Learning-guided Prioritized Planning for Lifelong Multi-Agent Path Finding in Warehouse Automation

HAN ZHENG, Massachusetts Institute of Technology, USA
YINING MA*, Massachusetts Institute of Technology, USA
BRANDON ARAKI, Symbotic, USA
JINGKAI CHEN, Symbotic, USA
CATHY WU, Massachusetts Institute of Technology, USA

Lifelong Multi-Agent Path Finding (MAPF) is critical for modern warehouse automation, which requires multiple robots to continuously navigate conflict-free paths to optimize the overall system throughput. However, the complexity of warehouse environments and the long-term dynamics of lifelong MAPF often demand costly adaptations to classical search-based solvers. While machine learning methods have been explored, their superiority over search-based methods remains inconclusive. In this paper, we introduce Reinforcement Learning (RL) guided Rolling Horizon Prioritized Planning (RL-RH-PP), the first framework integrating RL with search-based planning for lifelong MAPF. Specifically, we leverage classical Prioritized Planning (PP) as a backbone for its simplicity and flexibility in integrating with a learning-based priority assignment policy. By formulating dynamic priority assignment as a Partially Observable Markov Decision Process (POMDP), RL-RH-PP exploits the sequential decision-making nature of lifelong planning while delegating complex spatial-temporal interactions among agents to reinforcement learning. An attention-based neural network autoregressively decodes priority orders on-the-fly, enabling efficient sequential single-agent planning by the PP planner. Evaluations in realistic warehouse simulations show that RL-RH-PP achieves the highest total throughput among baselines and generalizes effectively across agent densities, planning horizons, and warehouse layouts. Our interpretive analyses reveal that RL-RH-PP proactively prioritizes congested agents and strategically redirects agents from congestion, easing traffic flow and boosting throughput. These findings highlight the potential of learning-guided approaches to augment traditional heuristics in modern warehouse automation.

JAIR Track: Multi-Agent Path Finding

JAIR Associate Editor: Sven Koenig

JAIR Reference Format:
Han Zheng, Yining Ma, Brandon Araki, Jingkai Chen, and Cathy Wu. 2026. Learning-guided Prioritized Planning for Lifelong Multi-Agent Path Finding in Warehouse Automation. *Journal of Artificial Intelligence Research* 85, Article 28 (March 2026), 44 pages. DOI: 10.1613/jair.1.20611

1 Introduction

The rapid growth of e-commerce has intensified the demand for efficient, high-throughput logistics systems (Yu et al., 2016). Traditional manual workflows struggle with scalability and high labor costs (Živičnjak et al.,

*Corresponding Author.

Authors' Contact Information: Han Zheng, hanzheng@mit.edu, ORCID: 0009-0008-3768-5556, Massachusetts Institute of Technology, Cambridge, MA, USA; Yining Ma, ORCID: 0000-0002-6639-8547, yiningma@mit.edu, Massachusetts Institute of Technology, Cambridge, MA, USA; Brandon Araki, ORCID: 0000-0002-3094-1587, maraki@symbotic.com, Symbotic, Wilmington, MA, USA; Jingkai Chen, ORCID: 0000-0002-3528-8185, jichen@symbotic.com, Symbotic, Wilmington, MA, USA; Cathy Wu, ORCID: 0000-0001-8594-303X, cathywu@mit.edu, Massachusetts Institute of Technology, Cambridge, MA, USA.

DOI: 10.1613/jair.1.20611

Fig. 1. Robot fleet routing in a Symbotic warehouse (source: https://www.symbotic.com).

2022). To address these challenges, warehouse automation has transformed modern logistics by leveraging recent advances in robotics, optimization, and artificial intelligence (AI) to streamline operations and enhance efficiency (Custodio and Machado, 2020). Automated warehouse systems such as sortation and fulfillment centers have evolved into large-scale fleets of autonomous mobile robots (AMRs) navigating complex warehouse layouts (Wurman et al., 2007). As shown in Figure 1, Symbotic, a leading warehouse automation company, orchestrates advanced AMRs in high-density systems, reinventing warehouse automation for increased efficiency. In such AMR systems, effective and sustained coordination is essential to alleviate congestion, reduce operational costs, and maximize throughput. Even small improvements in AMR coordination can lead to substantial efficiency gains, boosting throughput while driving down costs (Farinelli et al., 2016; Atzmon et al., 2020).

A central paradigm for coordinating AMRs in warehouses is the Multi-Agent Path Finding (MAPF) problem (Stern et al., 2019), where multiple agents navigate from start positions to assigned goals while avoiding conflicts. MAPF aims to optimize global objectives such as minimizing total travel time (flowtime) or the completion time of the last agent (makespan). Although originally developed for warehouse automation, it also has broad applications in traffic coordination (Yan et al., 2024; Zheng et al., 2024), airport logistics (Von der Burg and Sharpanskykh, 2023), and video games (De Wilde et al., 2014; Sigurdson et al., 2018).

Classic MAPF assumes a static scenario with predefined goals and performs one-shot planning. Various methods have been proposed to solve one-shot MAPF, ranging from optimal methods like Conflict-Based Search (CBS) (Sharon et al., 2015) to heuristics that trade optimality for speed (Li et al., 2020a; Ma et al., 2019; Okumura et al., 2019). Among these, Prioritized Planning (PP) (Erdmann and Lozano-Perez, 1987) is a simple yet powerful heuristic that decomposes MAPF into sequential single-agent path plans based on a predefined priority order. With a well-chosen order, PP offers impressive scalability and efficiency in large, dynamic environments.

Extending beyond one-shot MAPF, real-world scenarios require a more dynamic formulation over the time dimension, known as lifelong MAPF (Ma et al., 2017), where agents are continuously assigned new tasks after completing their current ones. This formulation is crucial for modern warehouse automation, such as fulfillment centers and sortation hubs, where robots must transition seamlessly between tasks to maximize overall throughput continuously (Liu et al., 2019). Unlike one-shot MAPF, lifelong MAPF introduces several new challenges (Okumura et al., 2019): (i) agents repeatedly enter and leave the system, leading to continual re-coordination across time, (ii) congestion patterns evolve dynamically as tasks accumulate, requiring foresight rather than static planning, and (iii) myopic decisions can create cascading inefficiencies, as early plans directly shape future feasibility; (iv) paths with good quality need to be found within a realistic timeframe. Those challenges distinguish lifelong MAPF

from one-shot MAPF and motivate the need for frameworks that explicitly model long-horizon interactions while adapting to shifting conditions.

Recently, machine learning (ML) techniques have shown promise in helping address real-world complexities for one-shot MAPF (Sartoretti et al., 2018; Huang et al., 2021; Yan and Wu, 2024). However, they have yet to consistently outperform search-based methods in the more complex yet less explored lifelong settings (Damani et al., 2021; Skrynnik et al., 2024). As mentioned above, the key challenge is that planning in lifelong MAPF is inherently a sequential decision-making process with causal dependencies, where each planning decision directly influences future planning outcomes. Thus, approaches that only optimize for immediate conflict avoidance or short-term efficiency-as is often sufficient in one-shot MAPF-are insufficient in the lifelong setting, since they may worsen long-term congestion or deadlock risk. Effective solutions must account for these cascading effects by maintaining a longer preview horizon to explicitly model how current decisions shape future task allocations and path interactions. This requirement complicates the direct adaptation of straightforward one-shot or short-horizon learning-based approaches, which often struggle with efficiently handling long-term dependencies and dynamic task assignment. This presents an exciting opportunity to leverage deep reinforcement learning, in novel ways that explicitly capture the causal structure of lifelong MAPF, potentially enabling learning-based methods to surpass search-based techniques in such dynamic, real-world settings.

In this paper, we introduce **RL-guided Rolling Horizon Prioritized Planning (RL-RH-PP)**, the first hybrid framework that integrates reinforcement learning (RL) for dynamic priority order generation with search-based prioritized planning (PP). Our design choices are tailored to lifelong MAPF: the rolling horizon mechanism enables continual re-planning as new tasks arrive, while the RL-guided priority assignment explicitly captures long-horizon dependencies and adapts prioritization to evolving congestion patterns. Our motivation is to leverage reinforcement learning to address the hard-to-model sequential decision-making challenges of lifelong MAPF by capturing complex spatiotemporal interactions among agents, while integrating a simple yet efficient prioritized planning (PP) framework to achieve the best of both worlds. Importantly, PP provides an attractive backbone because it is lightweight, highly scalable, and easily extensible to dynamic settings, unlike more complex search-based solvers (e.g., CBS or PBS) whose computational cost grows rapidly with team size. The sequential structure of rolling horizon planning is naturally compatible with learning-guided optimization, as it exposes the long-horizon ordering decision as the key lever of coordination. This synergy allows us to focus the learning component on optimizing priorities, while relying on PP's efficiency to compute collision-free paths in real time, making the overall framework achieve the best of both worlds.

Specifically, RL-RH-PP casts dynamic priority assignment as a Partially Observable Markov Decision Process (POMDP) and uses a learned neural network to autoregressively decode priority orders. These orders are then fed into a rolling-horizon extension of Prioritized Planning (RH-PP), where multiple promising priority orders are sampled, with a conflict-repair mechanism applied as needed to ensure high-quality, conflict-free path planning. By integrating data-driven RL with PP, our learning-guided RL-RH-PP induces a dynamically streamlined search space for priority order sampling, significantly boosting throughput for lifelong MAPF, especially in complex, dynamic warehouse environments.

At the core of our proposed RL-RH-PP is a transformer-styled neural network (Vaswani et al., 2017) that captures both *temporal* and *spatial* dependencies in multi-agent interactions for lifelong MAPF. Specifically, the proposed encoder processes multiple agent paths spanning the time dimension using a dictionary-based embedding mechanism and stacked multi-head attention layers with alternating temporal and spatial attention mechanisms. The temporal attention captures long-term dependencies along each agent's path over time, while the spatial attention models dynamic interactions between agents in the warehouse map. This enables RL-RH-PP to dynamically adjust prioritization in response to changing navigation constraints. Based on the learned agent embeddings, the decoder then autoregressively constructs the total priority orders for RH-PP execution, where diverse promising priority orders can be efficiently sampled from such a learned policy. By integrating spatial

and temporal context in a data-driven manner, RL-RH-PP generates high-quality priority orders that account for both current and future agent interactions, which significantly improves the agent coordination and throughput, especially in dense environments with a high likelihood of congestion.

We train RL-RH-PP in a warehouse simulation environment built on real-world-inspired warehouse such as Amazon fulfillment center and Symbotic warehouses. Our work is the first to incorporate the Symbotic warehouse layout into lifelong MAPF research, whereas prior studies have predominantly focused on Amazon warehouse maps (Li et al., 2020b; Jiang et al., 2024). The Symbotic warehouse presents a fundamentally different structure, characterized by higher obstacle density, structured regional constraints, and distinct navigation challenges such as bottlenecks and congestion-prone cross-aisles. This contribution provides new benchmarking opportunities and helps validate lifelong MAPF methods in more constrained and operationally relevant settings.

The RL policy interacts with an RH-PP planner, observes state transitions, and receives feedback through a customized reward function. The reward incentivizes efficient lifelong MAPF by minimizing robust travel distances and reducing congestion. Extensive experiments demonstrate that RL-RH-PP significantly improves throughput and planning efficiency, achieving on average 25% higher throughput over RH-PP with random priority orders. Furthermore, RL-RH-PP exhibits strong zero-shot generalization across varying agent densities, planning horizons, and unseen maps, achieving the highest total throughput over various search-based baselines, especially in scenarios with high obstacle density. Experiments and analyses, including rendering priority heatmaps and agent movement traces, show that RL-RH-PP learns to assign higher priorities to agents in congested regions, alleviating bottlenecks and quickly recovering from severe deadlocks. We illustrate how RL-RH-PP steers agents away from congested regions, opening pathways for other robots and improving long-term throughput, making RL-RH-PP a robust solution for more efficient path planning in complex, dynamic warehouse automation environments.

To summarize, our key contributions are:

- We propose RL-RH-PP, the first deep reinforcement learning-guided framework that dynamically optimizes priority orders for lifelong MAPF.
- We introduce RH-PP, a rolling-horizon extension of Prioritized Planning (PP) that serves as the efficient backbone for learning-guided decision-making.
- We design a transformer-style neural architecture that captures both spatial and temporal dependencies, allowing for data-driven priority order optimization.
- We achieve gains in throughput, planning efficiency, and generalization, demonstrating RL-RH-PP's superiority over various baselines in solving the challenging lifelong MAPF.
- We conduct interpretable analysis using priority heatmaps and agent-trace comparisons, revealing how RL-RH-PP prioritizes congested agents and recovers from deadlocks and boosts throughput.

Notably, our findings highlight the power of learning-guided approaches in advancing multi-agent coordination in warehouse automation and beyond. The RL-RH-PP framework and training pipeline will be open-sourced to facilitate further research at https://github.com/MikeZheng777/RL-RH-PP.

The remainder of this paper is structured as follows. Section 2 discusses related work on MAPF and lifelong MAPF, highlighting existing learning-based approaches. Section 3 formally defines the MAPF and Lifelong MAPF problems. Section 4 introduces our rolling-horizon extension of PP, and formulates the lifelong priority assignment as a POMDP and presents the solution RL-RH-PP. Section 5 presents the evaluation and generalization behavior of RL-RH-PP and reports ablation studies on architectural and algorithmic choices. Section 6 provides an interpretation of the learned policy. We conclude with a summary of findings and future research directions in Section 7.

2 Related Work

We now discuss the bodies of work that form the foundation of our method and highlight how our approach differs from existing techniques.

2.1 Multi-Agent Path Finding

Classical multi-agent path finding (MAPF) (Stern et al., 2019) is NP-hard (Yu and LaValle, 2013) and involves computing conflict-free paths for multiple agents on a discrete graph. Due to the vast joint trajectory space (Sharon et al., 2015), most MAPF algorithms rely on single-agent planners such as A* (Hart et al., 1968) and SIPP (Phillips and Likhachev, 2011), treating other agents' paths as constraints. Conflict-Based Search (CBS) (Sharon et al., 2015) remains the state-of-the-art optimal MAPF algorithm, resolving conflicts via backtracking while ensuring completeness and optimality; however, its exponential worst-case complexity limits scalability. Enhanced CBS (ECBS) (Barer et al., 2014) improves efficiency by introducing bounded suboptimality, balancing speed and solution quality. For greater scalability, Prioritized Planning (PP) (Erdmann and Lozano-Perez, 1987) and Priority-Based Search (PBS) (Ma et al., 2019) plan sequentially based on agent priorities, treating higher-priority agents as constraints. While PP is efficient, it lacks completeness and optimality guarantees; PBS mitigates this by searching over global priority orders. Priority Inheritance with Backtracking (PIBT) (Okumura et al., 2019) further improves adaptability by dynamically adjusting priorities to prevent deadlocks, though it sacrifices optimality due to localized decision-making.

Beyond search-based families, one-shot MAPF has also advanced through powerful repair and search techniques. MAPF-LNS2 (Li et al., 2022) performs large-neighborhood search with fast repairing, yielding strong anytime behavior. LaCAM* (Okumura, 2024) targets real-time, large-scale settings with near-optimal performance via careful engineering of cardinal conflict handling. Compared to these approaches, prioritized planning remains a compelling choice for settings demanding fast and scalable solutions. Our contribution is orthogonal: we keep a lightweight prioritized-planning backbone for its speed and efficiency and learn to select effective global priority orders within to improve solution quality under lifelong planning.

2.2 Lifelong Multi-Agent Path Finding

Lifelong MAPF algorithms balance solution quality and computational efficiency, with different approaches excelling in distinct scenarios. Rolling-Horizon Conflict Resolution (RHCR) (Li et al., 2020b) introduces a planning window to manage complexity. It integrates well with centralized planners like CBS (Sharon et al., 2015) and PBS (Ma et al., 2019), enabling high-quality solutions in small but dense environments; however, RHCR struggles with scalability when handling thousands of agents due to its computational overhead. In contrast, PIBT (Okumura et al., 2019) offers a highly scalable alternative, employing a greedy single-step decentralized approach that achieves real-time performance even with thousands of agents, albeit with solution-quality trade-offs due to its local decisions. Our method follows a centralized rolling-horizon formulation where we use RL to propose a high-quality global priority order, rather than making purely local one-step decisions as in PIBT.

To better balance speed and quality, WPPL (Jiang et al., 2024)—winner of the 2023 League of Robot Runner Competition (Chan et al., 2024)—combines PIBT for fast initial plans with windowed MAPF-LNS (Li et al., 2021a) for refinement. Our approach shares the rolling-horizon concept with RHCR and prioritization strategies with PP (Erdmann and Lozano-Perez, 1987) but differs by leveraging reinforcement learning to dynamically optimize global priority orders within each window.

2.3 Learning-based Approaches for MAPF and Lifelong MAPF

A growing body of work integrates learning with search to improve scalability and solution quality. For one-shot MAPF, Huang et al. (2021) employ a linear model with handcrafted features to guide CBS, showing that even

simple predictors can assist traditional solvers. PRIMAL (Sartoretti et al., 2018) uses imitation and multi-agent RL for decentralized coordination based on local neighborhoods encoded by CNNs. Transformer-based models (Yan and Wu, 2024) improve MAPF-LNS (Li et al., 2021a) by learning powerful heuristics that scale to thousands of agents, while ALPHA (He et al., 2023) targets attention-based long-horizon planning in structured environments.

Several recent studies further explore how learning and search can be synergistic. Veerapaneni et al. (2024b) shows that augmenting learnt local policies with heuristic search yields consistent gains, and Wang et al. (2025) combines multi-agent RL with LNS, reinforcing the utility of hybrid pipelines that couple global repair with learned guidance. On the learning-rule side, simple imitation of CS-PIBT can outperform large-scale imitation learning for MAPF (Veerapaneni et al., 2024a), and social-behavior priors have been shown to improve coordination under dilemma-like interactions (He et al., 2024).

For lifelong MAPF specifically, PRIMAL$_2$ (Damani et al., 2021) and Followers (Skrynnik et al., 2024) use RL to provide global guidance over extended horizons. More recently, large-scale imitation learning has been shown to scale lifelong MAPF to ten thousand robots (Jiang et al., 2025), highlighting the promise of data-driven coordination at warehouse scale. Despite these advances, learning-based approaches have not universally displaced strong non-learning baselines such as RHCR (Li et al., 2020b) and LNS variants (Li et al., 2022, 2021a; Jiang et al., 2024). These observations motivate our evaluation protocol and the design of RL-RH-PP: rather than replacing search, we learn to propose high-quality global priority orders that a rolling-horizon PP backbone executes efficiently.

2.4 Prioritized Planning

Prioritized Planning (PP) was first proposed in Erdmann and Lozano-Perez (1987). It is a classic decoupled MAPF approach that assigns a strict priority ordering to agents and plans their paths sequentially, where each agent avoids conflicts with all higher-priority agents. While PP does not guarantee completeness or optimality, it remains popular for its efficiency and simplicity, with performance hinging on how the priority ordering is determined. Heuristic strategies include the query-distance heuristic (Berg and Overmars, 2005; Ma et al., 2019), which prioritizes agents by travel distance, the least-option heuristic (Wu et al., 2019), which favors agents with fewer feasible path options. When no strong heuristic is available, random priority orderings combined with random restarts can improve success rates (Bennewitz et al., 2002).

Most closely related to our approach is Zhang et al. (2022), who introduced a machine learning framework to automatically learn priority orderings for PP for one-shot MAPF. Their approach extracts handcrafted features, and trains linear ranking models to predict agent priorities by supervised learning. While the ML-guided PP proposed in Zhang et al. (2022) achieved moderate improvements over random priority ordering, it struggled to consistently outperform strong heuristics such as the query-distance heuristic (Berg and Overmars, 2005), reflecting the limitations of feature engineering and static supervision. Our work differs fundamentally: 1) We study lifelong MAPF with continual task arrivals, whereas Zhang et al. (2022) addresses the one-shot MAPF setting. 2) rather than relying on predefined features and supervised learning, we formulate dynamic priority assignment as a reinforcement learning problem. By integrating an attention-based neural policy, our approach adapts priorities online to congestion and spatial-temporal interactions, yielding significantly higher throughput in lifelong MAPF.

3 Problem Definition

Definition 1 (MAPF). *The Multi-Agent Path Finding (MAPF) problem involves computing conflict-free paths for a set of N agents, denoted as $\mathcal{A} = \{1, 2, \ldots, N\}$, operating on a graph $G = (V, E)$, where V represents the set of discrete locations and E denotes the connectivity between them. In this paper, we consider a two-dimensional grid map G with obstacles, where each location has up to four neighbors (up, down, left, right), and obstacles are represented*

as non-traversable locations. Each agent i is assigned an initial location $s_i \in V$ and a goal location $g_i \in V$. Time is discretized into timesteps, and at each timestep, an agent can either move to an adjacent vertex or remain stationary. A solution to MAPF is a set of conflict-free paths $\Pi = \{\pi_1, \pi_2, \ldots, \pi_K\}$ connect agents from initial locations to goals, where each path π_i is a sequence of locations traversed by agent i. A solution is feasible if it avoids:

- ***Obstacle Conflicts**: No agent occupies an obstacle vertex at any timestep.*
- ***Vertex Conflicts**: No two agents occupy the same vertex at the same timestep.*
- ***Edge Conflicts**: No two agents traverse the same edge in opposite directions at the same timestep.*

Note that MAPF solutions typically assume agents move synchronously in discrete timesteps, which may differ from real-world robotic systems that involve continuous physical dynamics. Our environment follows such standard MAPF assumption of discrete timesteps in the literature, and considers five movement types which are *up*, *down*, *left*, *right*, and *wait*. The objective of one-shot MAPF is to minimize total travel time (flowtime) or the completion time of the last agent (makespan). While MAPF has been extensively studied, it features a static one-shot scenario where each agent is assigned a single pre-defined goal and the task concludes upon reaching the goal. This limits applicability in real-world settings where agents must continuously perform new tasks, motivating the lifelong MAPF setting.

Definition 2 (Lifelong MAPF). *Lifelong Multi-Agent Path Finding (Lifelong MAPF) extends MAPF (Definition 1) to dynamic environments where agents are continuously assigned new goal locations upon completing their previous tasks, reflecting ongoing operations in real-world environments. Formally, given a set of agents $\mathcal{A}$ and a graph $G = (V, E)$, each agent a_i is assigned an infinite sequence of tasks, where each task consists of reaching a dynamically assigned goal location $g_i^t \in V$ at time t.*

Unlike standard MAPF, which terminates when all agents reach their initial goals, lifelong MAPF operates in an online setting, where a task assigner dynamically provides new goal locations throughout the operation. The objective of lifelong MAPF is to maximize **throughput**, defined as the total number of agents successfully reaching their goal locations over a given time horizon T, while ensuring that paths remain conflict-free. Lifelong MAPF is particularly relevant for warehouse automation and multi-robot systems, where agents must perform continuous tasks efficiently. In this paper, we study the challenging lifelong MAPF scenarios.

4 Method

In this section, we first review prioritized planning, then introduce Rolling-Horizon Prioritized Planning (RH-PP), and finally describe our reinforcement learning guided RL-RH-PP framework.

4.1 Preliminary

Prioritized Planning (PP) (Erdmann and Lozano-Perez, 1987) is a widely adopted decoupled approach for solving MAPF problems, known for its computational efficiency. Instead of planning paths for all agents simultaneously, PP plans the path for each agent according to a predefined priority order obtained via heuristic schemes and plans their paths sequentially via a low-level single-agent path solver e.g., SIPP (Phillips and Likhachev, 2011). Specifically, the path for the highest-priority agent is computed first, considering only static obstacles in the environment. The path for each subsequent agent is then planned by treating the paths of previously prioritized agents as dynamic obstacles, ensuring conflict-free navigation.

Definition 3 (Priority Order). *A priority order is a strict partial order defined over a set of agents $\mathcal{A} = \{1, 2, \ldots, N\}$, represented by a binary relation $\prec$ on $\mathcal{A}$. The relation $i \prec j$ indicates that agent i has a higher priority than agent j in the planning sequence. A priority order satisfies the **transitivity property**: if $i \prec j$ and $j \prec k$, then $i \prec k$. In general, a priority order is a **partial order**, meaning some pairs of agents may not be comparable.*

Definition 4 (Total Priority Order). *A total priority order is a special case of a priority order where every pair of agents is comparable. Formally, a total priority order $\prec$ on $\mathcal{A}$ satisfies the* ***totality property****: for any two agents $i, j \in \mathcal{A}$, exactly one of the following holds: $i \prec j$ or $j \prec i$. Under a total priority order, all agents are assigned unique, strict rankings, determining an unambiguous planning sequence.*

For simplicity, we represent a total priority order using a tuple notation. For example, a possible total priority order for three agents could be $(1 \prec 2 \prec 3)$, meaning agent a_1 plans first, followed by a_2, and then a_3. Traditional PP methods typically consider a total priority order, which is established prior to the planning process. Once the order is defined, planning for each agent is sequentially executed according to this strict ordering (Erdmann and Lozano-Perez, 1987; Bennewitz et al., 2002; Berg and Overmars, 2005). In contrast, partial priority orders arise in approaches like PBS (Ma et al., 2019), where priorities emerge dynamically during the planning process and cannot be predetermined. In this paper, we only focus on using RL to determine total priority orders for PP, and any reference to a priority order should be understood as a total priority order.

4.2 Rolling Horizon Prioritized Planning

In this subsection, we present the motivation, design, and discussion of Rolling Horizon Prioritized Planning (RH-PP), which serves as the backbone for the RL–guided framework introduced in the next subsection.

4.2.1 Why Prioritized Planning? Compared to other MAPF solvers, prioritized planning stands out for its simplicity and efficiency. In contrast, search-based MAPF solvers such as Priority-Based Search (PBS) (Ma et al., 2019) and Conflict-Based Search (CBS) (Sharon et al., 2015) must maintain an explicit search tree, where each node represents a different priority assignment or conflict resolution step. Such branching often grows exponentially with the number of agents in the worst case. Prioritized planning, on the other hand, bypasses explicit search trees by assigning a fixed priority order to agents and solving their paths sequentially. This results in a computational complexity that scales linearly with the number of agents, making PP a potentially desirable approach for large-scale planning and real-time applications.

While prioritized planning may not outperform search-based solvers in one-shot MAPF settings, its simplicity and scalability make it well-suited for lifelong MAPF and learning-guided optimization frameworks. In this paper, we exploit such properties by integrating prioritized planning into a reinforcement learning-guided framework that learns to automatically generate effective priority orders in a data-driven manner. This novel hybrid approach enables the adaptive coordination of agents in dynamic, real-world environments and demonstrates superiority over search-based methods in lifelong MAPF.

We also note that PIBT (Okumura et al., 2019) provides a decentralized alternative, where priorities are reassigned locally at each timestep. While this yields excellent scalability, the purely local and greedy nature of PIBT's decisions often leads to congestion or suboptimal routes in dense environments. By contrast, our RL-RH-PP (formally introduced in Section 4.2.2) framework leverages rolling-horizon prioritized planning with full priority orders sampled and evaluated per replan, allowing the RL policy to internalize global, multi-step interactions within the horizon while keeping the search space tractable. This balance between scalability and solution quality underpins our choice of prioritized planning as the backbone for RL-guided solver for lifelong MAPF.

4.2.2 Rolling Horizon Prioritized Planning. We now introduce the planning backbone Rolling-Horizon Prioritized Planning (RH-PP). RH-PP extends Prioritized Planning (PP) into a dynamic, lifelong setting by iteratively replanning paths in response to newly assigned tasks. Despite the long-standing perception of PP as a simple and suboptimal heuristic for one-shot MAPF, we observe that RH-PP, when combined with randomly sampled total priority orders, performs surprisingly well in lifelong MAPF scenarios.

Building on the concept of Rolling-Horizon Conflict Resolution (RHCR) (Li et al., 2020b), RH-PP divides both path planning and execution into discrete episodes that unfold sequentially over time. Each episode is governed by two key parameters: the planning horizon w and the execution horizon h, where we must have $h \leq w$. In each episode, the solver replans paths using PP within a bounded planning horizon w, following a total priority order $\prec$. Based on such priority order, the low-level single-agent pathfinder SIPP (Phillips and Likhachev, 2011) computes paths to visit a sequence of goal locations known at the current timestep, while treating the paths of already-planned higher-priority agents as dynamic obstacles. Once all paths are obtained, agents execute their movements for the first h timesteps. After completing the execution horizon, a new planning cycle begins, repeating the process with planning horizon w for solving the lifelong MAPF.

A key challenge in PP lies in its reliance on a fixed total priority order, which strongly affects solution quality. Poor prioritization can cause congestion and unnecessary detours, particularly in dense, constrained environments (Berg and Overmars, 2005). In RH-PP, determining an effective priority order at each planning step is even more critical, as decisions made within the current horizon directly impact the feasibility and efficiency of future planning. Since RH-PP operates in a rolling-horizon manner, suboptimal priority assignments can cause early commitments that lead to long-term conflicts or deadlocks, complicating the search for high-quality solutions in later episodes.

Traditional one-shot PP-based solvers primarily focus on a search process for high-quality priority assignments for agents. In Azarm and Schmidt (1996), all possible priority assignments are evaluated before execution. Bennewitz et al. (2002) employs search with hill-climbing to identify high-quality orders, while Berg and Overmars (2005) proposes a distance query heuristic to construct a good priority order. However, in high-obstacle-density environments, finding a high-quality priority order can be particularly challenging. This difficulty stems from the combinatorial nature of priority ordering, where the number of possible assignments grows exponentially in complex MAPF instances.

To address these challenges, we first propose a novel yet intuitive heuristic approach that has not been explored in the literature to our knowledge. RH-PP starts with a potentially infeasible priority order obtained through top-K random sampling guided by heuristics. We define the heuristic cost function for evaluating each candidate priority order $\prec_k$ as follows:

$$\text{cost}(\prec_k) = -\frac{1}{N}\Big(\sum_{i=1}^{N} e_i + \beta s_i\Big), \tag{1}$$

where:

- e_i is the path length of agent i in the initial solution computed under priority order $\prec_k$. If agent i fails to find a feasible path using the single-agent planner, then we make agent i follow its shortest path, and e_i is the length of the shortest path.
- $s_i = 1$ if agent i follows its shortest path, making the initial solution infeasible; otherwise, $s_i = 0$.
- β is a trade-off that heavily penalizes infeasible partial solutions.

The priority order with the lowest cost, $\prec^*$, is selected for execution. This heuristic favors priority orders that initially yield more feasible paths, i.e., those requiring fewer agent repairs, thereby promoting more efficient overall planning. The parameter K serves as a trade-off between solution quality and efficiency. A larger K increases the number of priority orders evaluated, potentially leading to a better solution but at a higher computational cost. Conversely, a smaller K improves efficiency but may yield less optimal paths.

Note that prioritized planning with any given total priority order can be incomplete (Ma et al., 2019; Stern, 2019). A fundamental requirement in RH-PP is ensuring that the final selected order from the Top-K set produces collision-free paths within the planning horizon w. To achieve this, we adopt a repair mechanism from RHCR (Li et al., 2020b), which converts an initially infeasible set of per-agent intended moves into a feasible conflict-free

movement within the w window by iteratively resolving local conflicts. Specifically, the repair process iteratively applies local conflict resolution at each timestep: If the initial solution is infeasible, agents are processed in random order, their desired moves are checked for conflicts, and necessary wait actions are inserted to prevent collisions. This mechanism guarantees a conflict-free schedule within the w-step horizon, but it is purely a safety repair: it does not guarantee progress, completeness, or deadlock resolution. For example, in cyclic blocking configurations (e.g., a loop where each agent wants to move into the next agent's cell), the repaired execution may stall or exhibit livelock/looping while remaining collision-free. Empirically, such deadlock behaviors are uncommon in our evaluated scenarios for RH-PP. We use the repair only to maintain safe execution under rolling-horizon planning, not to resolve all deadlocks.

4.3 Reinforcement Learning-Guided Rolling Horizon Prioritized Planning

As established, the quality of solutions generated by RH-PP is highly dependent on the choice of the total priority order. In the previous section, we introduced a simple yet intuitive heuristic for generating such orders. However, in lifelong MAPF, relying on randomly sampled orders is often insufficient for finding effective priority orders at each planning step, as decisions made in the current step directly influence future planning outcomes. This continual re-planning setting is fundamentally different from one-shot MAPF: new tasks continually arrive, congestion patterns evolve as agents execute sequences of goals, and short-sighted decisions can create cascading bottlenecks or deadlocks later. Capturing these long-term dependencies requires a more adaptive and forward-looking strategy.

Addressing the above challenges naturally aligns with a sequential decision-making process, which makes reinforcement learning (RL) well-suited for generating such orders. Meanwhile, RH-PP provides a structured yet flexible backbone for integrating RL: 1) its rolling-horizon framework ensures that planning decisions are dynamically revisited and refined; and 2) RL policy can generate promising priority orders in a data-driven manner, substantially reducing the search space for finding the best priorities. This combination directly addresses the lifelong MAPF challenges: the rolling horizon allows continual adaptation to new goals, while RL-guided sampling mitigates cascading inefficiencies by reasoning over long-horizon effects. In this section, we introduce an RL approach that intelligently generates priority orders dynamically at each timestep, optimizing for both immediate feasibility and long-term throughput.

To this end, we first reformulate the task of selecting an optimal total priority order at each planning step as a Partially Observable Markov Decision Process (POMDP). We then propose RL-RH-PP, a reinforcement learning-guided rolling horizon prioritized planning framework. The overall framework is illustrated in Figure 2. The RL policy interacts with an environment that includes a warehouse simulation and an RH-PP planner. At each planning step, the RL policy generates K total priority orders, those orders are passed to RH-PP and evaluated according to Eq 1, the selected $\prec^*$ is passed to subsequent planning to compute feasible paths for the agents. These paths are then executed within the warehouse simulation. Feedback rewards from the executed paths and next observations are used to update the RL policy. This iterative learning process enables the policy to refine its decision-making on sampling priority orders, progressively improving the quality of the total priority order and the resulting agent paths over multiple training episodes.

Our RL formulation is particularly beneficial in this lifelong setting, where sequential decision-making and adapting to new goals are essential to prevent cascading congestion, but these characteristics may not reflected in one-shot MAPF.

4.3.1 POMDP Formulation. We define the POMDP considered in this paper as follows, where:

- **State**: The state should encompass all agent locations, their goal lists, and any environment details needed to maintain the Markov property. However, explicitly modeling all such details can be cumbersome. Instead, we rely on partial observations-the agents' shortest paths-to capture the key spatiotemporal information

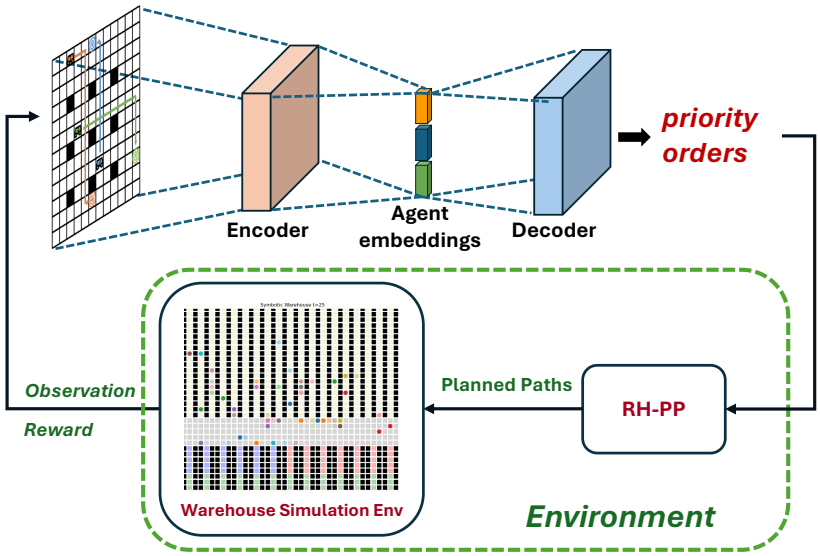

Fig. 2. The framework of our proposed RL-RH-PP. At each planning step, the RL policy encodes shortest path of each agents into agent embeddings and autoregressively decodes a total of K priority orders. These orders are passed to the RH-PP planner to compute the conflict-free path, which is then executed in the warehouse simulation. Feedback rewards are used to update the policy. This closed-loop interaction enables the RL policy to learn to generate effective priority orders during training.

for conflict resolution. The underlying transitions remain governed by the full (but unmodeled) state and the RH-PP module, while the agent learns directly from these compact observations.

- **Observation**: The observation $\boldsymbol{o}_t$ at time step t encapsulates essential information for the RL to determine the effective priority order. In this paper, we consider the observation to be the shortest paths σ_i^t that connect the current location of an agent i at the planning time step t to its future goal sequence. Mathematically, we have $\boldsymbol{o}_t = [\sigma_1^t, \sigma_2^t, ...\sigma_N^t]$. Note we define the position on the MAPF map using integers, then we have $\sigma_i^t \in \mathbb{Z}^r$, where r is the (maximum) length of the shortest path. Shortest paths are a natural and informative choice for representing MAPF observations in the literature, e.g., Yan and Wu (2024), as they capture both spatial and temporal information about an agent's intended trajectory. By including the full sequence of planned steps toward a goal, shortest paths implicitly encode potential conflicts, path overlaps, and estimated completion times, all of which are relevant for downstream prioritization.
- **Action**: For a MAPF instance with N agents, the space of all possible total priority orders is $N!$. We consider the action to be a set of K promising priority orders, which is used to reduce the search space to guide the RH-PP heuristics. Those orders are then passed to RH-PP to obtain the final feasible path.
- **Reward function**: The reward R guides the learning process and is defined as:

$$R(\boldsymbol{o}_t, a_t) = -\frac{1}{N}\Big(\sum_{i=1}^{N} d_{i,t} + \kappa c_{i,t} + \sigma s_{i,t}\Big). \tag{2}$$

In the lifelong setting, at time t agent i may have multiple outstanding goals arranged in a sequence. We define $d_{i,t}$ as the average Manhattan distance from the agent's current location to each of these future

goal locations. The congestion penalty $c_{i,t}$ is a binary indicator equal to 1 at time t if the agent's planned actions for the next h steps are all wait (the agent is locally stalled), and 0 otherwise. The infeasibility penalty $s_{i,t}$ is a binary indicator equal to 1 at time t if, under the selected priority order, RH-PP fails to find a feasible path for agent i, and 0 otherwise. To keep the binary penalties on a scale comparable to the distance term, we introduce weighting factors κ and σ, both set empirically to 1000. As shown in our experiments (Section 5.6.2), smaller weights slow convergence, while extremely large weights provide little additional benefit. Note that these weights are intended to match the magnitude of the total distance summed over all agents; therefore, they should scale with the map size (e.g., warehouse width + length), since larger maps typically induce longer average path lengths. Maximizing the return under this reward encourages fast progress while actively avoiding congestion and infeasible orders, thereby improving overall throughput. Note that our reward function mirrors the heuristic cost in Eq.1 but differs in a crucial way. Whereas e_i in Eq.1 is the prospective pre-execution path length computed at planning time, $d_{i,t}$ measures the actual remaining distance after the agent has already moved. Consequently, $d_{i,t}$ more directly reflects each agent's real progress and is thus more informative for increasing overall system throughput.

- **Discount factor**: The discount factor $\gamma = 0.99$ determines the importance of future rewards, promoting long-term planning.

Note that the transition function of our POMDP is implicitly determined by RH-PP. The RL policy learns directly from data collected by interacting with the warehouse simulation environment and observing state transitions on the fly during training.

4.4 Neural Architecture Design for RL-RH-PP

To effectively generate total priority orders that guide RH-PP, we employ a neural network composed of an *encoder* and an autoregressive *decoder*. The encoder extracts informative embeddings for each agent from the environment observations, while the decoder autoregressively generates the total priority orders.

4.4.1 Encoder Design. The encoder is responsible for transforming the raw input observation into a compact representation that effectively captures both spatial and temporal information about agents and obstacles in the environment. Given the observation at time step t, defined as $\boldsymbol{o}_t = \left[\sigma_1^t, \sigma_2^t, \ldots, \sigma_N^t\right]$ in section 4.3.1, we employ attention-based encoders to extract agent embeddings.

Figure 3 depicts the encoding process. To effectively represent the observation, we introduce a dictionary of learnable position embeddings, $\mathbf{X} \in \mathbb{R}^{d \times W \cdot H}$. Each column of $\mathbf{X}$ corresponds to a drivable location in the map, where d denotes the dimension of each embedding and W and H are the width and height of the warehouse map. This representation design provides generalization capability, enabling zero-shot transfer to environments with different numbers of agents and planning horizons, as well as to unseen maps with varied topologies (e.g., obstacle rearrangements or corridor geometries), while holding map size fixed.

For each agent i, its shortest path σ_i^t is used to index the embedding dictionary. Specifically, for each integral location $\sigma_{i,m}^t$ at index m in σ_i^t, we retrieve the corresponding column $\mathbf{X}[\sigma_{i,m}^t]$. This results in a sequence of embeddings that represent the agent's raw path. Formally, the representation h_i^t of the input path is defined as:

$$\mathbf{h}_i^t = \left[\mathbf{X}[\sigma_{i,0}^t], \mathbf{X}[\sigma_{i,1}^t], \ldots, \mathbf{X}[\sigma_{i,r}^t]\right] \in \mathbb{R}^{r \times d}. \tag{3}$$

We add sinusoidal positional encoding to every h_i^t before feeding them to temporal and spatial encoding layers following the Transformer design (Vaswani et al., 2017).

Temporal and Spatial Encoding. To encode both the temporal progression of each agent's trajectory and the spatial relationships among agents, we stack L encoder layers. Each layer ℓ applies two multi-head self-attention (MHA) blocks sequentially: one for temporal attention and one for spatial attention. In addition, we adopt

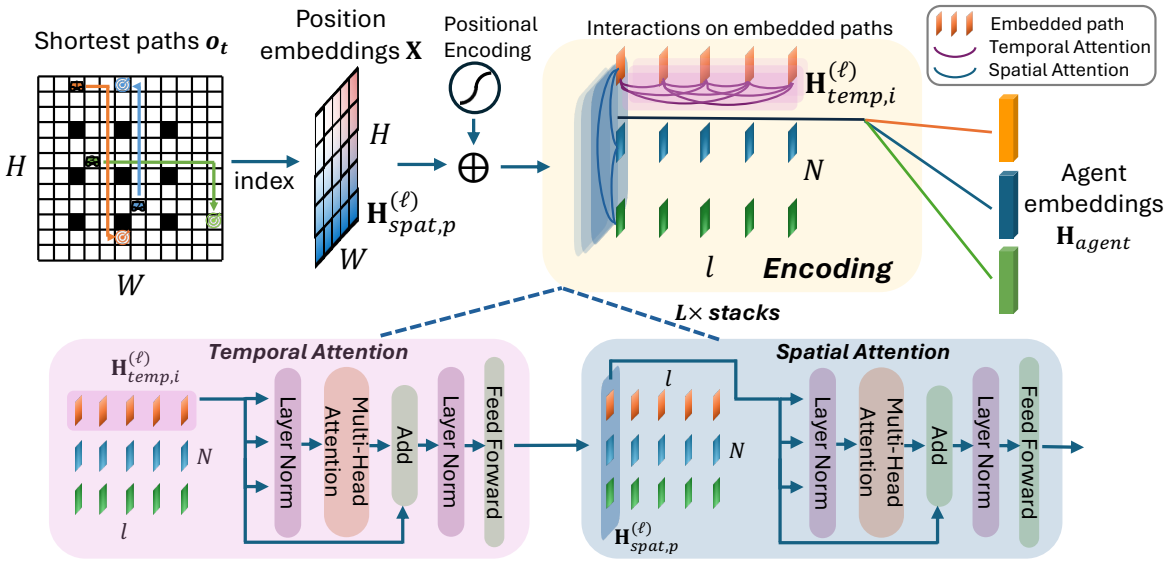

Fig. 3. Our proposed encoding process for RL-RH-PP. Raw path inputs are indexed using position embeddings to construct a learnable representation for agent paths. Temporal and spatial attention mechanisms are applied in turn, along with normalization and residual modules, to extract and refine agent embeddings.

pre-layer normalization and include a residual connection (He et al., 2016) within each attention block. This design integrates temporal information from each agent's trajectory and spatial interactions among all agents, while the residual connections and layer normalization (Ba et al., 2016) stabilize training and enable deeper stacking of layers. Let $\mathbf{H}^{(\ell-1)} \in \mathbb{R}^{N\times r\times d}$ be the output of the $(\ell-1)$-th layer, $\mathbf{H}^{(0)}$ as the input to the first layer of the encoder, where N is the number of agents, r is the (maximum) shortest path length, and d is the embedding dimension. In the rest within this section, we refer *agent dimension* as indexing a tensor along its N dimension while *time dimension* along its r dimension. The ℓ-th layer proceeds as follows.

Temporal Attention. Let $\mathbf{H}^{(\ell-1)}_{\text{temp},i} \in \mathbb{R}^{r\times d}$ be the index of $\mathbf{H}^{(\ell-1)}$ along agent dimension for agent i, We first normalize:

$$\widetilde{\mathbf{H}}^{(\ell)}_{\text{temp},i} = \text{LayerNorm}\big(\mathbf{H}^{(\ell-1)}_{\text{temp},i}\big). \tag{4}$$

Then we feed $\widetilde{\mathbf{H}}^{(\ell)}_{\text{temp},i}$ into a multi-head self-attention module operating along the time dimension l for each agent independently. We project $\widetilde{\mathbf{H}}_{\text{temp},i}$ into queries, keys, and values:

$$\mathbf{Q} = \widetilde{\mathbf{H}}^{(\ell)}_{\text{temp},i}\,\mathbf{W}^{Q}, \quad \mathbf{K} = \widetilde{\mathbf{H}}^{(\ell)}_{\text{temp},i}\,\mathbf{W}^{K}, \quad \mathbf{V} = \widetilde{\mathbf{H}}^{(\ell)}_{\text{temp},i}\,\mathbf{W}^{V}, \tag{5}$$

and compute attention in each head u using

$$\text{head}_u = \text{Softmax}\left(\frac{\mathbf{Q}_u\mathbf{K}_u^{\top}}{\sqrt{d/U}}\right)\mathbf{V}_u, \tag{6}$$

where U is the number of heads and $\mathbf{Q}_u, \mathbf{K}_u, \mathbf{V}_u$ denote the u-th partitions of $\mathbf{Q}, \mathbf{K}, \mathbf{V}$. After concatenating all heads and applying an output projection $\mathbf{W}^{O}$, we obtain:

$$\widehat{\mathbf{H}}^{(\ell)}_{\text{temp},i} = \text{Concat}\big(\text{head}_1, \ldots, \text{head}_H\big)\,\mathbf{W}^{O}, \tag{7}$$

following a residual connection:

$$\mathbf{H}^{(\ell)}_{\text{temp},i} = \widetilde{\mathbf{H}}^{(\ell)}_{\text{temp},i} + \widehat{\mathbf{H}}^{(\ell)}_{\text{temp},i}. \tag{8}$$

Spatial Attention. We applied the temporal attention for every agent, we stack back the output to get $\mathbf{H}^{(\ell)}_{\text{temp}} \in \mathbb{R}^{N\times r\times d}$, and index it along the time dimension for every time step p to get $\mathbf{H}^{(\ell)}_{\text{spat},p} \in \mathbb{R}^{N\times d}$. We then normalize:

$$\widetilde{\mathbf{H}}^{(\ell)}_{\text{spat},p} = \text{LayerNorm}\big(\mathbf{H}^{(\ell)}_{\text{spat},p}\big), \tag{9}$$

and apply multi-head self-attention across the agent dimension N for each time step, reusing the same multi-head attention (MHA) formulation but focusing on agent-to-agent interactions:

$$\widehat{\mathbf{H}}^{(\ell)}_{\text{spat},p} = \text{MHA}\big(\widetilde{\mathbf{H}}^{(\ell)}_{\text{spat},p}\big). \tag{10}$$

One can interpret this attention mechanism as operating on an agent-agent graph (Ying et al., 2021) at each time step, thereby capturing global interactions among all agents. The residual connection here yields:

$$\mathbf{H}^{(\ell)}_{\text{spat},p} = \widetilde{\mathbf{H}}^{(\ell)}_{\text{spat},p} + \widehat{\mathbf{H}}^{(\ell)}_{\text{spat},p} \tag{11}$$

We finally stack $\mathbf{H}^{(\ell)}_{\text{spat},p}$ for every time step p to obtain the output from the encoder $\mathbf{H}^{(\ell)} \in \mathbb{R}^{N\times r\times d}$.

Following each attention block, we apply the feed-forward layers with ReLU activation following the original Transformer (Vaswani et al., 2017). These layers use the same logic on layer normalization and a residual connection as the attention layer, maintaining stable gradient flow and facilitating deeper stacking. This additional transformation further refines agent representations before passing them to the next encoder layer.

Agent Embeddings. Repeating this process for $\ell = 1, 2, \ldots, L$ produces a final representation $\mathbf{H}^{(L)} \in \mathbb{R}^{N\times r\times d}$. We then choose the embeddings at the first index along the time dimension of $\mathbf{H}^{(L)}$ as the agent embeddings, denoted as $\mathbf{H}_{agent} \in \mathbb{R}^{N\times d}$. And $\mathbf{H}_{agent,i} \in \mathbb{R}^{d}$ is the final agent-level feature for agent i.

Remarks. The encoder is designed to capture both temporal and spatial dependencies efficiently. Temporal attention enables each agent to aggregate information from its trajectory, while spatial attention models interactions among agents at each time step. By stacking multiple layers, the model can learn hierarchical representations, capturing complex dependencies that are critical for effective multi-agent coordination.

4.4.2 Decoder Design. After obtaining the agent embeddings $\mathbf{H}_{agent} \in \mathbb{R}^{N\times d}$ from the encoder for N agents, the decoder generates total priority orders in an autoregressive manner. Figure 4 depicts the decoding process. Specifically, at each decoding step, one agent is selected until all N agents are chosen exactly once, forming a priority order (permutation) of the N agents. Finding the optimal priority order over all N agents constitutes a combinatorial optimization problem. Our decoder is inspired by recent advances in neural combinatorial optimization, particularly the attention model (Kool et al., 2019) originally developed for vehicle routing problems.

Autoregressive Decoding with Multi-Head Attention. We first project $\mathbf{H}_{agent}$ into keys, values, and glimpse keys:

$$\mathbf{K} = \mathbf{H}_{agent}\,\mathbf{W}^{K}, \quad \mathbf{V} = \mathbf{H}_{agent}\,\mathbf{W}^{V}, \quad \mathbf{L} = \mathbf{H}_{agent}\,\mathbf{W}^{L}, \tag{12}$$

where $\mathbf{W}^{K}, \mathbf{W}^{V}, \mathbf{W}^{L} \in \mathbb{R}^{d\times d}$ are learnable projection weights. We also compute a global embedding by averaging $\mathbf{H}_{agent}$ in the agent dimension and project it to a fixed context $\mathbf{f} \in \mathbb{R}^{d}$. These quantities in (12) and $\mathbf{f}$ are only computed once and remain unchanged throughput decoding to enhance efficiency.

In the following section, we describe our decoder and how it generates a single priority order. In practice, K orders can be efficiently generated in parallel, enabled by batched autoregressive decoding in deep learning frameworks such as PyTorch. The decoding proceeds in N steps, where at each decoding step n we sample one agent and then add to the total priority order. At each step n, we form a decoding step context $\mathbf{q}_n$ by combining

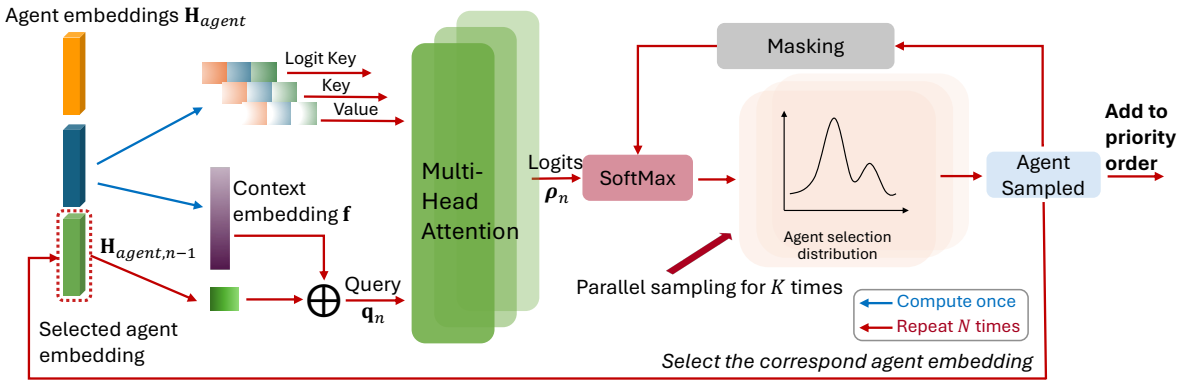

Fig. 4. Our proposed decoding process for RL-RH-PP. Agent embeddings from the encoder are fed into a self-attention mechanism to generate a sampling distribution. To construct each priority order, agents are sampled and added one by one in an autoregressive manner until all N agents are assigned. This process can be executed in parallel to generate a set of K promising orders efficiently.

the fixed context $\mathbf{f}$ with the embedding of the agent selected at the previous decoding step. If $n = 1$, we use a learned placeholder instead of a previously selected agent. Concretely,

$$\mathbf{q}_n = \mathbf{f} \; + \; \mathbf{W}^{Q_{\text{step}}}\left(\mathbf{H}_{agent,n-1}\right), \tag{13}$$

where $\mathbf{H}_{agent,n-1} \in \mathbb{R}^d$ is the embedding of the agent chosen at decoding step $n-1$ (or the placeholder for $n = 1$), and $\mathbf{W}^{Q_{\text{step}}} \in \mathbb{R}^{d\times d}$ is a learnable projection.

We then broadcast $\mathbf{q}_n$ across U heads to form $\mathbf{q}_{n,u}$. For each head u, we compute a dot-product attention with the keys and values:

$$\text{head}_u = \text{Softmax}\left(\frac{\mathbf{q}_{n,u}\,\mathbf{K}_u^{\top}}{\sqrt{d/U}}\right)\mathbf{V}_u, \tag{14}$$

where $\mathbf{K}_u, \mathbf{V}_u$ are the partitions of $\mathbf{K}, \mathbf{V}$ for head u. Concatenating the heads along the embedding dimension and applying a linear projection yields a glimpse vector $\mathbf{g}_n \in \mathbb{R}^d$. We compute unnormalized logits for each agent by

$$\boldsymbol{\rho}_n = \mathbf{L}\mathbf{g}_n, \tag{15}$$

where $\mathbf{L} \in \mathbb{R}^{N\times d}$ is defined in (12).

Sampling and Masking. Let $\mathcal{U}_{n-1}$ be the set of agents already chosen in the first $(n-1)$ decoding steps. We impose a mask over $\boldsymbol{\rho}_n$ by setting $\boldsymbol{\rho}_n[j] = -\infty$ for every agent $j \in \mathcal{U}_{n-1}$. The decoder then applies a softmax to the remaining logits:

$$p(a_n = i \mid \mathbf{q}_n, \mathcal{U}_{n-1}) = \frac{\exp\left(\boldsymbol{\rho}_n[i]\right)}{\sum_{v\notin\mathcal{U}_{n-1}} \exp\left(\boldsymbol{\rho}_n[v]\right)}. \tag{16}$$

To select an agent a_n at decoding step n, we sample from this categorical distribution:

$$a_n \sim p(a_n \mid \mathbf{q}_n, \mathcal{U}_{n-1}), \tag{17}$$

thereby ensuring that no agent is chosen more than once.

Algorithm 1 Training for RL-RH-PP

Input: Initial policy parameters θ_0, initial value function parameters ϕ_0
for $z = 0, 1, 2, \ldots, Z$ **do** ▷ Policy update iterations
  if $z \mod f = 0$ **then** ▷ Start collecting rollouts, reuse for f updates
    for $t = 0, 1, 2, \ldots, T$ **do** ▷ Simulate
      if $t \mod h = 0$ **then** ▷ Replan paths
        sample K priority orders from π_{θ_z} and pass them to RH-PP for path planning
      end if
    end for
    Collect set of trajectories $\mathcal{D} = \{\tau_j\}$
  end if
  Compute rewards-to-go $\hat{R}_t$ and advantage estimates $\hat{A}_t$ based on the current value function V_{ϕ_z}.
  Update the policy by maximizing the PPO objective:

$$\theta_{z+1} = \arg\max_{\theta} \frac{1}{|\mathcal{D}|T} \sum_{\tau \in \mathcal{D}} \sum_{t=0}^{T} \left[\min \left(\frac{\pi_\theta(a_t|\boldsymbol{o}_t)}{\pi_{\theta_z}(a_t|\boldsymbol{o}_t)} \hat{A}(\boldsymbol{o}_t, a_t),\ g(\epsilon, \hat{A}(\boldsymbol{o}_t, a_t)) \right) + \eta \mathcal{H}(\pi_\theta(\cdot|\boldsymbol{o}_t)) \right] \tag{18}$$

  Fit value function by regression on mean-squared error:

$$\phi_{z+1} = \arg\min_{\phi} \frac{1}{|\mathcal{D}|T} \sum_{\tau \in \mathcal{D}} \sum_{t=0}^{T} \left(V_\phi(\boldsymbol{o}_t) - \hat{R}_t \right)^2, \tag{19}$$

end for

Top-K sampling of Priority Orders. By repeating this procedure for $n = 1, 2, \ldots, N$, the decoder generates a permutation of N agents that includes each agent exactly once. Leveraging batch parallelism, the decoder generates K priority orders simultaneously in parallel. These orders are then passed to the RH-PP planner to solve for conflict-free paths. Please refer to Table 9 in the Appendix for detailed network hyperparameter selections.

4.4.3 RL Training. To train the reinforcement learning policy for generating effective total priority orders, we adopt Proximal Policy Optimization (PPO) (Schulman et al., 2017), a widely used on-policy RL algorithm known for its stability and sample efficiency. PPO optimizes the policy by iteratively interacting with the environment, collecting trajectories, and updating the policy while ensuring constrained updates to prevent instability. Algorithm 1 shows how our training for RL-RH-PP. We implement an actor-critic PPO framework where the policy and value networks share the same encoder architecture but maintain separate parameters. The value network uses the same encoder architecture as the policy network to obtain agent embeddings, which are then processed by a value decoder consisting of an attention layer followed by linear layers to produce a scalar value, similar to Ma et al. (2021).

To enhance sample efficiency during rollouts, we reuse the rollout dataset $\mathcal{D}$ for f iterations in both policy and value updates. We add the entropy loss $\mathcal{H}(\pi_\theta(\cdot|\boldsymbol{o}_t))$ to encourage exploration and prevent premature policy collapse. For stable training, we incorporate key implementation techniques, including advantage normalization and gradient clipping, as suggested by Shengyi et al. (2022).

5 Experimental Results

We present a comprehensive evaluation of our proposed RL-RH-PP approach across various warehouse environments and agent densities. Our evaluation is structured around the following key research questions:

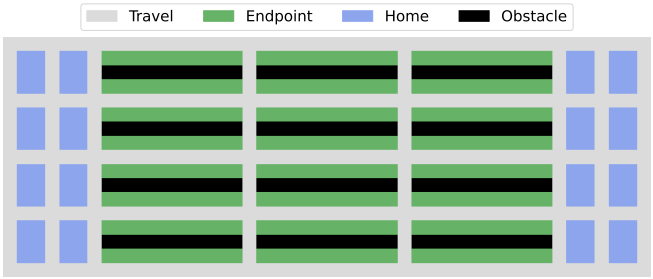

Fig. 5. Amazon fulfillment center dense map (obstacle density = 15.3%), modified from (Li et al., 2021b).

- **Q1:** What is the effectiveness of RL training in RL-RH-PP for boosting RH-PP (see Section 5.2)?
- **Q2:** How does RL-RH-PP compare to existing lifelong MAPF solvers in terms of throughput and solve time at inference time (see Section 5.3)?
- **Q3:** How does RL-RH-PP generalize to new scenarios with varying agent densities, planning horizons and warehouse layouts (see Section 5.4)?
- **Q4:** What is the anytime behavior of RL-RH-PP (see Section 5.5)?

To address these questions, we first outline the experimental setups, including the warehouse simulation environments, baseline methods, and evaluation metrics. We then present our key findings on throughput and solve time across two real-world-inspired warehouse layouts. Next, we assess the zero-shot transferability of the trained RL policy to different agent densities, planning horizons, and unseen maps. Finally, we conduct several ablation studies to analyze the effectiveness of various key designs in our method.

5.1 Experimental Setups

We introduce the details of our experimental setups.

5.1.1 Simulation Environment. We develop a lifelong warehouse simulation environment based on open-source MAPF benchmarks (Stern et al., 2019; Li et al., 2020b), using RH-PP as the back-end planner. The simulation takes a total priority order as input and outputs the corresponding reward and observation, as defined in Section 4.3.1. By default, policy is trained using a planning horizon of w=20, an execution horizon of h=5, and a total simulation horizon of T=800 steps. Experiments were conducted on two warehouse maps, with tasks randomly generated and assigned according to the respective warehouse layout. Specifically, we consider:

- **Amazon fulfillment center dense.** As shown in Figure 5, this map is characterized by multiple parallel aisles and narrow corridors. To create scenarios with high agent density, we reduce the size of the map to half of its original size in Li et al. (2021b). Agents are randomly initialized in the drivable free space, which consists of home and travel locations, and the goal location candidates are the endpoint locations (green blocks in Figure 5). When assigning a new task for each agent, the system first checks if the agent's current goal list is empty. If empty, it randomly selects an endpoint from a predefined set. It ensures that the selected endpoint is not the agent's current location and the endpoint is not already held by another agent.
- **Symbotic warehouse.** As shown in Figure 6, this map is already considered a high obstacle density environment, where the layout contains bottlenecks (e.g., small cross-aisles) that can cause agent congestion if not planned properly. Note that the map used in our simulation is shaped like a Symbotic warehouse,

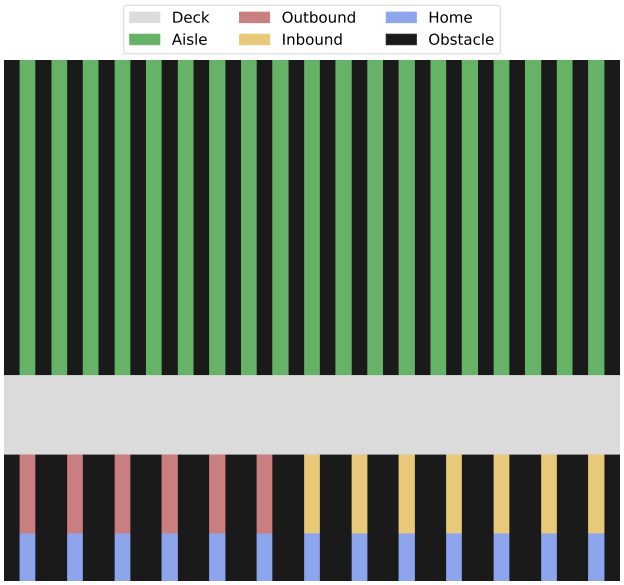

Fig. 6. Symbotic warehouse map (obstacle density = 56.6%).

but it is not based on exact dimensions. The entire map is divided into four regions: inbound, outbound, deck, and aisles. Agents handle package operations, where inbound and outbound are used for shipping in and out packages, respectively, and aisles serve as locations for storage and retrieval. The task assignment depends on the last goal location and the agent's loading state. Initially, all agents are randomly initialized in the deck and inbound areas with packages loaded. If the last goal location is an inbound station, the next destination is randomly assigned as an aisle station, after which the agent transitions to an unloaded state. If the last goal is from an outbound station, the next destination is randomly selected between an inbound station and an aisle station, resulting in the agent becoming loaded. When the last goal is from an aisle station, the next destination depends on the current loading state: an unloaded agent selects randomly between an inbound station and an aisle station and becomes loaded; and a loaded agent is directed to an outbound station, thereby returning to an unloaded state.

In both maps, non-drivable map locations are treated as static obstacles. The Amazon fulfillment center map has an obstacle density of 15.3%, while the Symbotic warehouse map features a much higher density of 56.6%. Hence, lifelong MAPF solvers may face significantly greater difficulty in finding feasible paths within the Symbotic map. For each map, we also vary the number of agents to adjust the density level, allowing us to evaluate both performance and scalability in different scenarios. Specifically, we train the RL policy with $N \in \{80, 100, 120\}$ agents for both warehouse environments. These settings create a high agent density; for instance, in the Amazon fulfillment center, the highest density in our setting is 2 times higher than that reported in Li et al. (2021b).

Finally, to enable more efficient roll-out sampling during training, we parallelize the environment on CPUs using Tianshou (Weng et al., 2022) and wrap it with OpenAI Gym (Brockman et al., 2016). These setups ensures scalable and efficient rollout data collection during RL training.

5.1.2 Training Setups. We configure the hyperparameters in Algorithm 1 as follows: the total number of epochs is set to $Z = 4000$, the rollout reuse per epoch to $f = 3$. For the PPO objective, we set the entropy loss weighting

factor to $\eta = 0.01$ and the PPO clipping parameter to $\epsilon = 0.2$. The optimization is performed using the Adam optimizer (Kingma and Ba, 2014) with an initial learning rate of $\lambda = 0.001$ and an exponential decay rate of 0.999 for both policy and value network updates. We use a batch size $B = 32$ and apply gradient clipping with a threshold $g_{clip} = 0.5$ to stabilize training. All the training hyperparameters are summarized in Table 9.

Training is conducted on a workstation using a single NVIDIA RTX 6000 Ada GPU. For each agent density setting, we train a separate RL policy from scratch. We use $K = 1$ for the Top-K sampling in RH-PP to improve rollout efficiency. The longest training duration, corresponding to $N = 120$, is approximately four days.

5.1.3 Evaluation Setups. To benchmark RL-RH-PP, we compare it against several representative non-learning-based solvers, including Priority-based Search (PBS) (Ma et al., 2019), Conflict-based Search (CBS) (Sharon et al., 2015), PIBT (Okumura et al., 2019) and WPPL (Jiang et al., 2024). Additionally, we include our proposed method, RH-PP, as part of the baselines.

These solvers are chosen for their relevance and complementary strengths in lifelong MAPF. CBS ensures optimality but struggles with scalability (Sharon et al., 2015), while PBS improves efficiency via global partial priority orders, making it suitable for large-scale problems (Ma et al., 2019; Li et al., 2020b). The prefix "RH" added to CBS and PBS indicates that they are implemented in a rolling horizon framework for lifelong MAPF following RHCR (Li et al., 2020b). PIBT performs greedy, single-step decentralized coordination with priority inheritance and backtracking to break cycles, delivering excellent scalability on large agent populations. WPPL, the winner of the 2023 League of Robot Runner Competition (Chan et al., 2024), combines PIBT (Okumura et al., 2019) for fast initial planning with windowed MAPF-LNS (Li et al., 2021a) for refinement, balancing speed and solution quality. Their diverse trade-offs in scalability, efficiency, and solution quality ensure a comprehensive evaluation of RL-RH-PP. We exclude purely end-to-end learning-based MAPF approaches such as PRIMAL$_2$ (Damani et al., 2021) and Followers (Skrynnik et al., 2024), as they have failed to consistently outperform non-learning-based solvers like RHCR (Li et al., 2020b).

Each solver is allocated a total time budget of 1s CPU time per planning step, following Jiang et al. (2024). For search-based methods such as PBS and CBS, finding a feasible solution within this limited time becomes challenging, especially in high-agent-density scenarios. To address this, we apply the same repair mechanism from Section 4.2 to any infeasible solution returned upon solver timeout. For experiments in this section, we use $K = 5$ and $\beta = 100$ for the Top-K sampling during testing. Each reported value is averaged over 16 held-out evaluation environments with different random seeds.

We track the total throughput of the warehouse over the $T = 800$ simulation episode in 16 held-out evaluation environments to monitor the training process, where each held-out environment has a different unseen random seed for initializing agent location and task assignments. We focus on two primary metrics:

- **Throughput per agent (TPA)**: The average number of tasks successfully completed per agent given the total $T = 800$ simulation horizon. Higher throughput implies more efficient task completion under continuous goal assignments.
- **Total throughput**: The total number of tasks successfully completed per agent given the total $T = 800$ simulation horizon, equivalent to TPA $\times$ N.
- **Solve time**: The average solve time per planning step, reflecting how quickly a solver can update paths. We track the solve time as only CPU wall time for other baselines, and CPU wall time + GPU inference time for RL-RH-PP since the RL policy is running on GPU.

5.2 Demonstrating the Effectiveness of RL

We first show the effectiveness of RL during training. Figure 7 depicts two typical throughput curves during training, on the Amazon and Symotic maps, respectively. To better visualize the curves, we apply smoothing using a moving average of 5 consecutive policy updates, and the curves show the averaged value over the 16

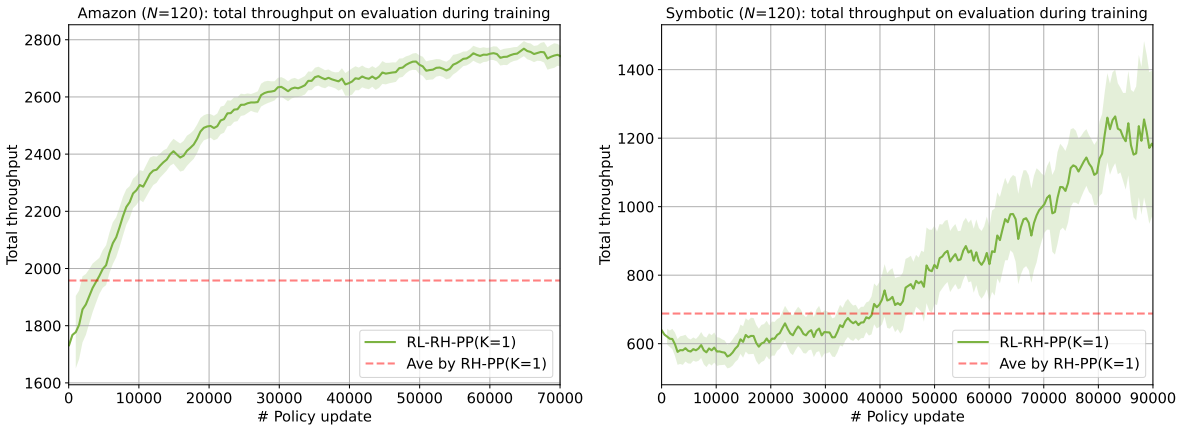

Fig. 7. Ave. total throughput on held-out evaluation environments during training with $N = 120$.

evaluation environments. To make the learning progress interpretable, the training plot is augmented with a horizontal reference line representing the corresponding RH-PP baseline.

Table 1. Evaluation comparison of RH-PP and RL-RH-PP with $K = 1$, RL-RH-PP policy trained from scratch with $w = 20$ for each setting, TPA = throughput per agent, 'Ama.' for Amazon and 'Sym.' for Symbotic.

	Method	$N=80$ TPA ↑	$N=80$ Time(s) ↓	$N=100$ TPA ↑	$N=100$ Time(s) ↓	$N=120$ TPA ↑	$N=120$ Time(s) ↓
Ama.	RH-PP ($K = 1$)	28.77 ± 0.91	0.13	22.49 ± 1.29	0.19	16.34 ± 1.22	0.25
	RL-RH-PP ($K = 1$)	**30.55** ± 0.56	0.13	**25.59** ± 1.01	0.20	**22.69** ±1.07	0.26
Sym.	RH-PP ($K = 1$)	15.86 ± 0.69	0.11	10.85 ± 4.69	0.16	5.97 ± 0.95	0.21
	RL-RH-PP ($K = 1$)	**17.73** ± 0.22	0.12	**13.29** ± 3.44	0.17	**10.74** ± 3.21	0.22

Table 2. Evaluation comparison of RH-PP and RL-RH-PP with $K = 5$, RL-RH-PP policy trained from scratch with $w = 20$ for each setting, TPA = throughput per agent, 'Ama.' for Amazon and 'Sym.' for Symbotic.

	Method	$N=80$ TPA ↑	$N=80$ Time(s) ↓	$N=100$ TPA ↑	$N=100$ Time(s) ↓	$N=120$ TPA ↑	$N=120$ Time(s) ↓
Ama.	RH-PP ($K = 5$)	30.85 ± 0.52	0.65	25.92 ± 1.32	0.78	21.74 ± 2.25	0.95
	RL-RH-PP ($K = 5$)	**31.80** ± 0.46	0.65	**28.84** ± 0.35	0.79	**25.56** ±0.55	0.96
Sym.	RH-PP ($K = 5$)	16.51 ± 0.35	0.68	11.39 ± 4.22	0.80	8.21 ± 2.24	0.96
	RL-RH-PP ($K = 5$)	**18.38** ± 0.50	0.68	**15.38** ± 1.85	0.82	**11.31** ± 2.21	0.99

From the plot, we observe that, initially, throughput is relatively low due to the random initialization of the policy; however, as training advances, the agent refines its prioritization strategy, potentially reducing congestion

and improving path efficiency; by the later stages of training, the throughput stabilizes, suggesting that the policy has converged to an effective strategy to generate priority orders. Compared to RH-PP, the RL-RH-PP curve rises above the RH-PP reference line early in training and remains consistently higher throughout the training process. This trend indicates that the RL policy is learning to generate more effective priority orders, leading to improved coordination among agents and more efficient task completion.

To further comprehensively demonstrate the consistent effectiveness of RL in enhancing RH-PP across various numbers of agents and inference samples K, Table 1 and Table 2 compares RH-PP with its learning-guided variant, RL-RH-PP, under the same sampling budget $K = 1$ and K=5 with planning horizon w=20. Across all settings, RL-RH-PP achieves higher throughput per agent (TPA) at essentially the same solving time, indicating that the priority orders proposed by the RL policy are consistently better than randomly sampled orders. On Amazon map, gains are notable at N=100 and persist on N=120. On Symbotic map, improvements grow with density/scale, from a modest lift at N=80 to substantial gains at N=120, again with comparable runtime. These results (i.e., an average improvement of 25%) demonstrate that learning to propose global priority orders significantly enhances RH-PP, especially in dense, high-traffic regimes where naive sampling struggles.

Table 3. Evaluation comparison of RL-RH-PP to various baselines, RL-RH-PP policy trained from scratch with $w = 20$ for each setting. TPA = throughput per agent. Best is **bolded**; second best is underlined.

	Method	$N=80$		$N=100$		$N=120$	
		TPA ↑	**Time(s)** ↓	**TPA** ↑	**Time(s)** ↓	**TPA** ↑	**Time(s)** ↓
Amazon	RH-CBS	4.94 ± 0.83	1.00	3.58 ±0.52	1.00	2.84 ±0.29	1.00
	RH-PBS	22.13 ± 7.58	1.00	4.92 ± 0.65	1.00	3.37 ± 0.26	1.00
	PIBT	21.19 ± 0.58	0.16	18.62 ± 0.35	0.20	16.09 ± 0.48	0.27
	WPPL	29.70 ± 0.51	1.00	26.60 ± 0.48	1.00	23.59 ± 0.26	1.00
	RL-RH-PP (ours)	**31.80** ± 0.46	0.65	**28.84** ± 0.35	0.79	**25.56** ±0.55	0.96
Symbotic	RH-CBS	3.27 ± 0.89	1.00	2.15 ± 0.78	1.00	1.50 ± 0.45	1.00
	RH-PBS	**20.83** ± 1.32	0.75	14.04 ± 5.39	1.00	1.76 ± 1.10	1.00
	PIBT	5.16 ± 1.02	0.11	3.65 ± 0.77	0.15	2.67 ± 0.49	0.21
	WPPL	16.29 ± 1.31	1.00	13.73 ± 1.83	1.00	10.05 ± 1.33	1.00
	RL-RH-PP (ours)	18.38 ± 0.50	0.68	**15.38** ± 1.85	0.82	**11.31** ± 2.21	0.99

5.3 Benchmarking Against Baselines

We now benchmark the TPA of our proposed RL-RH-PP (K=5) against baselines mentioned in Section 5.1.3. Table 3 shows that RL-RH-PP achieves the best or near-best throughput per agent across both warehouse layouts and all tested scales, while maintaining inference times comparable to the strongest baselines. On the Amazon map, RL-RH-PP consistently improves over both classical solvers (RH-CBS, RH-PBS) and strong pipelines (WPPL) at similar CPU time. PIBT remains the fastest but yields clearly lower throughput. On the Symbotic map, our RL-RH-PP performs comparably to the strongest solver, RH-PBS. However, as agent density increases, the performance of RH-PBS degrades significantly, reflecting its sensitivity to congestion and scalability. In contrast, our RL-RH-PP maintains robust performance and becomes notably stronger in high-traffic regimes, surpassing all search-based methods without incurring additional inference-time cost. Overall, the proposed learning-guided RL-RH-PP achieves robust gains in coordinating robots to maximize throughput by learning to predict global priority orders, while preserving practical runtime, particularly in more complex scenarios under heavy congestion.

A natural question is whether using step-wise decentralized coordination can match lifelong performance. Here, we address this question by focusing on the comparison of RL-RH-PP vs PIBT, which assigns and inherits priorities

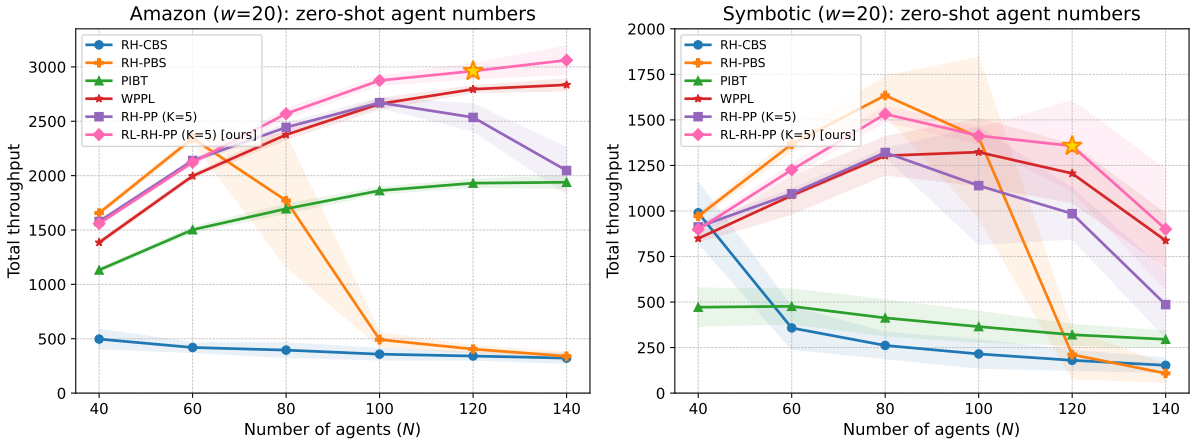

Fig. 8. Total throughput versus agent number N in zero-shot transfer evaluation at $w = 20$. The base model (yellow star) is trained at $N = 120$ and $w = 20$, and evaluated zero-shot at other N.

step-wise at each timestep without lookahead rolling-horizon planning. Empirically, RL-RH-PP outperforms PIBT across both maps and scales in Table 3, with the gap widening at higher densities on the Symbotic layout. We attribute this to RL-RH-PP's rolling-horizon planning and learned global priority orders, which explicitly account for cascading, long-horizon interactions. In contrast, PIBT's per-step, myopic updates lack lookahead and are more sensitive to congestion, leading to reduced throughput despite its lower runtime.

5.4 Generalizing to Different Scenarios

Since the proposed method involves learning, its zero-shot generalization performance is evaluated across diverse setups, including variations in the number of agents, planning window size, and unseen map layouts.

5.4.1 Generalization across different numbers of agents. We assess zero-shot transfer across agent populations using a single base policy trained at $N = 120$ and planning horizon $w = 20$. At inference time, we fix the same planning horizon and vary N without any retraining, keeping all planner settings identical across methods for fairness. Figure 8 plots total throughput as a function of N.

We report the results in Figure 8, where similar performance can be observed compared to the in-distribution performance shown in Section 5.3, indicating consistent robust performance of our proposed RL-RH-PP. Specifically, on the Amazon maps, the strong search-based baseline RH-PBS slightly outperforms the proposed method at lower agent densities. However, as the number of agents increases, its effectiveness drops significantly, while our RL-RH-PP maintains the best performance throughout. RH-PP remains a strong non-learning reference with random orders; however, our RL-RH-PP consistently improves upon it even with zero-shot generalization, indicating that the policy has learned generalizable knowledge on predicting higher-quality global priority orders, especially under heavier congestion. Moreover, it is notable that as the number of agents increases, the performance of the original RH-PP declines, while RL-RH-PP continues to improve. This suggests that RL-RH-PP is less constrained by the performance ceiling of the RH-PP backbone. The RL component potentially enables better adaptation under higher agent densities, partially mitigating limitations of the underlying RH-PP. This is a compelling case for learning-guided optimization. On the more constrained Symbotic maps, throughput is lower and decreases as the number of agents increases for all methods, likely due to inherent environmental patterns. Compared with RH-PP, our proposed method consistently achieves better performance despite operating under

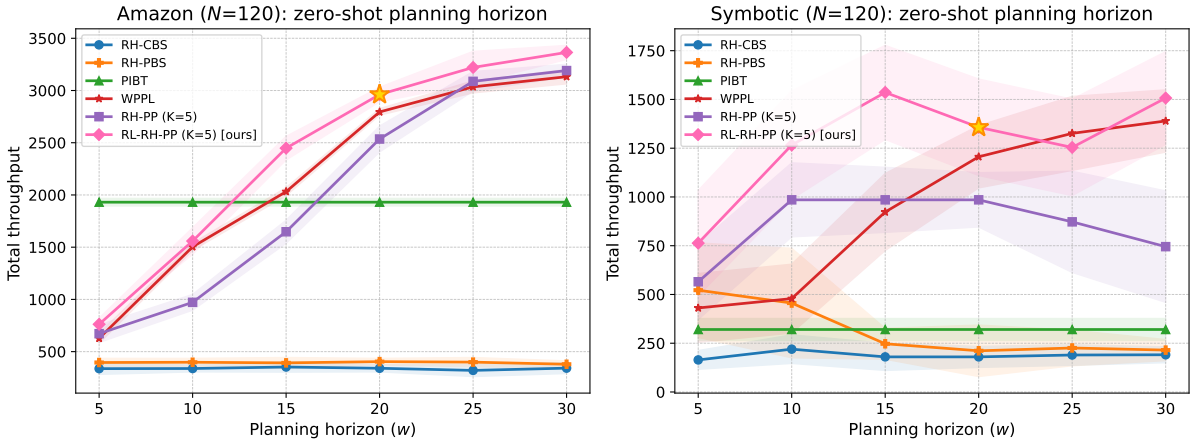

Fig. 9. Total throughput versus planning horizon w in zero-shot transfer evaluation at $N = 120$. The base model (yellow star) is trained at $N = 120$ and $w = 20$, and evaluated zero-shot at other w.

zero-shot generalization. The value of robust prioritization becomes increasingly evident with larger agent populations, suggesting that the learned policy captures congestion patterns that scale with crowding. WPPL serves as a strong baseline on both maps, and RL-RH-PP outperforms it under almost all settings. Overall, a policy trained only at $N = 120$ generalizes smoothly to both smaller and larger populations without any retraining.

5.4.2 Generalization across different planning windows. We next examine zero-shot transfer across planning horizons by fixing $N = 120$ and varying w at inference time, using the same base policy trained at $w = 20$. For comparability, RH-CBS, RH-PBS, RH-PP, and WPPL use the same window in each experiment. Figure 9 shows total throughput versus w.

As can be seen from the results, on Amazon maps, increasing w generally benefits all methods by providing more lookahead information. RL-RH-PP is competitive at small windows and remains at or near the top as the window grows, leveraging the longer preview to coordinate agents more proactively. RH-PP also improves with larger w, but the learned order generator usually yields additional gains over random orders, underscoring the advantage of learned global prioritization even when the planner sees further ahead. On Symbotic, longer windows are more consequential: RL-RH-PP capitalizes on larger w to anticipate corridor conflicts and relieve pressure in cross-aisles earlier within the horizon, surpassing baselines in the higher-w regime. These results indicate that a policy trained once at $w = 20$ adapts well to both smaller and larger windows at test time. Again, it is observed that as the planning horizon increases, RL-RH-PP is able to reverse the declining trend of RH-PP, providing further evidence of its ability to bypass inherent limitations of RH-PP for robust learning-guided optimization. We further evaluate mixed zero-shot transfer where both the agent population N and the planning window w vary simultaneously; the results are reported in the appendix A.

5.4.3 Remark on best configuration for each method. As observed in our experiments, for a given fixed map, each solver exhibits its peak performance in terms of throughput per bot at a specific value of N and w. The total throughput of the warehouse is computed as the throughput per bot multiplied by N. From a practical perspective of an autonomous warehouse service provider, it is essential to not only optimize throughput per agent but also to identify the ideal configuration of agent density and planning horizon to maximize overall operational throughput. To maximize the total throughput of the warehouse, we first identify the highest potential total

throughput achievable by each solver under different configurations (N, w) from experiments in Section 5.4.1 and Section 5.4.2, and then compare their performance against this. The results are summarized in Table 4.

Table 4. Highest total throughput of each methods under the corresponding configurations (N, w) (TP = throughput).

	Method	Total TP ↑	Optimal (N, w)
Amazon	RH-CBS	512 ± 83	(40,10)
	RH-PBS	2343 ± 30	(60,20)
	PIBT	1939 ± 53	(140,1)
	WPPL	3132 ± 68	(120,30)
	RH-PP	3170 ± 60	(120,30)
	RL-RH-PP	**3364** ± 67	(120,30)
Symbotic	RH-CBS	990 ± 165	(40,20)
	RH-PBS	1666 ± 105	(80,20)
	PIBT	477 ± 97	(60,1)
	WPPL	1389 ± 159	(120,30)
	RH-PP	1611 ± 57	(100,10)
	RL-RH-PP	**1740** ± 32	(100,10)

Our proposal RL-RH-PP achieves the highest total throughput in both warehouse maps, demonstrating its robustness and its ability to generalize effectively across different scenarios. By identifying the optimal configuration for each solver, our analysis provides practical insights into maximizing total throughput.

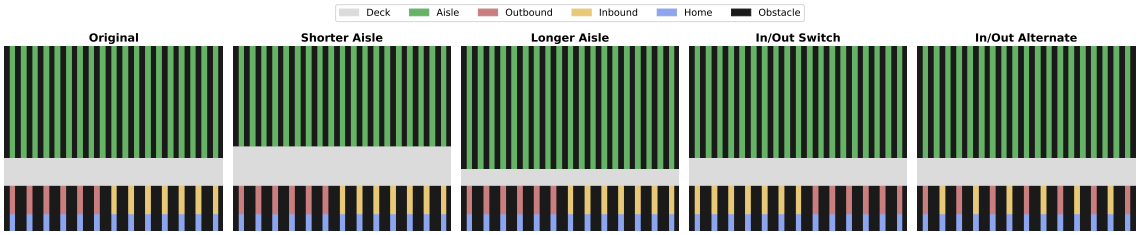

Fig. 10. Symbotic-style layout family: Original (training layout), Shorter Aisle (uniformly shortened aisles), Longer Aisle (uniformly lengthened aisles), In/Out Switch (inbound and outbound docks swapped), and In/Out Alternate (inbound/outbound docks interleaved along the perimeter). All share the same map size (W, H).

5.4.4 Generalization across different maps. It is also of interest to evaluate the extent to which the the trained policy generalizes across unseen map layouts. To probe cross-layout robustness, we derive four Symbotic-style variants from the base warehouse while preserving the same map size (W, H) so that absolute indices remain valid for our encoder. Those map variants are shown in Fig. 10. We choose Symbotic rather than Amazon because its higher obstacle density, narrower aisles, and more frequent intersections create stronger congestion externalities, yielding a stricter test for cross-layout robustness. Specifically, we include the following maps:

- Original: the unmodified training layout.
- Shorter Aisle: aisles uniformly shortened, increasing cross-aisle contention near intersections.
- Longer Aisle: aisles uniformly lengthened, creating longer corridors and queueing at choke points.
- In/Out Switch: inbound and outbound docks swapped to reverse macroscopic flow.

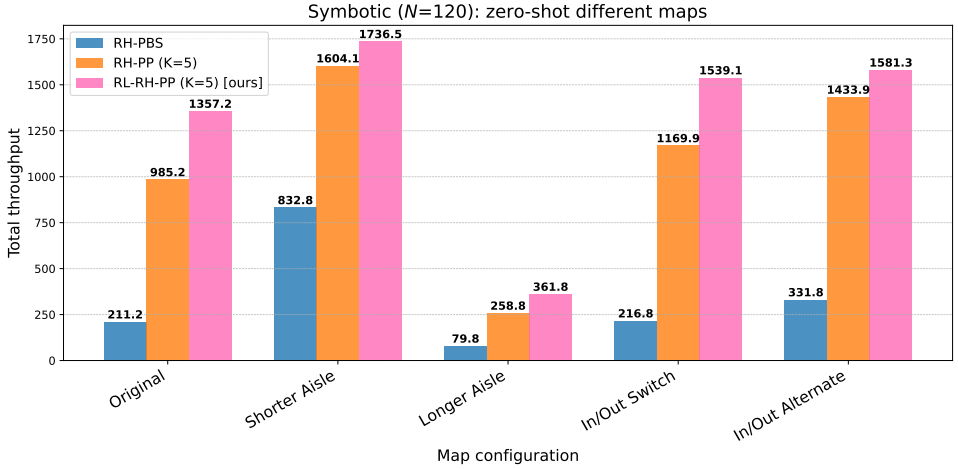

Fig. 11. Total throughput versus different Symbotic maps in zero-shot transfer evaluation at $N = 120$ and $w = 20$. The base model is trained at $N = 120$ and $w = 20$ on the Original map.

- In/Out Alternate: inbound/outbound docks interleaved to induce persistent opposing perimeter streams.

We choose a single base policy trained on the Original layout at $N = 120$ and $w = 20$ with fixed inference configuration, then evaluate it zero-shot on each variant. Figure 11 reports total throughput at $N = 120$ and $w = 20$ across the original and four variant maps. The zero-shot policy consistently exceeds RH-PP and significantly surpasses RH-PBS on every variant, with the largest gains on layouts that intensify congestion externalities (Longer Aisle, In/Out Alternate) and smaller but positive margins on layouts closer to the training geometry (Shorter Aisle, In/Out Switch). These results suggest that learning-guided prioritization provides useful cross-layout robustness within same-footprint reconfigurations, while full map-agnostic generalization (e.g., different map size) is left for future work. We also evaluate a mixed zero-shot setting where the agent population N varies across the different layout variants without retraining; results are provided in the appendix A.

5.4.5 Remark on why zero-shot transfer works. These experiments indicate that the policy, trained under a fixed configuration, generalizes effectively within the same map to different agent densities and planning horizons. A key reason is the position-embedding design: each drivable location has a trainable embedding, and the policy learns to interpret spatial and temporal relationships by attending over sequences of these embeddings along agents' paths. Because the map remains unchanged, the same embedding dictionary is reused at test time; varying N or w simply changes which sequences are presented, not the underlying representation. Coupled with attention layers that merge spatial and temporal information, this yields robustness to workload and horizon changes. Cross-map zero-shot transfer is harder with absolute embeddings and is addressed in our layout-variant study above; fully map-agnostic encoders are an avenue for future work.

5.5 Anytime Behavior of RL-RH-PP

With Top-K sampling, we can increase the number of sampled priority orders K to evaluate more candidates, making both RH-PP and RL-RH-PP anytime solvers. We first study their anytime behavior by examining how the final selected order depends on K. Specifically, we evaluate $K \in \{1, 5, 50, 200\}$ with a planning horizon $w = 5$ on both maps. We fix $w = 5$ to avoid excessive runtime at large K (e.g., $K = 200$). Figures 12 illustrate the relationship between K and throughput per agent (TPA) at $N = 100$, and we save the result for $N = 80$ in appendix B.

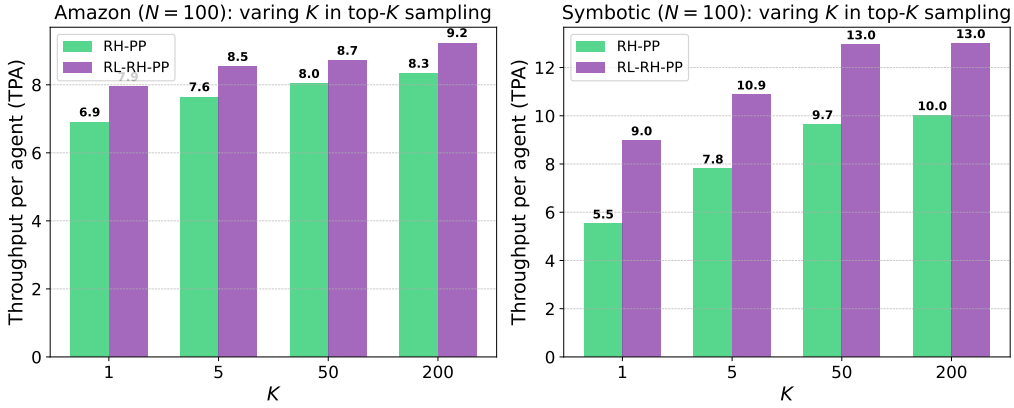

Fig. 12. Throughput per agent (TPA) at evaluation vs K, evaluated with $N = 100$

Table 5. Per priority order runtime breakdown, evaluated at Symbotic map with $N = 100$

Component	Time (ms)
Priority order generation (network forward pass)	32
Priority order evaluation (run prioritized planning)	46

When only a single priority order is sampled ($K = 1$), RL-RH-PP consistently outperforms RH-PP across all settings. This is expected, as the RL policy is trained by sampling a single priority order ($K = 1$), thereby specializing in generating high-quality solutions using that order. As K increases, by leveraging learned autoregressive decoding strategy, RL-RH-PP effectively shrinks the search space for finding optimal priority orders,enabling more efficient planning compared to naive, uninformed sampling. We also observe an improvement in throughput per agent for both methods as K increases, demonstrating that performance can be further enhanced by exploring a broader set of priority orders.

We next quantify the runtime–quality trade-off across methods on each map with. For fairness, we sweep one knob per method that primarily controls compute: the number of sampled orders K for RH-PP/RL-RH-PP, the LNS time budget for WPPL, and the search-time budget for RH-CBS/RH-PBS. For every configuration, we sweep the computational budget from {0.5s, 1.0s, 1.5s, 2.0s, 2.5s} CPU time, and we report average total throughput and average end-to-end runtime per planning step on identical hardware. Fig. 13 show that increasing K yields predictable anytime behavior for RL-RH-PP. Moreover, RL-RH-PP contributes competitive points to the frontier alongside baselines, illustrating that learning to propose priority orders provides favorable quality at practical runtimes.

Our current implementation of Top-K evaluation is executed serially on a CPU, leading to a linear increase in computation time with K. Table 5 reports the average time to (i) generate a priority order with the trained policy in inference mode and (ii) evaluate that order by running prioritized planning within RL-RH-PP. We have moved order generation to a batched, GPU-parallel implementation, so for large K the dominant cost is the CPU-bound Top-K evaluation. For example, at $K = 200$, generating all orders takes just over 32 ms, whereas evaluating them takes about 9 s. Future work could optimize solving efficiency by leveraging multi-threading techniques to parallelize Top-K evaluation, significantly reducing computation overhead and improving real-time applicability.

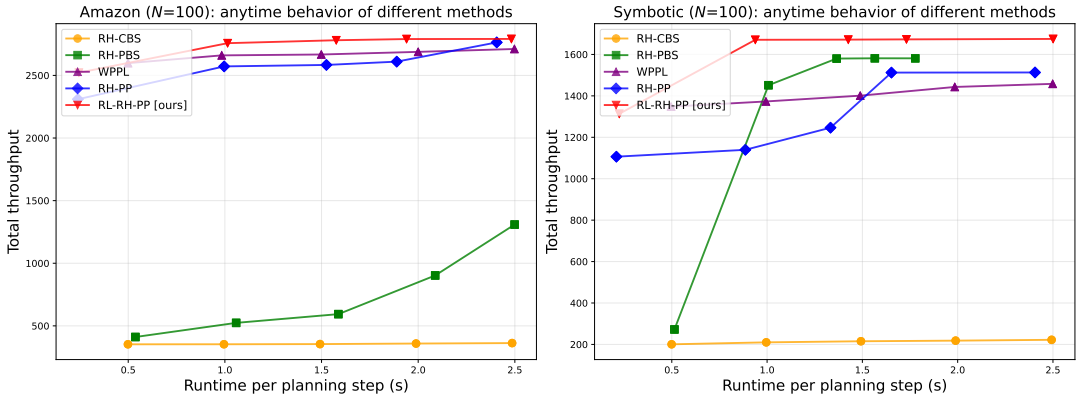

Fig. 13. Total throughput vs runtime budget of all methods, evaluated with $N = 100$ and $w = 20$

5.6 Ablation Study

In this section, we assess the effectiveness of our proposal through a series of ablation studies.

5.6.1 Effects of the feasibility handling schemes of the priority order. Prioritized planning is not complete, hence a selected priority order can occasionally yield an infeasible initial plan. As described in Section 4, two approaches are adopted. First, within the RH-PP backbone, a heuristic scoring function guides selection among the Top-K sampled orders. Second, penalties are incorporated into the reward design of RL-RH-PP training to encourage the generation of feasible orders. We now ablate the effects of both the above designs.

We first study the effect of the infeasibility heuristic measure in the Top-K selector. Recall that in Eq. 1, RH-PP evaluates each sampled total order $\prec_k$ by $\text{cost}(\prec_k) = \sum_{i=1}^{N} \left(e_i + \beta\, s_i\right)$, where e_i is the initial path length under $\prec_k$, $s_i \in \{0, 1\}$ indicates whether agent i was forced onto its shortest path (making the initial solution infeasible), and β penalizes such infeasibility. We study how the infeasibility penalty β affects the quality of the final selected priority order. We sweep $\beta \in \{0, 1, 10, 100\}$ for both RH-PP and RL-RH-PP. Because the order selection in Eq. 1 is applied only at inference, we do not retrain the RL policy for each β. We set $K = 10$ to include more priority orders under consideration to create more challenging short preview horizon life-long planning, while holding all other settings fixed as the experiments in Table 2. We report throughput per agent (TPA) under $N = 100$ in both maps in the Figures. 14, while saving the result for $N = 80$ in appendix B. We observe a clear upward trend in TPA as β increases for both RH-PP and RL-RH-PP. A larger β raises the cost of candidate orders in which many agents are forced onto their unplanned shortest paths, so the selector prefers orders with fewer such agents. This choice reduces downstream repair, early conflicts, and wait-induced detours, yielding higher TPA.

Next, we study whether our penalty in the reward term truly boosts the feasibility of the learned policy. We quantify this by the percentage of planning steps whose selected order is infeasible (lower is better), and report the values in Table 6. For comparability, we report results from the same experiments in Table 2. Across maps and agent densities, RL-RH-PP yields a lower infeasibility rate than RH-PP, consistent with the reward term that penalizes infeasibility (the $\sigma s_{i,t}$ in Eq. 2).

5.6.2 Effect of the penalty weights in the reward function. We have weighted reward terms in our reward function (Eq. 2), and now we study the effect of these penalty weights on the training process. Recall that the designed reward function (Eq. 2) used for RL training is as follows: $R(\boldsymbol{o}_t, \boldsymbol{a}_t) = -\frac{1}{N}\left(\sum_{i=1}^{N} d_{i,t} \;+\; \kappa\, c_{i,t} \;+\; \sigma\, s_{i,t}\right)$, where

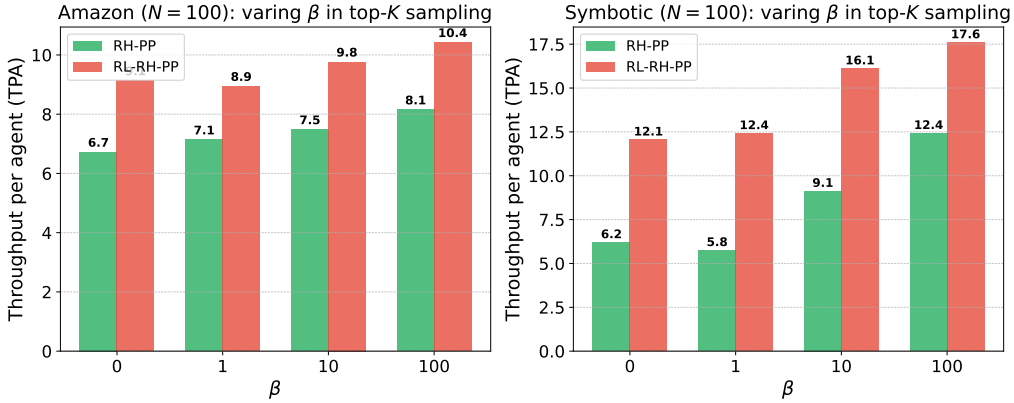

Fig. 14. Throughput per agent (TPA) at evaluation vs β, evaluated with $N = 100$

Table 6. Percentage of steps where the selected order yields an infeasible initial plan (lower is better), tracked from same experiments in Table 2, 'Ama.' for Amazon and 'Sym.' for Symbotic.

	Method	$N=80$	$N=100$	$N=120$
Ama.	RH-PP ($K = 5$)	8.1% ± 2.1%	39% ± 5.1%	88% ± 4.3%
	RL-RH-PP ($K = 5$)	**7.8%** ± 1.9%	**28%** ± 5.0%	**60%** ± 6.2%
Sym.	RH-PP ($K = 5$)	83% ± 3.9%	97% ± 1.1%	99% ± 0.1%
	RL-RH-PP ($K = 5$)	**34%** ± 4.8%	**84%** ± 3.5%	**97%** ± 0.3%

$d_{i,t}$ is the average Manhattan distance from agent i to its queued goals, $c_{i,t} \in \{0, 1\}$ indicates that agent i has all-wait actions for the next h steps (i.e., congested), and $s_{i,t} \in \{0, 1\}$ indicates that no feasible path can be found for agent i under the current priority. κ and σ are weighting factors. We study how the congestion weight κ and the infeasibility weight σ affect the total throughput of the trained RL-RH-PP policy. Because these weighting factors shape the training objective, we re-train RL-RH-PP from scratch for $\kappa \in \{0, 100, 1000, 10000\}$ while fixing $\sigma = 1000$, and for $\sigma \in \{0, 100, 1000, 10000\}$ while fixing $\kappa = 1000$, under identical training and evaluation settings as the experiments for $N = 120$ in Table 2. Our primary metric is total throughput. To stress-test decongestion behavior and make the effect of κ and σ most visible, we run this ablation on the Symbotic map, which induces sustained congestion and frequent aisle conflicts. As shown in Fig. 15, increasing either weight generally improves both learning speed and the final throughput plateau up to a broad optimum around 1000. In the congestion sweep, $\kappa = 0$ underperforms, $\kappa = 100$ improves stability, and $\kappa = 1000$ attains the best overall curve, while $\kappa = 10000$ yields no further gains. In the infeasibility sweep, the trend is more pronounced: performance is more sensitive to σ, with $\sigma = 0$ lagging, $\sigma = 100$ narrowing the gap, $\sigma = 1000$ achieving the highest plateau, and $\sigma = 10000$ offering little additional benefit. These results indicate that both penalties are useful for throughput in dense regimes, that the infeasibility weight is the more sensitive knob, and that excessively large values are unnecessary. We therefore set $\kappa = 1000$ and $\sigma = 1000$ in our main experiments.

5.6.3 Effects of the temporal and spatial attention. To isolate the contribution of the encoder's temporal and spatial attention design, we perform the following ablation experiments by training four encoder variants from scratch under identical budgets, seeds, and inference settings (decoder unchanged): (i) Full (ours), (ii) w/o Temporal

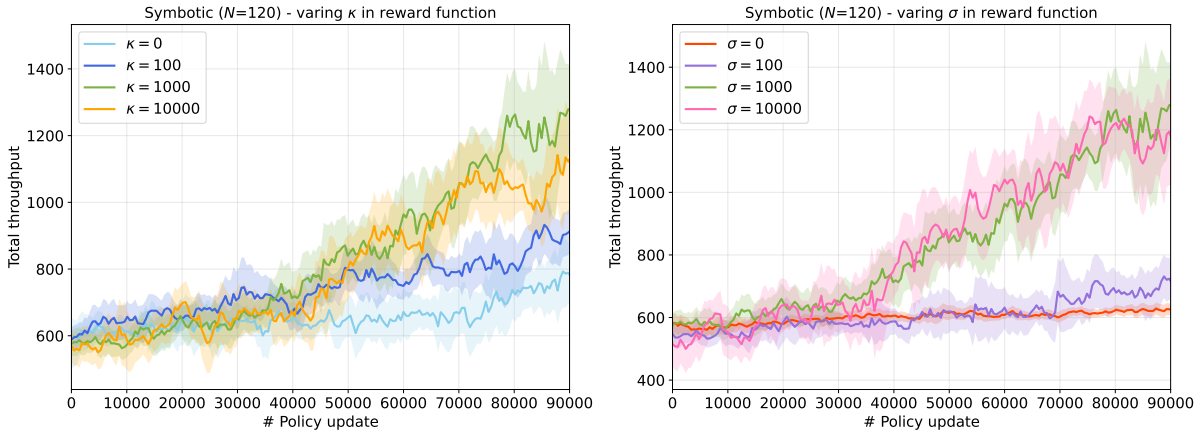

Fig. 15. Total throughput during training under different congestion (κ) and infeasibility (σ) weights in reward function Eq. 2, evaluated on-the-fly with $N = 120$

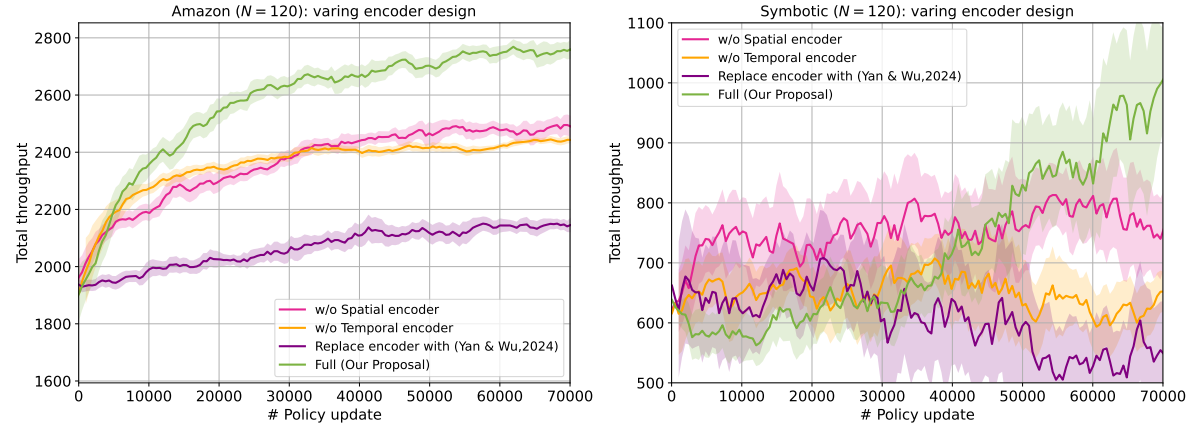

Fig. 16. Total throughput during training for encoder ablation study, evaluated on-the-fly with $N = 120$

encoder (temporal attention replaced by MLPs), (iii) w/o Spatial encoder (spatial attention replaced by MLPs), and (iv) Replace the entire encoder with the architecture of Yan & Wu (Yan and Wu, 2024) (intra-path attention + 3D conv). We report total throughput during training on Amazon and Symbotic at N=120, as shown in Fig. 16.

Across both maps, removing either attention block lowers learning speed and the final plateau relative to the Full model, confirming that both temporal and spatial attention are beneficial. On Amazon, the differences are modest: dropping temporal attention slows down early learning and ends slightly below Full, while dropping spatial attention yields a similarly small decrease—consistent with the map's wider corridors and fewer bottlenecks. On Symbotic, however, the contrast is sharper. Without temporal attention, performance degrades substantially, highlighting the need to capture long-range interactions to resolve aisle congestion. Removing spatial attention also hurts, though less dramatically, suggesting that both components matter but temporal attention is particularly critical in constrained, high-density environments.

Replacing the entire encoder with Yan & Wu's (Yan and Wu, 2024) yields consistently lower throughput on Amazon and fails to learn a competitive policy on Symbotic. We attribute this drop to the limitations of 3D convolution, which emphasizes local interactions within a fixed receptive field: it captures short-range dependencies but struggles with global agent–agent interactions needed for long-term coordination. In contrast, our spatial attention computes agent-wise weights over the entire frame, enabling global context and more informed decisions, which improves coordination and overall throughput in lifelong MAPF.

5.6.4 Compare with other priority order assignment heuristics. To demonstrate the effectiveness of our learning-guided heuristic for priority ordering, we compare the priority orders sampled from Top-K sampling in RL-RH-PP against those constructed using rule-based methods. Specifically, we evaluate the distance-query heuristic used in Berg and Overmars (2005); Ma et al. (2019), where an agent's priority is determined by the length of its shortest path from its current location to its goal. The intuition behind this heuristic is to minimize the maximum arrival time by prioritizing agents with longer paths, allowing them to move unhindered, while those with shorter paths can afford delays to yield to higher-priority agents. To integrate this heuristic into RH-PP, we replace the Top-K sampling mechanism with a direct priority assignment based on the distance heuristic, constructing a total priority order at each planning step. We refer to this ablation variant as Distance Query Rolling Horizon Prioritized Planning (DQ-RH-PP).

We do not include Zhang et al. (2022) as a baseline because, in the one-shot setting it targets, its learned ordering underperforms the distance-query heuristic (Berg and Overmars, 2005; Ma et al., 2019) reported in the same literature. To provide a stronger and directly applicable comparator under our rolling-horizon protocol, we evaluate the DQ-based variant, DQ-RH-PP (Table 7).

Table 7. Compare with other priority order assignment heuristics ($w = 20$). 'Ama.' for Amazon and 'Sym.' for Symbotic.

	Method	$N=80$		$N=100$		$N=120$	
		TPA ↑	**Time(s)** ↓	**TPA** ↑	**Time(s)** ↓	**TPA** ↑	**Time(s)** ↓
Ama.	DQ-RH-PP	27.23 ± 0.77	0.17	20.89 ± 1.64	0.25	17.66 ± 0.56	0.48
	RL-RH-PP ($K = 5$)	**31.80** ± 0.46	0.65	**28.84** ± 0.35	0.79	**25.76** ±0.55	0.96
Sym.	DQ-RH-PP	17.33 ± 0.56	0.15	14.01 ± 2.31	0.22	9.88 ± 0.74	0.43
	RL-RH-PP ($K = 5$)	**18.38** ± 0.50	0.68	**15.38** ± 1.85	0.82	**11.31** ± 2.21	0.99

Across all settings, we observe that RL-RH-PP consistently outperforms DQ-RH-PP in terms of throughput per agent. This highlights the effectiveness of our proposed Top-K sampling, demonstrating that the constrained priority order space contains high-quality solutions that can be efficiently retrieved through multiple trials of sampling. These findings suggest that relying solely on a fixed-rule heuristic to construct priority orders may not be an optimal approach for achieving efficient coordination.

5.6.5 Replace RL with contextual bandit formulation. To further demonstrate the contribution of multi-step planning, we conduct an ablation study in which the RL agent is trained with a horizon of one step - essentially a contextual bandit formulation. This simplified setup removes the influence of long-term future planning, allowing us to evaluate the performance of the encoder/decoder architecture on its own. As shown in Figure 17, the contextual bandit variant converges more quickly, however, its ultimate throughput is lower compared to our RL-RH-PP. This suggests that while the encoder/decoder effectively captures context, the ability to plan over long-horizon using RL is crucial for managing congestion and achieving higher overall performance.

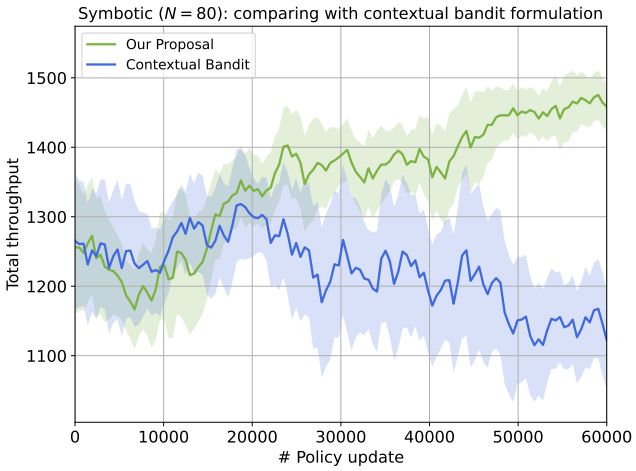

Fig. 17. Total throughput during training for contextual bandit ablation study, evaluated on-the-fly with $N = 80$

6 What Does RL Learn?

In this section, we take a closer look at specific planning steps and visually analyze the step-wise priority order sampling of RL-RH-PP to interpret what RL has learned. To this end, we analyze both what the policy learns and how it leverages that knowledge to improve planning performance. First, we investigate the priority assignment patterns using heatmaps over time, revealing how RL-RH-PP adapts to congestion through strategic prioritization. Then, we examine the policy's ability to recover from congested states, comparing movement directions towards next goals derived from shortest paths with those chosen by the RL policy.

6.1 Priority Heat Maps

We conduct a step-wise analysis of the tasks completed every h execution steps during simulation in a single evaluation environment for an RL-RH-PP policy trained on Symbotic map with $N = 100$ and $w = 20$, as illustrated in Figure 18. Our observations show that RL-RH-PP consistently maintains a high task-completion rate throughout the simulation. In contrast, RH-PP initially achieves comparable performance but suffers a rapid decline as the simulation progresses and congestion accumulates. This decline suggests that suboptimal priority assignments lead to premature commitments, thereby generating long-term conflicts or deadlocks that hinder effective solutions in subsequent planning steps. Note that the planning step is defined as where the replanning happens. In our setting with $T = 800$ and $h = 5$, there are a total of 160 planning steps.

To gain further insight into what the RL policy has learned, we pause the simulation at planning steps 0, 30, and 100 (vertical dash lines in Figure 18), sampled 100 priority orders from both RH-PP and RL-RH-PP, and compute the average priority assignment for each agent across these samples. As shown in Figure 19, the lighter shades represent higher priority assignments.

As expected, priority assignments from RH-PP (first row in Figure 19) are purely random, resulting in mid-level average priorities across agents. This randomness aligns with the fact that RH-PP samples priorities uniformly, inherently having higher entropy. Additionally, at planning step 100, RH-PP demonstrates noticeable congestion.

Interestingly, as depicted in the second row of Figure 19, RL-RH-PP has an order sampling distribution with lower entropy, and RL-RH-PP strategically assigns higher priorities to agents located in congested regions as the

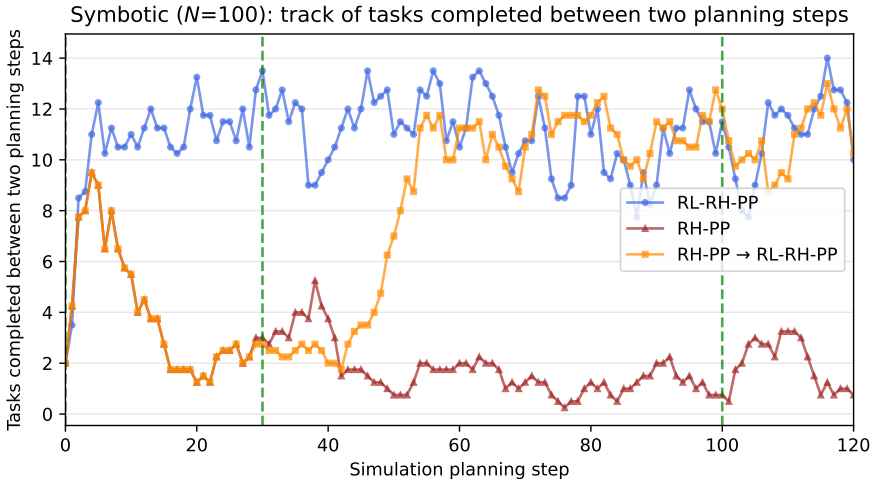

Fig. 18. Number of tasks completed during simulation (Symbotic map, $N = 100$). We measure how many tasks are completed between consecutive planning steps. The green dashed vertical lines highlight the key planning steps 0, 30, and 100 shown in Figure 19. In the RH-PP → RL-RH-PP curve, we first use RH-PP for planning and switch to RL-RH-PP at planning step 30.

simulation progresses (steps 30 and 100). This adaptive prioritization implies that the RL policy has implicitly learned to recognize and proactively manage potential bottlenecks. By assigning higher priorities to agents within congested areas, RL-RH-PP effectively mitigates conflict propagation, reducing the likelihood of deadlocks and improving overall system throughput. This learned behavior is crucial, as it directly addresses congestion at its onset, providing long-term stability and more efficient resource utilization during subsequent planning stages.

From Figure 18 and the first row of Figure 19, we observe that RH-PP begins to accumulate congestion around planning step 30. To investigate whether the RL policy can effectively recover from such congested states, we conduct an experiment where RH-PP is used for the first 29 planning steps, followed by switching to RL-RH-PP for the remaining steps starting from planning step 30. This setup corresponds to the RH-PP → RL-RH-PP curve in Figure 18 and the third row of Figure 19. Interestingly, we find that task completion per planning step starts to improve shortly after the switch, indicating that the congestion is being alleviated. Moreover, the heatmap at planning step 100 in the third row shows a significantly reduced level of congestion compared to RH-PP alone (first row), further confirming this recovery. These results suggest that the learned RL policy is capable of proactively managing congestion and effectively resolving severe deadlocks when inherited from suboptimal priority planning, despite not being explicitly trained to do so.

6.2 How Does RL-RH-PP Recover from Congestion

The evidence of RL-RH-PP successfully recovering from the congestion built up by RH-PP, as discussed in Section 6.1, is fascinating. To gain a deeper understanding of how the RL policy facilitates this recovery, we analyze the behavior of RL-RH-PP at planning step 45-the point at which the RH-PP → RL-RH-PP curve in Figure 18 begins to rise, indicating that congestion is starting to resolve. Specifically, we compare the hypothetical next movement direction toward each agent's next goal, derived from their shortest paths, with the actual next movement direction selected by RL-RH-PP. These directions are visualized as arrows in Figure 20.

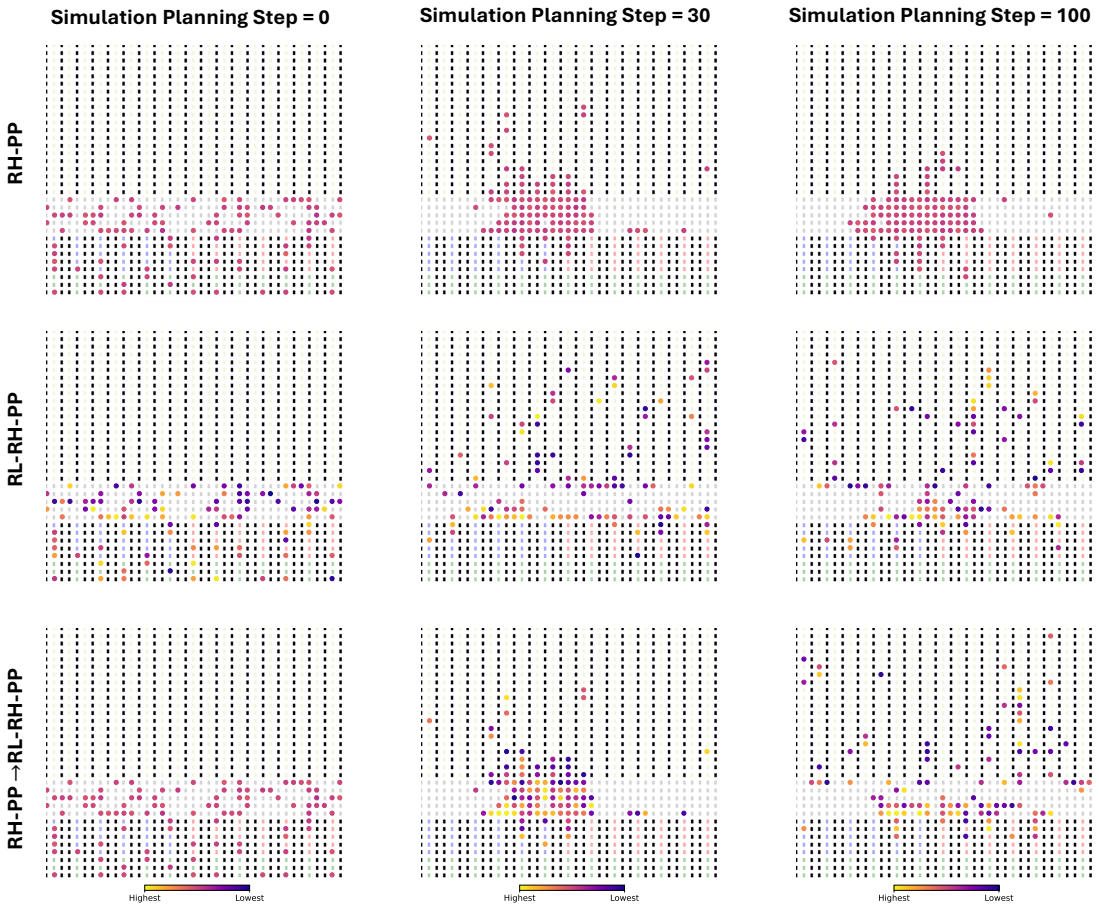

Fig. 19. Priority heatmaps for three planning methods (rows) at different steps (columns): 0, 30, and 100, as shown in Figure 18. Agents are shown in dots. The top row shows RH-PP, which assigns uniformly random priorities and experiences severe congestion by step 100. The middle row shows RL-RH-PP, which adaptively assigns higher priorities for agents in congested regions, sustaining high throughput. The bottom row shows switching from RH-PP (planning steps 0–29) to RL-RH-PP at step 30, effectively resolving inherited congestion by step 100. Lighter colors indicate higher priority assignments.

Interestingly, we observe a key behavioral shift in the RL-guided planning. For agents located near the boundary of the congested region (highlighted in the blue bounding boxes), we see that the actual planned direction given by RL-RH-PP tends to be counterintuitive - away from the hypothetical shortest-path directions which guide agents to the goals. This “backtracking” behavior emerges most clearly when agents deeper within the congestion must pass through a narrow aisle that is currently blocked (see the vertical blue bounding box). By recognizing that such agents become space-time obstacles for those agents closer to the aisle entrance, RL-RH-PP proactively repositions these boundary agents by assigning them lower priorities to clear the route. Specifically, RL-RH-PP may temporarily reverse their direction, opening a passage so that the congested agent can advance without colliding or creating a bottleneck. Once the way is cleared, these boundary agents resume their forward progress

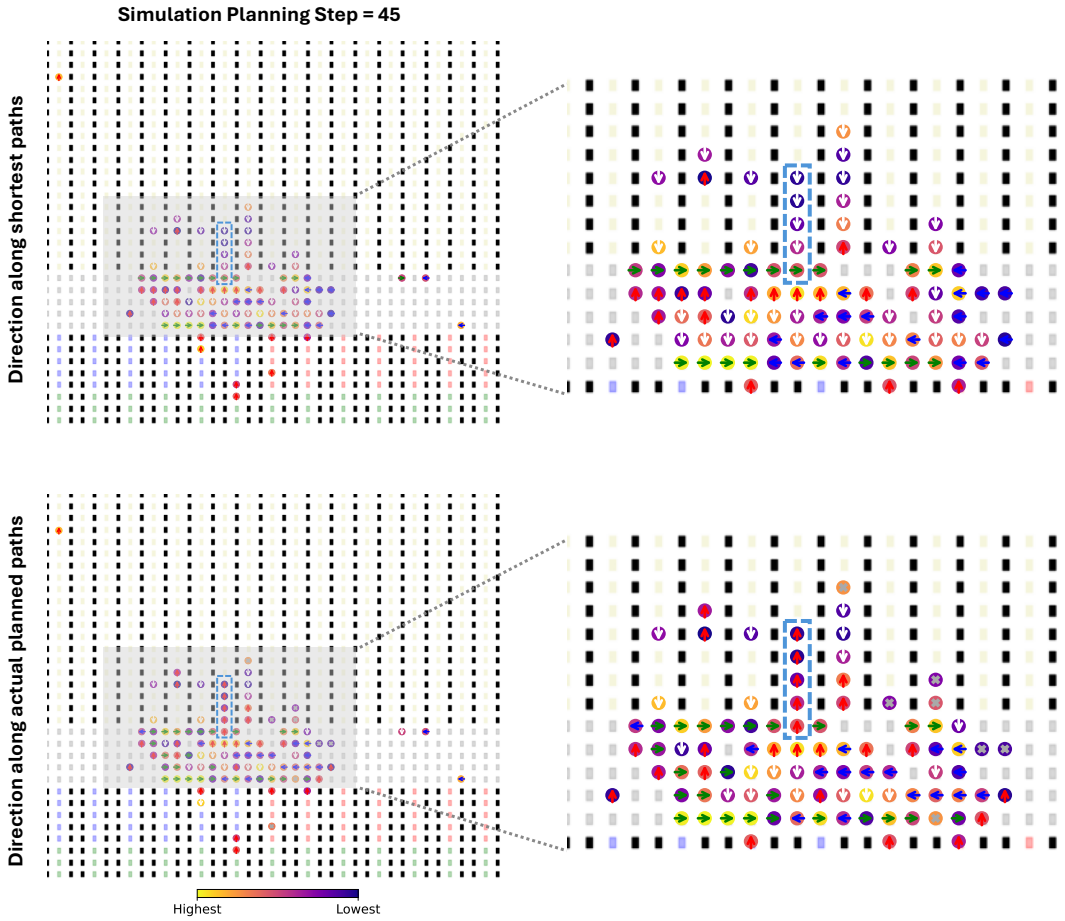

Fig. 20. Next movement directions along shortest path and the actual planned path of RH-PP → RL-RH-PP curve at planning step = 45 in Figure 18. The blue bounding boxes highlight the agents located near the boundary of the congested region. Different ordinal directions are shown in different colors, the cross symbol indicates *wait* action.

toward their original goals. Such behavior demonstrates that the RL policy has learned to coordinate local detours that reduce global conflict, enabling more efficient resolution of congestion and improving overall throughput.

To further clarify the mechanism behind RL-RH-PP's recovery from congestion, consider a hypothetical grid world example in Figure 21. In this scenario, agents are positioned in a congested desk with a narrow aisle. We specifically focus on agents 1 and 2. Initially, both agents appear to be heading toward their respective goals. The RL-RH-PP policy assigns a higher priority to agent 2, which is located deeper within the congested area, and a lower priority to agent 1, positioned near the aisle's exit. Under such a priority assignment, although agent 1 is closer to its destination, its action is not to move forward but to momentarily backtrack. By stepping backward, agent 1 clears the path for other congested agents (blue color) and lets the high-priority agent 2 pass, effectively

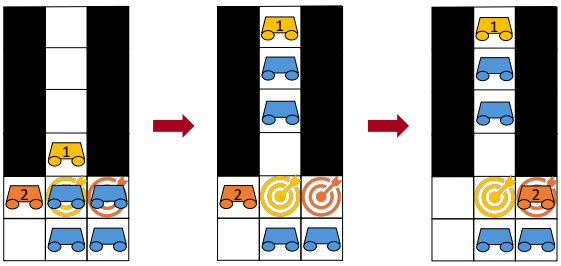

Fig. 21. Illustrative grid world example of RL-RH-PP congestion recovery. It shows a deck and a narrow aisle with agents labeled "1" (lower priority) and "2" (higher priority), and their goals. As a result, agent 1 backtracks to clear the path, enabling agent 2 to move forward. This temporary detour mitigates congestion and prevents deadlocks, thereby improving throughput.

turning itself from a potential blockage into an enabler of smoother flow. Once agent 2 passes the aisle's exit, agent 1 can resume its forward progress, and the deadlock is resolved.

7 Conclusion

In this paper, we introduced RL-guided Rolling Horizon Prioritized Planning (RL-RH-PP), the first framework that integrates reinforcement learning with a classical search-based planner for learning-guided optimization of lifelong MAPF. Our approach builds upon a rolling-horizon extension of Prioritized Planning (RH-PP) to continuously replan in high-density warehouse environments. By casting dynamic priority ordering as a Partially Observable Markov Decision Process, we train a transformer-based neural network that autoregressively decodes high-quality priority orders, effectively reducing the search space for the single-agent path solver.

Through extensive experiments in two large-scale warehouse environments, inspired by Amazon and Symbotic, we observe that integrating RL into the RH-PP backbone delivers an average 25% improvement across both simulators, which in turn enables RL-RH-PP to achieve higher throughput than conventional lifelong MAPF baselines. Notably, RL-RH-PP consistently outperforms RH-PP with random sampling, and showcases unique benefits over strong baselines such as RH-CBS, RH-PBS, PIBT, and WPPL, especially in highly constrained scenarios. Moreover, our learned policy exhibits strong zero-shot generalization across varying agent densities, planning horizons and warehouse layouts. This adaptability is facilitated by a learnable dictionary-based position embedding and attention-based neural architecture that captures both spatial and temporal inter-agent dependencies. We additionally report thorough ablation studies verifying the importance of our core design choices. Finally, a case study demonstrates that the RL component learns effective congestion-avoidance policies and can adaptively recover feasible plans when congestion has already arisen under RH-PP. This evidences a unique and potentially impactful practical value of our proposed learning-guided optimization approach in handling long-term dynamics, particularly in complex systems such as warehouse automation.

Our work also highlights the promise of learning-based approaches that complement, rather than replace, established search-based solvers. Our RL-RH-PP effectively capitalizes on the strengths of both data-driven decision-making and efficient single-agent path searches. Moreover, some experiments suggest that RL-RH-PP can reverse the performance decline of RH-PP under challenging conditions, indicating that our approach may help address inherent limitations of the backbone solver to some extent, which is likely due to the ability of reinforcement learning to automate data-driven coordination and prioritization. These findings underscore the potential for learning-guided hybrid methods to achieve state-of-the-art performance in the complex, dynamic, and fast-evolving real-world applications such as multi-robot coordination in warehouse automation.

One promising avenue for future research is to scale our approach to even larger warehouses with thousands of agents by parallelizing the evaluation of multiple candidate priority orders. While Top-K sampling can yield high-quality solutions, it may increase computational overhead. Efficient, parallelizable planning engines could mitigate this cost, making RL-RH-PP feasible for real-time operation with thousands of robots. Meanwhile, our encoder indexes absolute map locations. Such a representation design allows zero-shot transfer to layout variation with fixed map size, but precludes zero-shot transfer to layouts with different map sizes. To broaden applicability, future work will pursue full map-agnostic generalization state representations, training regimes that encourage cross-map-size generalization. We also note that our neural architecture for generating priority orders could, in principle, be applied to the one-shot MAPF setting; one might design an alternative learning paradigm to train it for producing effective static orders, though we leave this as a separate line of research beyond the scope of this paper. Moreover, our framework naturally extends to more complex warehouse tasks. For example, jointly optimizing task assignment and path planning could yield a fully integrated solution for maximizing throughput. Concretely, the autoregressive decoder can emit a sequence of (agent, task) pairs rather than an agent-only priority order. At each decoding step, we apply masks to enforce feasibility (each task assigned at most once, agent queue/capacity limits, task release/eligibility), append the selected task to the agent's queue, and continue until a stopping criterion is met (e.g., budgeted number of pairs or no feasible pairs). The resulting partial assignment for the window is then executed by the same RH-PP planner. Training can optimize a combined throughput/tardiness reward. Incorporating robustness to model uncertainties, such as stochastic delays or partial observability, would further enhance real-world applicability. Finally, we believe it is promising to explore similar learning-guided optimization frameworks for addressing long-horizon dynamics in broader classes of optimization and decision-making problems.

Acknowledgments

This research was supported by Symbotic and a Fundamental Research grant from the National Science Foundation and AI Singapore (NSF-AISG).

References

Dor Atzmon, Roni Stern, Ariel Felner, Glenn Wagner, and neng-fa Zhou. 2020. Robust Multi-Agent Path Finding and Executing. *Journal of Artificial Intelligence Research* 67 (03 2020), 549–579. https://doi.org/10.1613/jair.1.11734

K. Azarm and G. Schmidt. 1996. A decentralized approach for the conflict-free motion of multiple mobile robots. In *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems. IROS '96*, Vol. 3. 1667–1675 vol.3. https://doi.org/10.1109/IROS.1996.569036

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. 2016. Layer normalization. *arXiv preprint arXiv:1607.06450* 1 (2016).

Max Barer, Guni Sharon, Roni Stern, and Ariel Felner. 2014. Suboptimal variants of the conflict-based search algorithm for the multi-agent pathfinding problem. In *Seventh Annual Symposium on Combinatorial Search*.

Maren Bennewitz, Wolfram Burgard, and Sebastian Thrun. 2002. Finding and Optimizing Solvable Priority Schemes for Decoupled Path Planning Techniques for Teams of Mobile Robots. *Robotics and Autonomous Systems* 41 (01 2002), 89–99. https://doi.org/10.1016/S0921-8890(02)00256-7

J.P. Berg and M.H. Overmars. 2005. Prioritized motion planning for multiple robots. *2005 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS*, 430 – 435. https://doi.org/10.1109/IROS.2005.1545306

Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. 2016. OpenAI Gym. arXiv:arXiv:1606.01540

Shao-Hung Chan, Zhe Chen, Teng Guo, Han Zhang, Yue Zhang, Daniel Harabor, Sven Koenig, Cathy Wu, and Jingjin Yu. 2024. The League of Robot Runners Competition: Goals, Designs, and Implementation. In *ICAPS 2024 System's Demonstration track*. https://openreview.net/forum?id=mPmCnEHTvJ

Larissa Custodio and Ricardo Luiz Machado. 2020. Flexible automated warehouse: a literature review and an innovative framework. *The International Journal of Advanced Manufacturing Technology* 106 (01 2020), 1–26. https://doi.org/10.1007/s00170-019-04588-z

Mehul Damani, Zhiyao Luo, Emerson Wenzel, and Guillaume Sartoretti. 2021. PRIMAL$_2$: Pathfinding Via Reinforcement and Imitation Multi-Agent Learning - Lifelong. *IEEE Robotics and Automation Letters* 6, 2 (April 2021), 2666–2673. https://doi.org/10.1109/lra.2021.3062803

Boris De Wilde, Adriaan W. Ter Mors, and Cees Witteveen. 2014. Push and rotate: a complete multi-agent pathfinding algorithm. *Journal of Artificial Intelligence Research* 51, 1 (Sept. 2014), 443–492.

Michael Erdmann and Tomas Lozano-Perez. 1987. On multiple moving objects. *Algorithmica* 2 (1987), 477–521.

Alessando Farinelli, Nicolo Boscolo, Elena Zanotto, and Enrico Pagello. 2016. Advanced approaches for multi-robot coordination in logistic scenarios. *Robotics and Autonomous Systems* 90 (09 2016). https://doi.org/10.1016/j.robot.2016.08.010

Peter E Hart, Nils J Nilsson, and Bertram Raphael. 1968. A formal basis for the heuristic determination of minimum cost paths. *IEEE transactions on Systems Science and Cybernetics* 4, 2 (1968), 100–107.

Chengyang He, Tanishq Duhan, Parth Tulsyan, Patrick Kim, and Guillaume Sartoretti. 2024. Social Behavior as a Key to Learning-based Multi-Agent Pathfinding Dilemmas. arXiv:2408.03063 [cs.RO] https://arxiv.org/abs/2408.03063

Chengyang He, Tianze Yang, Tanishq Duhan, Yutong Wang, and Guillaume Sartoretti. 2023. ALPHA: Attention-based Long-horizon Pathfinding in Highly-structured Areas. arXiv:2310.08350 [cs.RO] https://arxiv.org/abs/2310.08350

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 770–778.

Taoan Huang, Sven Koenig, and Bistra Dilkina. 2021. Learning to resolve conflicts for multi-agent path finding with conflict-based search. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 35. 11246–11253.

He Jiang, Yutong Wang, Rishi Veerapaneni, Tanishq Duhan, Guillaume Sartoretti, and Jiaoyang Li. 2025. Deploying Ten Thousand Robots: Scalable Imitation Learning for Lifelong Multi-Agent Path Finding. arXiv:2410.21415 [cs.MA] https://arxiv.org/abs/2410.21415

He Jiang, Yulun Zhang, Rishi Veerapaneni, and Jiaoyang Li. 2024. Scaling Lifelong Multi-Agent Path Finding to More Realistic Settings: Research Challenges and Opportunities. arXiv:2404.16162 [cs.MA] https://arxiv.org/abs/2404.16162

Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).

Wouter Kool, Herke Van Hoof, and Max Welling. 2019. Attention, learn to solve routing problems! *International Conference on Learning Representations* (2019).

Jiaoyang Li, Zhe Chen, Daniel Harabor, P Stuckey, and Sven Koenig. 2021a. Anytime multi-agent path finding via large neighborhood search. In *International Joint Conference on Artificial Intelligence (IJCAI)*.

Jiaoyang Li, Zhe Chen, Daniel Harabor, Peter J. Stuckey, and Sven Koenig. 2022. MAPF-LNS2: Fast Repairing for Multi-Agent Path Finding via Large Neighborhood Search. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*. 10256–10265. https://doi.org/10.1609/aaai.v36i9.21266

Jiaoyang Li, Wheeler Ruml, and Sven Koenig. 2020a. EECBS: A Bounded-Suboptimal Search for Multi-Agent Path Finding. *CoRR* abs/2010.01367 (2020). arXiv:2010.01367 https://arxiv.org/abs/2010.01367

Jiaoyang Li, Andrew Tinka, Scott Kiesel, Joseph W. Durham, T. K. Satish Kumar, and Sven Koenig. 2020b. Lifelong Multi-Agent Path Finding in Large-Scale Warehouses. *CoRR* abs/2005.07371 (2020). arXiv:2005.07371

https://arxiv.org/abs/2005.07371

Jiaoyang Li, Andrew Tinka, Scott Kiesel, Joseph W Durham, TK Satish Kumar, and Sven Koenig. 2021b. Lifelong multi-agent path finding in large-scale warehouses. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 35. 11272–11281.

Minghua Liu, Hang Ma, Jiaoyang Li, and Sven Koenig. 2019. Task and path planning for multi-agent pickup and delivery. In *Proceedings of the International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*.

Hang Ma, Daniel Harabor, Peter J Stuckey, Jiaoyang Li, and Sven Koenig. 2019. Searching with consistent prioritization for multi-agent path finding. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 33. 7643–7650.

Hang Ma, Jiaoyang Li, TK Satish Kumar, and Sven Koenig. 2017. Lifelong Multi-Agent Path Finding for Online Pickup and Delivery Tasks. In *Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems*. 837–845.

Yining Ma, Jingwen Li, Zhiguang Cao, Wen Song, Le Zhang, Zhenghua Chen, and Jing Tang. 2021. Learning to Iteratively Solve Routing Problems with Dual-Aspect Collaborative Transformer. In *Advances in Neural Information Processing Systems*, Vol. 34. 11096–11107.

Keisuke Okumura. 2024. Engineering LaCAM*: Towards Real-Time, Large-Scale, and Near-Optimal Multi-Agent Pathfinding. arXiv:2308.04292 [cs.AI] https://arxiv.org/abs/2308.04292

Keisuke Okumura, Manao Machida, Xavier Défago, and Yasumasa Tamura. 2019. Priority Inheritance with Backtracking for Iterative Multi-agent Path Finding. *CoRR* abs/1901.11282 (2019). arXiv:1901.11282 http://arxiv.org/abs/1901.11282

Mike Phillips and Maxim Likhachev. 2011. Sipp: Safe interval path planning for dynamic environments. In *2011 IEEE International Conference on Robotics and Automation*. IEEE, 5628–5635.

Guillaume Sartoretti, Justin Kerr, Yunfei Shi, Glenn Wagner, T. K. Satish Kumar, Sven Koenig, and Howie Choset. 2018. PRIMAL: Pathfinding via Reinforcement and Imitation Multi-Agent Learning. *CoRR* abs/1809.03531 (2018). arXiv:1809.03531 http://arxiv.org/abs/1809.03531

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. *Proximal policy optimization algorithms*. Technical Report. arXiv preprint arXiv:1707.06347.

Guni Sharon, Roni Stern, Ariel Felner, and Nathan R Sturtevant. 2015. Conflict-based search for optimal multi-agent pathfinding. *Artificial Intelligence* 219 (2015), 40–66.

Huang Shengyi, Raffin Antonin, Kanervisto Anssi, and Wang Weixun. 2022. The 37 Implementation Details of Proximal Policy Optimization. In *Blog Track at ICLR 2022*. https://openreview.net/forum?id=Hl6jCqIp2j

Devon Sigurdson, Vadim Bulitko, William Yeoh, Carlos Hernández, and Sven Koenig. 2018. Multi-Agent Pathfinding with Real-Time Heuristic Search. In *2018 IEEE Conference on Computational Intelligence and Games (CIG)*. 1–8. https://doi.org/10.1109/CIG.2018.8490436

Alexey Skrynnik, Anton Andreychuk, Maria Nesterova, Konstantin Yakovlev, and Aleksandr Panov. 2024. Learn to follow: decentralized lifelong multi-agent pathfinding via planning and learning. In *Proceedings of the Thirty-Eighth AAAI Conference on Artificial Intelligence and Thirty-Sixth Conference on Innovative Applications of Artificial Intelligence and Fourteenth Symposium on Educational Advances in Artificial Intelligence (AAAI'24/IAAI'24/EAAI'24)*. AAAI Press, Article 1956, 9 pages. https://doi.org/10.1609/aaai.v38i16.29704

Roni Stern. 2019. *Multi-Agent Path Finding – An Overview*. 96–115. https://doi.org/10.1007/978-3-030-33274-7_6

Roni Stern, Nathan R Sturtevant, Ariel Felner, Sven Koenig, Hang Ma, Thayne T Walker, Jiaoyang Li, Dor Atzmon, Liron Cohen, TK Satish Kumar, et al. 2019. Multi-agent pathfinding: Definitions, variants, and benchmarks. In *Twelfth Annual Symposium on Combinatorial Search*.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in neural information processing systems* 30 (2017).

Rishi Veerapaneni, Arthur Jakobsson, Kevin Ren, Samuel Kim, Jiaoyang Li, and Maxim Likhachev. 2024a. Work Smarter Not Harder: Simple Imitation Learning with CS-PIBT Outperforms Large Scale Imitation Learning for MAPF. arXiv:2409.14491 [cs.MA] https://arxiv.org/abs/2409.14491

Rishi Veerapaneni, Qian Wang, Kevin Ren, Arthur Jakobsson, Jiaoyang Li, and Maxim Likhachev. 2024b. Improving Learnt Local MAPF Policies with Heuristic Search. In *34th International Conference on Automated Planning and Scheduling*. https://openreview.net/forum?id=6JEBeiztNT

Malte Von der Burg and Alexei Sharpanskykh. 2023. Multi-Agent Planning for Autonomous Airport Surface Movement Operations. In *13th SESAR Innovation Days 2023 (SESAR Innovation Days)*. 13th SESAR Innovation Days 2023, SIDS 2023 ; Conference date: 27-11-2023 Through 30-11-2023.

Yutong Wang, Tanishq Duhan, Jiaoyang Li, and Guillaume Adrien Sartoretti. 2025. LNS2+RL: Combining Multi-agent Reinforcement Learning with Large Neighborhood Search in Multi-agent Path Finding. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*.

Jiayi Weng, Huayu Chen, Dong Yan, Kaichao You, Alexis Duburcq, Minghao Zhang, Yi Su, Hang Su, and Jun Zhu. 2022. Tianshou: A Highly Modularized Deep Reinforcement Learning Library. *Journal of Machine Learning Research* 23, 267 (2022), 1–6. http://jmlr.org/papers/v23/21-1127.html

Wenying Wu, Subhrajit Bhattacharya, and Amanda Prorok. 2019. Multi-Robot Path Deconfliction through Prioritization by Path Prospects. *CoRR* abs/1908.02361 (2019). arXiv:1908.02361 http://arxiv.org/abs/1908.02361

Peter R. Wurman, Raffaello D'Andrea, and Mick Mountz. 2007. Coordinating Hundreds of Cooperative, Autonomous Vehicles in Warehouses. *AI Mag.* 29 (2007), 9–20. https://api.semanticscholar.org/CorpusID:10475273

Zhongxia Yan and Cathy Wu. 2024. Neural Neighborhood Search for Multi-agent Path Finding. In *The Twelfth International Conference on Learning Representations*. https://openreview.net/forum?id=2NpAw2QJBY

Zhongxia Yan, Han Zheng, and Cathy Wu. 2024. Multi-agent Path Finding for Cooperative Autonomous Driving. 12361–12367. https://doi.org/10.1109/ICRA57147.2024.10611649

Chengxuan Ying, Tianle Cai, Shengjie Luo, Shuxin Zheng, Guolin Ke, Di He, Yanming Shen, and Tie-Yan Liu. 2021. Do Transformers Really Perform Bad for Graph Representation? *CoRR* abs/2106.05234 (2021). arXiv:2106.05234 https://arxiv.org/abs/2106.05234

Jingjin Yu and Steven LaValle. 2013. Structure and intractability of optimal multi-robot path planning on graphs. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 27. 1443–1449.

Ying Yu, Xin Wang, Ray Zhong, and George Q. Huang. 2016. E-commerce Logistics in Supply Chain Management: Practice Perspective. *Procedia CIRP* 52 (12 2016), 179–185. https://doi.org/10.1016/j.procir.2016.08.002

Shuyang Zhang, Jiaoyang Li, Taoan Huang, Sven Koenig, and Bistra Dilkina. 2022. Learning a Priority Ordering for Prioritized Planning in Multi-Agent Path Finding. In *Proceedings of the Symposium on Combinatorial Search (SoCS)*. 208–216. https://doi.org/10.1609/socs.v15i1.21769

Han Zheng, Zhongxia Yan, and Cathy Wu. 2024. Multi-agent Path Finding for Mixed Autonomy Traffic Coordination. In *2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 958–965. https://doi.org/10.1109/IROS58592.2024.10802051

Margareta Živičnjak, Kristijan Rogić, and Ivona Bajor. 2022. Case-study analysis of warehouse process optimization. *Transportation Research Procedia* (2022). https://api.semanticscholar.org/CorpusID:252901518

A Mixed Zero-Shot Transfer to Different Agent Numbers, Planning Horizon, and Map Layouts.

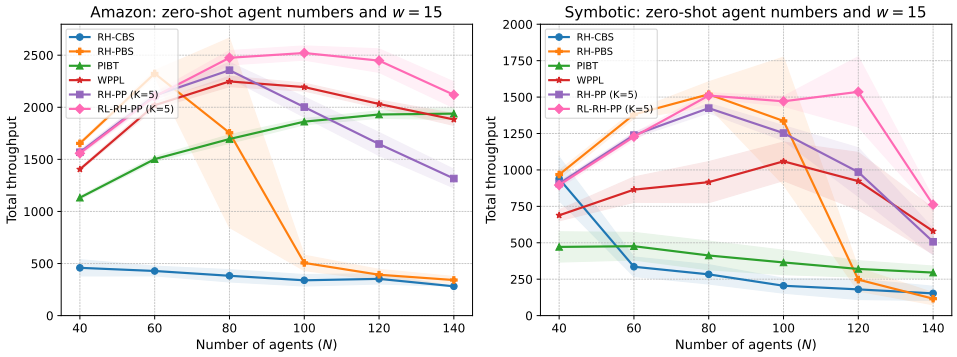

Fig. 22. Total throughput versus agent number N in zero-shot transfer evaluation. The base model is trained at $N = 120$ and $w = 20$, and evaluated zero-shot at other N with $w = 15$.

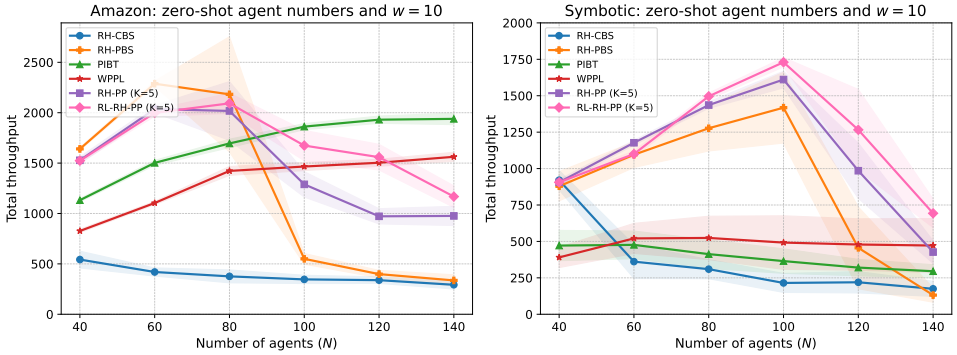

Fig. 23. Total throughput versus agent number N in zero-shot transfer evaluation. The base model is trained at $N = 120$ and $w = 20$, and evaluated zero-shot at other N with $w = 10$.

Fig. 24. Total throughput versus different Symbotic maps in zero-shot transfer evaluation. The base model is trained at $N = 120$ and $w = 20$ on the Original map, and evaluated zero-shot at other map layouts with $w = 20$.

B Supplementary Results for β and K in the Ablation Study.

Similar effects of increasing β and K is observed compared to the result shown in our main manuscript.

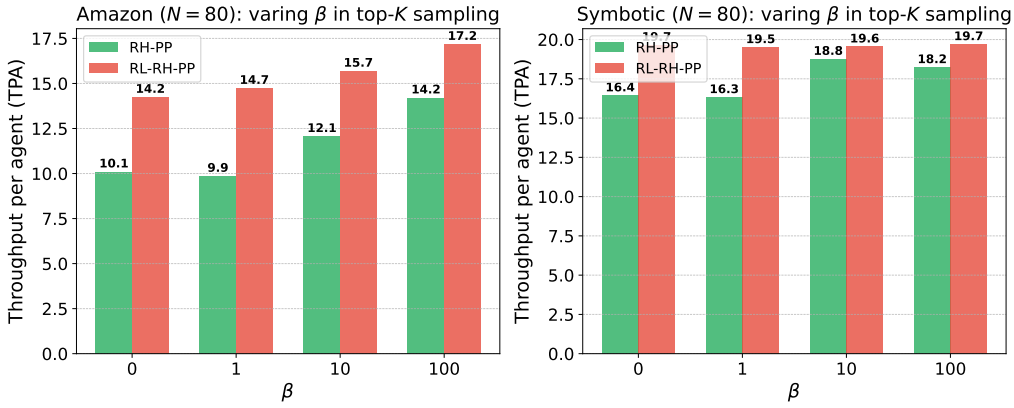

Fig. 25. Throughput per agent (TPA) at evaluation vs β, evaluated with $N = 80$

C Notation Table

Notation	Description
General MAPF and Agent Parameters	
N	Number of agents.
V, E	Vertices and edges defining the MAPF graph $G = (V, E)$.
W, H	Width and height of the warehouse map.
g_i	Goal location of agent i.
o_i	Indicator if agent i is forced to use its shortest path (when no feasible single-agent path is found).
π_i	Path sequence of locations for agent i.
$d_{i,t}$	Average distance of agent i to its next goals at time t.
TPA	Throughput per agent over the simulation horizon.
t	Discrete simulation time step index.
T	Total simulation horizon (e.g., 800 simulation time steps).
$c_{i,t}$	binary indicator equal to 1 if, at time t, agent i's plan consists of *wait* actions for the next h steps (and 0 otherwise).
$s_{i,t}$	binary indicator equal to 1 if, at time t, RH-PP fails to find a feasible path for agent i under the selected priority order (and 0 otherwise).
Rolling-Horizon Planning	
h	Execution horizon in rolling-horizon planning.

Table 8 (continued)

Notation	Description
w	Planning horizon in rolling-horizon planning.
$\prec$	A total priority order over agents.
K	Number of candidate priority orders in Top-K sampling.
$\text{cost}(\prec_k)$	Heuristic cost for the k-th sampled priority order.
β	Penalty weight in the cost function $\text{cost}(\prec_k)$.
κ	Weighting factor for congestion penalty in the reward.
σ	Weighting factor for infeasibility penalty in the reward.
Reinforcement Learning and PPO	
a_t	Action at simulation time step t, representing the selection of one or more priority orders.
$\boldsymbol{o}_t$	Observation at time t.
$R(\boldsymbol{o}_t, a_t)$	Reward at observation $\boldsymbol{o}_t$ and action a_t.
σ_i^t	shortest path of agent i at simulation time step t.
r	(maximum) length of the shortest path.
$A(\boldsymbol{o}_t, a_t)$	Advantage function in PPO.
$\hat{R}_t$	Reward-to-go from simulation time step t onward in PPO.
γ	Discount factor in reinforcement learning.
ϵ	PPO clipping parameter.
η	Entropy coefficient in PPO's objective.
f	Rollout reuse per epoch in PPO (reusing collected data for multiple updates).
B	Batch size for PPO updates.
Z	Total number of PPO training epochs (policy updates).
τ	Rollout trajectories in PPO training.
$\mathcal{D}$	Rollout dataset.
α	Adam learning rate (used interchangeably with λ).
ω	Learning rate decay factor (exponential).
g_{clip}	Gradient clipping threshold.
ϕ, θ	Parameters of the value function and policy networks in RL.
Neural Architecture	
L	Number of stacked encoder layers in the transformer.
U	Number of attention heads in each multi-head attention layer.
u	The u-th head in attention.
d	Embedding dimension used in the network (e.g., for position embeddings).
$\mathbf{H}^{(\ell)}$	Hidden representation output by the ℓ-th encoder layer.
$\mathbf{X}$	Dictionary of learnable position embeddings for each cell/vertex.
$\mathbf{K}, \mathbf{V}$	Key and value matrices used in the transformer model.
$\mathbf{L}$	Logit key matrix used for sampling in the decoder.
$\mathbf{W}^K, \mathbf{W}^V, \mathbf{W}^L$	Learnable projection matrices for keys, values, and logit keys.
$\mathbf{W}^{Q_{\text{step}}}$	Learnable projection matrix for query at decoding step.
$\mathbf{f}$	Fixed global context embedding.

Table 9. Hyperparameter Settings for RL-RH-PP

Parameter	Value
Training epochs	4000
Rollout reuse per epoch (f)	3
Entropy loss weight (η)	0.01
PPO clipping parameter (ϵ)	0.2
Initial learning rate (λ)	0.001
Learning rate decay (ω)	0.999
Batch size (B)	32
Gradient clipping threshold (g_{clip})	0.5
Discount factor (γ)	0.99
Number of encoder layers (L)	2
Number of attention heads (H)	4
Embedding dimension (d)	32

Table 8 (continued)

Notation	Description
p	Autoregressive decoding step index in the transformer decoder.
a_n	Selected agent at decoding step n.
$\mathcal{U}_{n-1}$	Set of agents already chosen in previous decoding steps.
$p(a_n = i \mid \mathbf{q}_n, \mathcal{U}_{n-1})$	Probability of selecting agent i at decoding step n.
$\mathbf{q}_n$	Query vector at decoding step n.
$\mathbf{q}_{n,u}$	Query vector at decoding step n for attention head u.
$\mathbf{g}_n$	Glimpse vector at decoding step n.
$\boldsymbol{\rho}_n$	Unnormalized logits at decoding step n.

D Experiment Details

For the selection of hyperparameters in RL training, please refer to Table 9. Each parameter is chosen based on 5 empirical trials (considering memory usage, rollout efficiency, and training stability) rather than an extensive hyperparameter search or optimization. Additionally, we report an average environment sampling rate of 0.064 CPU time per parallel simulation step, achieved using an AMD Ryzen Threadripper PRO 5995WX CPU with 128 cores, running multi-threaded parallel simulations across 16 environments.

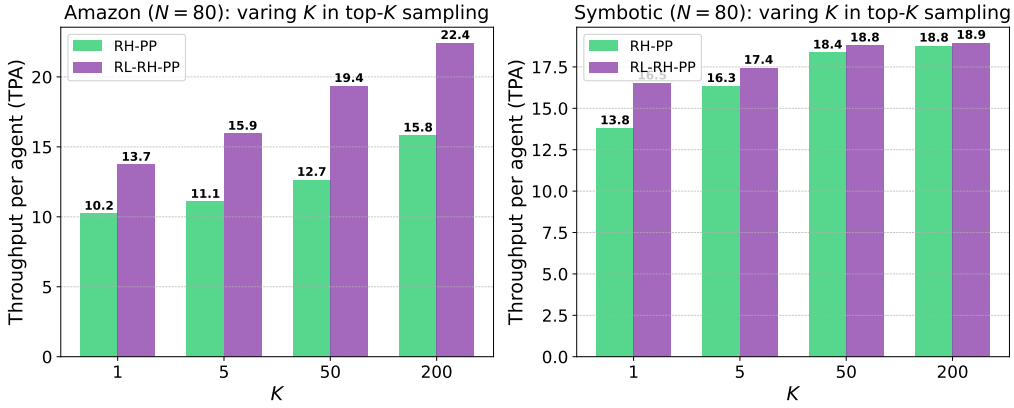

Fig. 26. Throughput per agent (TPA) at evaluation vs K, evaluated with $N = 80$

Received 30 September 2025; accepted 04 February 2026