," in *SoCS*, 2019, pp. 151–159.
- [2] H. Ma, J. Li, T. Kumar, and S. Koenig, "Lifelong multi-agent path finding for online pickup and delivery tasks," *arXiv preprint arXiv:1705.10868*, 2017.
- [3] A. S. Brown, "How Amazon robots navigate congestion," <https://www.amazon.science/latest-news/how-amazon-robots-navigate-congestion>, 2022, accessed: 2023-05-09.
- [4] 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*, vol. 35, no. 13, 2021, pp. 11 272–11 281.
- [5] 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.
- [6] 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, pp. 20674–20682.
- [7] H. Jiang, Y. Zhang, R. Veerapaneni, and J. Li, "Scaling lifelong multi-agent path finding to more realistic settings: Research challenges and opportunities," in *Proceedings of the International Symposium on Combinatorial Search (SoCS)*, 2024, pp. 234–242.
- [8] M. Damani, Z. Luo, E. Wenzel, and G. Sartoretti, "Primal .2: Pathfinding via reinforcement and imitation multi-agent learning-lifelong," *IEEE Robotics and Automation Letters*, vol. 6, no. 2, pp. 2666–2673, 2021.
- [9] W. Li, H. Chen, B. Jin, W. Tan, H. Zha, and X. Wang, "Multi-agent path finding with prioritized communication learning," in *2022 International Conference on Robotics and Automation (ICRA)*. IEEE, 2022, pp. 10 695–10 701.
- [10] 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.
- [11] 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, no. 16, 2024, pp. 17 531–17 540.
- [12] 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, no. 16, 2024, pp. 17 541–17 549.
- [13] 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.
- [14] C. Ferner, G. Wagner, and H. Choset, "Odrm* optimal multirobot path planning in low dimensional search spaces," in *2013 IEEE international conference on robotics and automation*. IEEE, 2013, pp. 3854–3859.
- [15] K. Okumura, M. Machida, X. Défago, and Y. Tamura, "Priority inheritance with backtracking for iterative multi-agent path finding," in *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19*. International Joint Conferences on Artificial Intelligence Organization, 7 2019, pp. 535–542. [Online]. Available: <https://doi.org/10.24963/ijcai.2019/76>
- [16] J. Li, Z. Chen, D. Harabor, P. J. Stuckey, and S. Koenig, "Anytime multi-agent path finding via large neighborhood search," in *IJCAI*, 2021, pp. 4127–4135.
- [17] R. Veerapaneni, Q. Wang, K. Ren, A. Jakobsson, J. Li, and M. Likhachev, "Improving learnt local mapf policies with heuristic search," in *Proceedings of the International Conference on Automated Planning and Scheduling*, vol. 34, 2024, pp. 597–606.
- [18] S.-H. Chan, Z. Chen, T. Guo, H. Zhang, Y. Zhang, D. Harabor, S. Koenig, C. Wu, and J. Yu, "The league of robot runners competition: Goals, designs, and implementation," in *ICAPS 2024 System's Demonstration track*.
- [19] 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.
- [20] Z. Ma, Y. Luo, and H. Ma, "Distributed heuristic multi-agent path finding with communication," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2021, pp. 8699–8705.
- [21] B. Wang, Z. Liu, Q. Li, and A. Prorok, "Mobile robot path planning in dynamic environments through globally guided reinforcement learning," *IEEE Robotics and Automation Letters*, vol. 5, no. 4, pp. 6932–6939, 2020.
- [22] 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.
- [23] 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 *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2020, pp. 11 748–11 754.
- [24] 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.
- [25] Q. Lin and H. Ma, "Sacha: Soft actor-critic with heuristic-based attention for partially observable multi-agent path finding," *IEEE Robotics and Automation Letters*, 2023.
- [26] J. Gao, Y. Li, X. Yang, and M. Tan, "Rde: A hybrid policy framework for multi-agent path finding problem," *arXiv preprint arXiv:2311.01728*, 2023.
- [27] T. Standley, "Finding optimal solutions to cooperative pathfinding problems," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 24, no. 1, 2010, pp. 173–178.
- [28] L. Cohen, "Efficient bounded-suboptimal multi-agent path finding and motion planning via improvements to focal search," Ph.D. dissertation, University of Southern California, 2020.
- [29] Y. Zhang, H. Jiang, V. Bhatt, S. Nikolaidis, and J. Li, "Guidance graph optimization for lifelong multi-agent path finding," in *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, 2024, pp. 311–320.
- [30] 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*, vol. 5, no. 1, 2014, pp. 19–27.
- [31] 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*, vol. 35, pp. 24 611–24 624, 2022.
- [32] A. Berndt, N. Van Duijkeren, L. Palmieri, and T. Keviczky, "A feedback scheme to reorder a multi-agent execution schedule by persistently optimizing a switchable action dependency graph," *arXiv preprint arXiv:2010.05254*, 2020.
- [33] A. Berndt, N. Van Duijkeren, L. Palmieri, A. Kleiner, and T. Keviczky, "Receding horizon re-ordering of multi-agent execution schedules," *IEEE Transactions on Robotics*, 2023.

APPENDIX

A. Visualization of all maps

We visualize all the large maps for evaluation in fig. 6 and all the down-scaled small maps for training in fig. 7, which keep the obstacle patterns in the corresponding large maps.

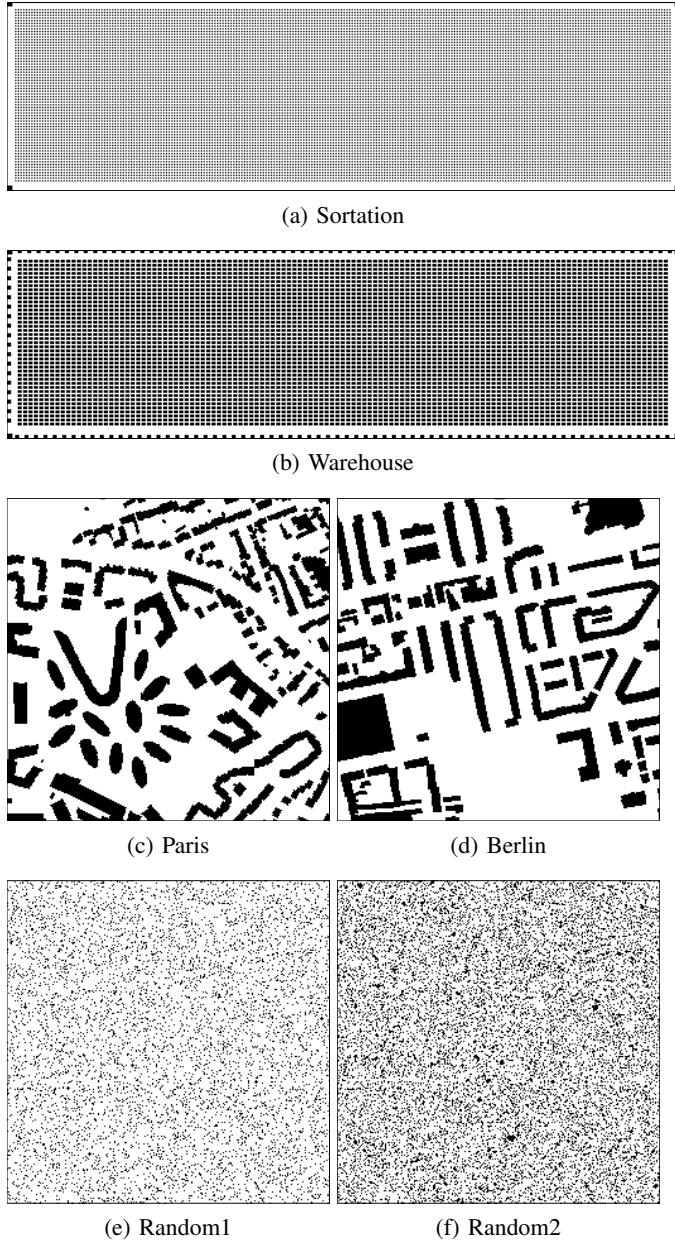


Fig. 6: Large maps with 10,000 agents for evaluation.

B. Evaluation with Different Numbers of Agents

In this section, we compare Learnable PIBT and PIBT with different global guidance and different numbers of agents. The conclusions are similar to the ones in section V. With the same global guidance, Learnable PIBT consistently outperforms PIBT, proving the effect of learning. Also, different global guidance excels in different scenarios. An

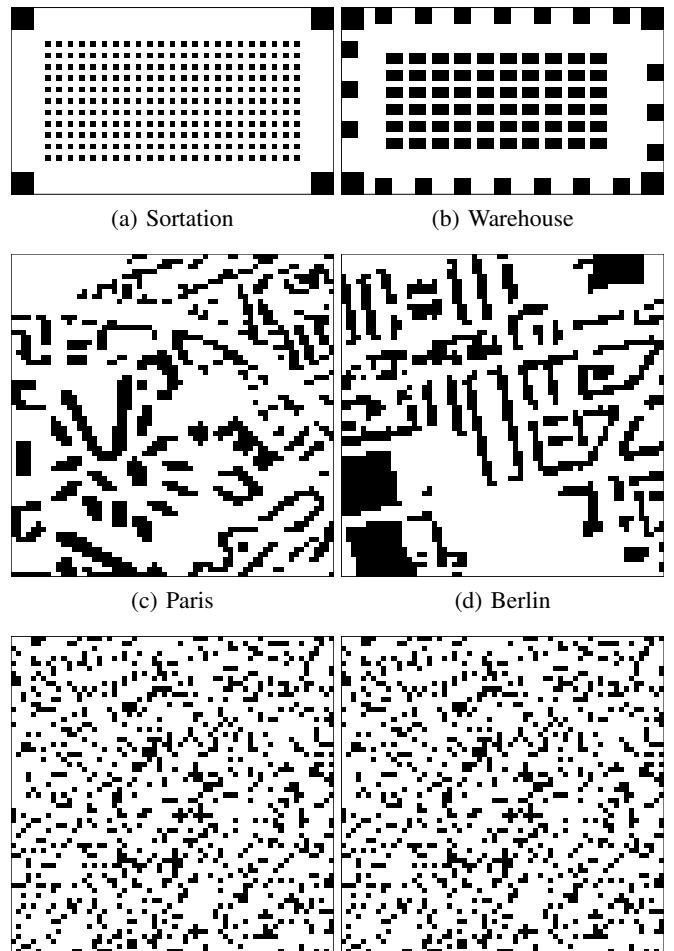


Fig. 7: Small maps with 600 agents for training.

interesting observation is that in map Random2, Backward Dijstraj (BD) performs better with $< 10,000$ agents, and Dynamic Guidance (DG) performs better with $> 10,000$ agents. The reason may be that with more agents, there is potentially more congestion, and DG addresses congestion better than BD.

C. Evaluation on Learn-to-Follow Benchmark

This section compares different decentralized methods with the Learn-to-Follow Benchmark [12] to validate the superiority of our SILLM (Learnable PIBT). Specifically, we compare Learnable PIBT trained with imitation learning and with reinforcement learning, Follower [12], SCRIMP [10], and PIBT [5]. For a simple comparison, we only use Backward Dijstraj as global guidance. All the training settings are the same as the ones in the Follower paper [12]. Notably, Learnable PIBT and Follower are trained on 40 Mazes maps and tested on 10 test maps and other maps. Our Learnable PIBT trained with imitation learning performs the best consistently across 4 different kinds of maps. Notably, Follower actually only outperforms PIBT in Mazes maps but may fail to outperform PIBT in other maps, which means the

generalization ability of Follower still needs improvement. In contrast, our Learnable PIBT is much more generalizable.

D. Real-World Mini Example

Since during the planning process, our algorithm assumes the position of all agents to be perfectly known at all times, we use ground truth positions for our virtual robots and use external localization (here, the Optitrack Motion Capture System) to obtain accurate position information for our real robots. However, the planned path may not be executed accurately due to disturbances and control inaccuracies. To eliminate these errors, we implement an Action Dependency Graph (ADG) [32], [33].

The video demo is available in the supplementary material. From our experiment with 10 real agents, we observe that agents can reach their goals quickly without collisions, and

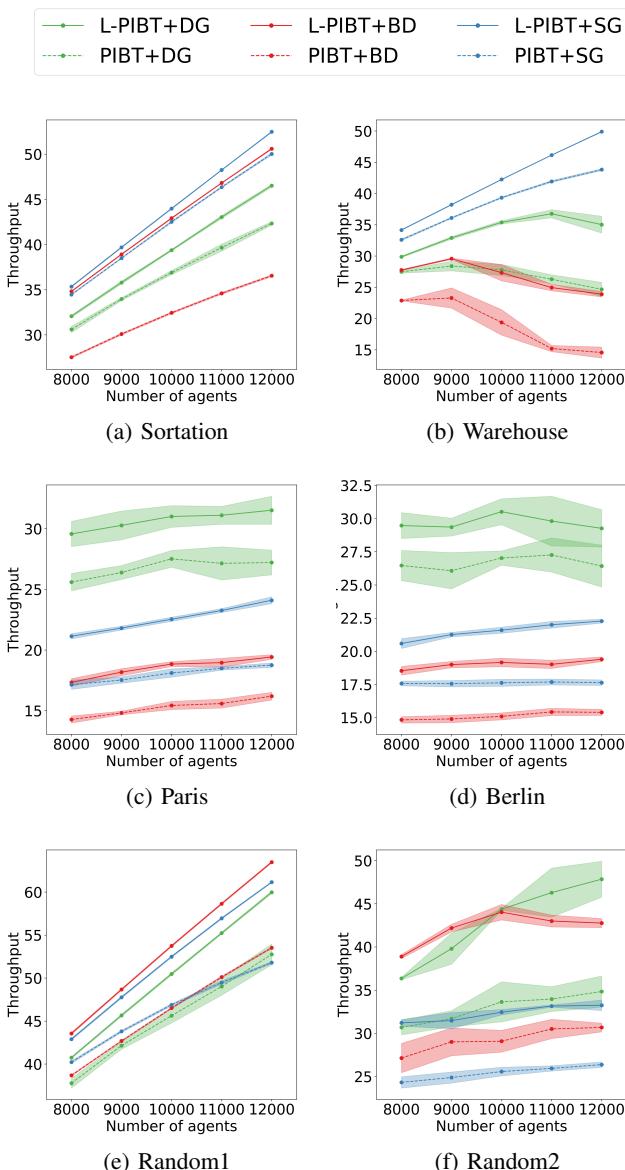


Fig. 8: Evaluation with different numbers of agents.

errors are eliminated by the ADG, demonstrating the potential of using our method in the real world. In our experiment with 100 virtual agents, we compare PIBT with Learnable PIBT. We can observe that Learnable PIBT outperforms PIBT with 50% more throughput.

E. Computation Resources

Our models are trained on servers with 72 vCPU AMD EPYC 9754 128-Core Processor, 4 RTX 4090D (24GB), and 240 GB memory. Training on each map takes less than 12 hours.

F. Reinforcement Learning Implementation

In the ablation study (Section V-B), we show that Imitation Learning is better than simple MAPPO-based Reinforcement Learning. Specifically, we use the following reward function.

$$r(v, v') = h(v) - h(v') - 1 \quad (1)$$

where h is the heuristic function defined in Section IV-B, v and v' are the current and next locations of an agent. Take Backward Dijkstra heuristics as an example. If v' is 1-step closer to the goal, the reward will be 0. Otherwise, the reward will be a negative penalty. If no other agents act as obstacles, the agent should follow its shortest path given this reward function after learning. Reward design is crucial to the performance of RL and worth further study.

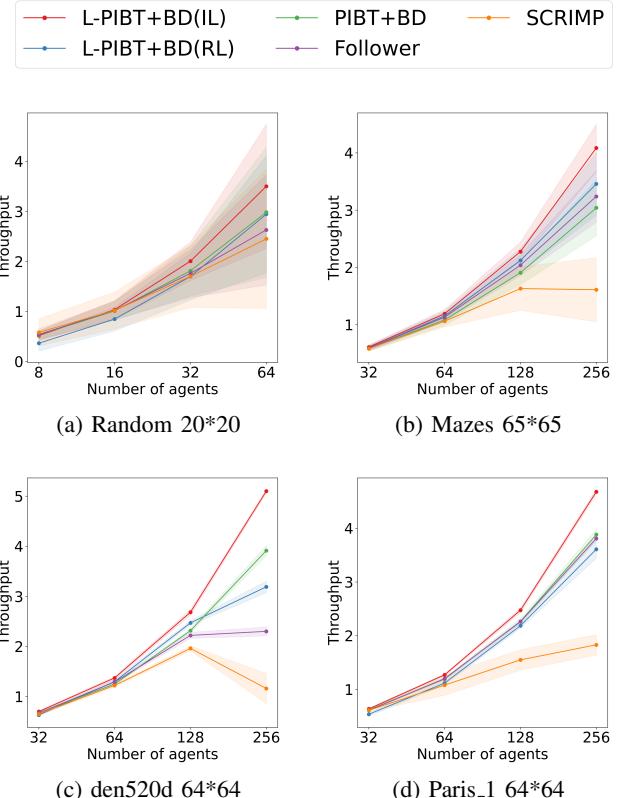


Fig. 9: Evaluation on Learn-to-Follow Benchmark.

TABLE IV: The comparison of PIBT and L-PIBT with different guidance. The left part is the result of downscaled small instances. The right part is the result of the original large instances. We evaluate each instance with 8 runs of different starts and goals. Each column records the mean throughput with standard deviation in the parentheses. The Time (in seconds) and Score refers to the average single-step planning time and the average score defined in Section V-A.

Algorithm	Small maps with 600 agents								Large maps with 10,000 agents							
	Sort	Ware	Pari	Berl	Ran1	Ran2	Time	Score	Sort	Ware	Pari	Berl	Ran1	Ran2	Time	Score
PIBT [5]	7.79 (0.36)	4.62 (0.10)	5.59 (0.15)	5.12 (0.35)	10.84 (0.08)	6.80 (0.48)	0.004	0.60	32.44 (0.10)	19.39 (2.04)	15.43 (0.34)	15.09 (0.26)	46.51 (0.08)	29.08 (1.26)	0.014	0.61
L-PIBT	14.76 (0.38)	8.69 (0.33)	7.76 (0.17)	7.16 (0.62)	12.73 (0.11)	10.76 (0.12)	0.007	0.89	42.91 (0.07)	27.34 (1.31)	18.84 (0.20)	19.16 (0.33)	53.74 (0.10)	44.02 (0.90)	0.024	0.80
PIBT+SG	13.66 (0.22)	9.91 (0.24)	6.18 (0.19)	4.50 (0.74)	11.66 (0.12)	7.46 (0.37)	0.004	0.74	42.51 (0.10)	39.34 (0.11)	18.11 (0.34)	17.62 (0.26)	46.89 (0.14)	25.57 (0.51)	0.014	0.75
L-PIBT+SG	16.52 (0.09)	14.28 (0.12)	8.85 (0.11)	7.19 (0.14)	12.34 (0.11)	10.09 (0.10)	0.007	0.98	43.98 (0.07)	42.24 (0.05)	22.54 (0.17)	21.59 (0.23)	52.49 (0.10)	32.44 (0.31)	0.023	0.85
PIBT+DG (TrafficFlow [6])	11.78 (0.20)	9.10 (0.42)	7.44 (0.24)	5.70 (0.43)	10.35 (0.37)	8.48 (0.10)	0.016	0.76	36.89 (0.22)	27.78 (0.88)	27.51 (0.69)	27.03 (0.54)	45.62 (0.84)	33.64 (2.32)	0.570	0.81
L-PIBT+DG	14.41 (0.16)	11.67 (0.54)	8.34 (0.07)	6.77 (0.28)	11.14 (0.13)	9.18 (0.14)	0.029	0.88	39.38 (0.09)	35.39 (0.26)	31.00 (0.89)	30.52 (0.98)	50.48 (0.14)	44.37 (0.11)	0.721	0.94

G. Detailed Comparison for Different Guidance

We report detailed throughput and average running time for comparing PIBT and L-PIBT with different guidance in Table IV. Notably, in contrast to the conclusion in the large map, Static Guidance works the best among the three heuristics in the small map for training. Overall, since all the heuristics are manually designed, the best heuristic is instance-dependent and should be evaluated empirically. Also, these heuristics only define the framework, but there could be a lot of hyperparameters that affect the performance. Automatic hyperparameter tuning can be helpful but doesn't necessarily remove the structural bias in the framework.