# LEARNING REPAIRABLE INITIALIZATIONS FOR MULTI-AGENT PATH FINDING VIA DISCRETE DIFFUSION

**Anonymous authors**
Paper under double-blind review

## ABSTRACT

Multi-Agent Path Finding (MAPF) is a coordination problem that requires computing globally consistent, collision-free trajectories for multiple agents from their start positions to assigned goals in a shared environment. In dense settings, sequential prioritized planning can concentrate repeated conflicts on a small subset of agents, hindering local repair. Despite its importance for downstream repair, the initialization stage of repair-based MAPF solvers has received relatively limited research attention. We propose DiffLNS, a hybrid framework that integrates a discrete denoising diffusion probabilistic model (D3PM) with LNS2 to improve initialization for repair. Trained on expert demonstrations, the D3PM initializer uses diffusion-aware sparse social attention to learn a spatiotemporal prior and samples diverse plans directly in the categorical action space. These plans serve as warm starts for LNS2, which completes unfinished trajectories and resolves conflicts under hard MAPF constraints. Controlled comparisons show that diffusion-generated initializations exhibit more balanced conflict loads across agents and achieve higher repair success under the same repair procedure and budget, despite starting with more colliding pairs. Although trained with at most 96 agents, the initializer generalizes to tested scenarios with up to 312 agents. Across 20 complex and congested settings, DiffLNS achieves an average success rate of 95.8%, exceeding the strongest tested baseline by 9.65 percentage points and matching or exceeding all baselines in every setting. To the best of our knowledge, this is the first work to use discrete diffusion to warm-start an LNS-based MAPF solver.

## 1 INTRODUCTION

Multi-Agent Path Finding (MAPF) aims to compute collision-free trajectories that guide multiple agents from their start locations to individual goals in a shared environment. As a fundamental multi-agent coordination problem, MAPF is computationally challenging and underpins a broad spectrum of real-world applications, including warehouse automation, autonomous vehicle coordination, unmanned aircraft systems, fleet management, and multi-robot systems (Stern et al., 2019). Consequently, scalable suboptimal planning has remained a sustained research focus.

Among existing approaches, repair-based methods such as Large Neighborhood Search 2 (LNS2) have demonstrated strong empirical performance by iteratively repairing the paths of selected agent subsets to resolve conflicts (Li et al., 2021a; 2022). Recent work on LNS-based MAPF has made significant progress in the repair stage, including adaptive neighborhood selection, destroy strategies, and enhanced replanning (Huang et al., 2022; Phan et al., 2024; Yan & Wu, 2024; Wang et al., 2025). However, the initialization stage remains comparatively underexplored, with limited work on selecting promising initial solutions (Huber et al., 2025). This gap is consequential because the initial plan can substantially affect the success rate and convergence speed of LNS-based repair (Huber et al., 2025; Li et al., 2021a). In dense and congested environments, the LNS2 prioritized planning (PP) initializer with SIPPS can leave a highly concentrated pattern of residual conflicts. Since PP commits paths sequentially, decisions made for earlier agents restrict the spatiotemporal options available to later ones, causing unresolved interactions to accumulate among a small subset of agents or agent pairs. During repair, LNS2 applies the same sequential planning mechanism to replan only a bounded neighborhood while keeping the remaining paths fixed. Consequently, these tightly coupled conflicts can be difficult to reorganize and may persist across iterations.



Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

A useful initializer for LNS-based MAPF should provide globally coherent joint plans that capture spatiotemporal coordination among agents, motivating a structured generative prior over coordinated trajectories. Discrete diffusion is well suited to this formulation because it can capture complex trajectory distributions through iterative denoising, operate directly in the categorical action space, and sample diverse candidate drafts for subsequent repair (Carvalho et al., 2023; Shaoul et al., 2025; Liang et al., 2025a). However, directly applying diffusion to MAPF remains challenging: the model must generate centralized joint plans for many agents, maintain temporal consistency toward individual goals, and efficiently model conflict-relevant agent–agent interactions during denoising.

To address these challenges, we introduce *DiffLNS*, a hybrid framework combining a discrete denoising diffusion probabilistic model (D3PM) (Austin et al., 2021) with LNS2. The D3PM initializer learns a spatiotemporal prior over coordinated multi-agent action trajectories from expert demonstrations. Diffusion-aware sparse social attention constructs dynamic neighborhoods from the current denoising state to focus computation on conflict-relevant agents. During inference, DiffLNS samples multiple drafts for the same instance, repairs them independently with LNS2, and selects the best feasible solution. This design provides diverse repair seeds and allows candidate generation and repair to be parallelized. Our controlled comparisons further reveal a counterintuitive finding: although the PP initializer seeks to limit collisions, diffusion-generated initializations achieve higher repair success than those from the PP initializer under the same LNS2 procedure and budget, despite starting with more colliding pairs. Their more balanced conflict loads suggest that conflict structure also matters for initialization repairability.

We summarize our main contributions and findings below:

- **A structural characterization of initialization repairability.** Through paired comparisons under an identical LNS2 repair procedure, we show that repairability is not determined by the raw number of initial conflicts alone. D3PM-generated initializations exhibit lower conflict concentration across colliding pairs and more balanced conflict loads across agents, which are associated with improved conflict-removal dynamics and repair success.
- **A hybrid framework for initialization-aware MAPF.** We propose DiffLNS, which integrates a discrete diffusion-based generator with LNS2 repair, shifting the focus from repair-only improvement to initialization quality. To the best of our knowledge, this is the first work that leverages discrete diffusion to warm-start an LNS-based MAPF solver.
- **Diffusion-aware sparse social attention.** We introduce a dynamic, step-dependent sparse attention mechanism that constructs local neighborhoods from the current noisy trajectory estimate. This design reduces unnecessary interaction modeling while focusing on conflict-relevant agent pairs, improving both efficiency and plan repairability.

## 2 PRELIMINARIES

### 2.1 MULTI-AGENT PATH FINDING

Multi-Agent Path Finding (MAPF) is commonly defined as the problem of finding collision-free paths for multiple agents from individual start locations to individual goals in a shared graph or grid environment (Stern et al., 2019). Let $G \in \{0,1\}^{H\times W}$ denote the map, where $G_{h,w} = 0$ indicates a traversable cell and $G_{h,w} = 1$ indicates an obstacle. A set of $N$ agents is specified by start locations $\{s_i\}_{i=1}^{N}$ and goal locations $\{g_i\}_{i=1}^{N}$. The planning horizon is denoted by $T$, and $\tau \in \{0,1,\dots,T\}$ indexes discrete timesteps. At each step $\tau \in \{1,\dots,T\}$, each agent $i$ executes an action

$$a_i^\tau \in \mathcal{A}, \qquad \mathcal{A} = \{\texttt{stay}, \texttt{up}, \texttt{down}, \texttt{left}, \texttt{right}\}.$$

A valid solution must avoid vertex conflicts, where different agents occupy the same cell at the same timestep, and edge conflicts, where two agents swap positions between consecutive timesteps. Solution quality is measured by sum of costs (SOC), defined as the sum of the individual path costs over all agents.

### 2.2 DISCRETE DENOISING DIFFUSION PROBABILISTIC MODEL

A discrete denoising diffusion probabilistic model defines a forward Markov chain that gradually corrupts a clean discrete variable $\mathbf{x}_0$ over $K$ steps (Hoogeboom et al., 2021; Austin et al., 2021). For

a one-hot row vector $\mathbf{x}_k \in \{0,1\}^C$, the one-step transition is parameterized by $\mathbf{Q}_k \in \mathbb{R}^{C\times C}$:

$$q(\mathbf{x}_k \mid \mathbf{x}_{k-1}) = \mathrm{Cat}(\mathbf{x}_k;\, \mathbf{p} = \mathbf{x}_{k-1}\mathbf{Q}_k)\,. \tag{1}$$

For structured discrete data, this transition is applied independently to each discrete element.

D3PM commonly adopts an $\mathbf{x}_0$-parameterization, in which the model predicts the clean state $\mathbf{x}_0$ from $\mathbf{x}_k$ through $\tilde{p}_\theta(\mathbf{x}_0 \mid \mathbf{x}_k)$ (Hoogeboom et al., 2021; Austin et al., 2021). The reverse transition is then constructed as

$$p_\theta(\mathbf{x}_{k-1} \mid \mathbf{x}_k) \propto \sum_{\tilde{\mathbf{x}}_0} q(\mathbf{x}_{k-1}, \mathbf{x}_k \mid \tilde{\mathbf{x}}_0)\, \tilde{p}_\theta(\tilde{\mathbf{x}}_0 \mid \mathbf{x}_k). \tag{2}$$

The standard training objective combines the variational bound with an auxiliary clean-state prediction term (Austin et al., 2021):

$$\mathcal{L}_\lambda = \mathcal{L}_{\mathrm{vb}} + \lambda\, \mathbb{E}_{q(\mathbf{x}_0)} \mathbb{E}_{q(\mathbf{x}_k \mid \mathbf{x}_0)} \left[-\log \tilde{p}_\theta(\mathbf{x}_0 \mid \mathbf{x}_k)\right]. \tag{3}$$

Appendix D.3 gives the variational decomposition and specifies the objective used in our implementation. At inference time, sampling starts from $\mathbf{x}_K \sim p(\mathbf{x}_K)$ and iteratively applies the learned reverse transitions for $k = K, \ldots, 1$.

### 2.3 Large Neighborhood Search for MAPF

MAPF-LNS2 (Li et al., 2022) is an unbounded suboptimal MAPF algorithm based on large neighborhood search. It first constructs an initial plan using prioritized planning (PP) (Erdmann & Lozano-Perez, 1986) with SIPPS as the low-level solver; the resulting plan may contain collisions. LNS2 then repeatedly selects a subset of agents for replanning while keeping the remaining paths fixed, and accepts an update if the number of colliding pairs does not increase. The same PP+SIPPS procedure is used for neighborhood replanning, where SIPPS computes single-agent paths under dynamic obstacles and soft collision constraints (Li et al., 2022). Because each iteration repairs only a local subset of agents, the difficulty of repair depends strongly on the conflict structure of the initial plan: localized and mild conflicts are easier to resolve than heavily entangled collisions. This property makes initialization quality a key factor in the effectiveness of LNS2.

## 3 Methodology

DiffLNS formulates MAPF initialization as structured generative warm-starting rather than direct feasibility solving. The denoising model learns a spatiotemporal prior over coordinated joint action sequences and generates spatiotemporally structured drafts, while LNS2 enforces hard MAPF constraints by repairing residual conflicts and incomplete paths.

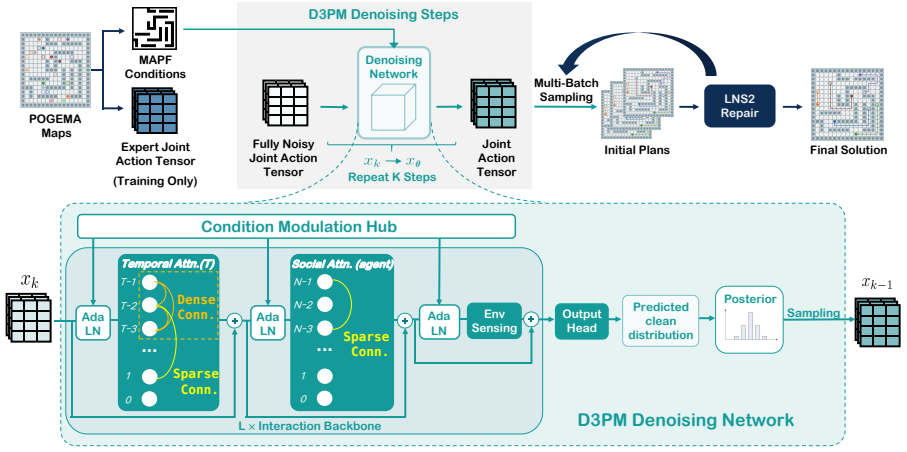



Figure 1: Overview of the DiffLNS hybrid framework. The lower panel shows the denoising network architecture used in the D3PM initializer, detailed in Section 3.2.

### 3.1 Hybrid Framework Overview

**Joint action formulation.** An initial plan is represented as a joint discrete action tensor

$$\mathbf{x}_0 \in \{0,1\}^{N\times T\times C}, \tag{4}$$

where $N$ is the number of agents, $T$ is the planning horizon, and $C = 5$ corresponds to the grid actions {`stay`, `up`, `down`, `left`, `right`}. Each slice $\mathbf{x}_{0,i,\tau,:}$ denotes a one-hot action token for agent $i$ at timestep $\tau$. This representation matches the categorical structure of grid-based MAPF and enables D3PM to operate directly in the discrete joint action space.

**Overall pipeline.** Given a MAPF instance, the D3PM initializer samples a batch of joint action drafts conditioned on the instance. Each draft is converted into a seed plan and repaired by LNS2. After completing repair of the current batch, DiffLNS returns the feasible candidate with the lowest sum of costs (SOC), if any. Otherwise, the system samples and repairs additional batches until the evaluation budget is exhausted. This design makes the framework complementary to repair-side improvements in LNS-based MAPF: rather than modifying the repair operator itself, DiffLNS improves the initial plans provided to it. The DiffLNS hybrid framework is illustrated in Fig. 1.

### 3.2 Structured Initialization with Discrete Diffusion

**Conditional denoising model.** Training samples consist of MAPF conditions $\mathbf{c}$ and expert joint action tensors $\mathbf{x}_0$ defined in Eq. (4). Given a noisy sample $\mathbf{x}_k \sim q(\mathbf{x}_k \mid \mathbf{x}_0)$, we train a conditional clean-state predictor $\tilde{p}_\theta(\mathbf{x}_0 \mid \mathbf{x}_k, \mathbf{c})$ under the $\mathbf{x}_0$-parameterization introduced in Sec. 2.2. The denoising network embeds the noisy one-hot action token of each agent at each timestep and fuses it with start, goal, map, and diffusion-step features. The resulting conditioned action tokens are processed by structured spatiotemporal blocks that integrate temporal attention, sparse social attention, and environment sensing. Temporal attention promotes motion consistency and goal progress through dense local-window attention and sparse strided global connections along each agent trajectory; sparse social attention models local conflict interactions, and environment sensing provides obstacle awareness. Details of dataset construction, preprocessing, and the full network architecture are provided in Appendix D; the sparse social attention module is described below.

**Sparse social attention.** A central bottleneck in centralized denoising lies in the social interaction module. Dense all-to-all attention is not only computationally expensive but also poorly aligned with MAPF, because most conflict-relevant interactions are local, whereas attending to all agents disperses attention across many irrelevant pairs. We therefore replace dense social attention with a diffusion-aware dynamic sparse variant. At each denoising step, agents attend to dynamic local neighborhoods constructed from inferred-trajectory proximity, thereby focusing computation on nearby agents that are most relevant to potential conflicts.

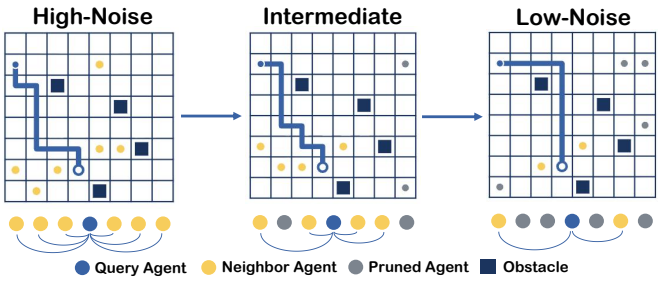



Figure 2: Diffusion-aware sparse social attention.

At diffusion step $k$, we construct the social neighborhood from the current noisy tensor $\mathbf{x}_k$ rather than from a fixed interaction graph. We first derive a reference trajectory from the sampled actions of each agent, with $\mathbf{p}_{i,\tau}^{\inf}$ denoting its inferred position at timestep $\tau$, and then compute pairwise trajectory proximity as

$$d_{ij} = \min_{\tau=1,\dots,T} \left\| \mathbf{p}_{i,\tau}^{\inf} - \mathbf{p}_{j,\tau}^{\inf} \right\|_1, \tag{5}$$

which measures the minimum distance between the two agents' inferred positions over the planning horizon. For each agent, we retain itself and the $M_k$ nearest non-self neighbors, with $M_k$ a clipped fraction of $N-1$. As illustrated in Fig. 2, the neighborhood is recomputed at every diffusion step: subject to the clipping bounds, the schedule favors broader neighborhoods early in denoising and sparser ones later as the inferred trajectory becomes more precise. Let $\mathcal{N}_k(i)$ denote the sparse neighborhood of agent $i$ at diffusion step $k$. For each timestep $\tau$, the social module first projects the agent features into query, key, and value vectors, denoted by $\mathbf{q}_{i,\tau}$, $\mathbf{k}_{j,\tau}$, and $\mathbf{v}_{j,\tau}$. Attention for agent $i$ is then computed only over $j \in \mathcal{N}_k(i)$:

$$\alpha_{ij}^{\tau} = \mathrm{softmax}_{j\in\mathcal{N}_k(i)}\left(\frac{\mathbf{q}_{i,\tau}^{\top}\mathbf{k}_{j,\tau}}{\sqrt{d}} + b_\theta\big(\mathbf{p}_{i,\tau}^{\mathrm{inf}} - \mathbf{p}_{j,\tau}^{\mathrm{inf}}\big)\right), \tag{6}$$

where $b_\theta(\cdot)$ is a learned geometric bias generated from relative inferred positions. The updated social feature is obtained by aggregating value vectors from the selected neighbors according to the sparse attention weights. This computation preserves focused attention on the most relevant local interactions while reducing the cost of dense $N^2$ interaction modeling.

**Training objective.** Our main training signal is the generative objective

$$\mathcal{L}_{\mathrm{gen}} = \mathcal{L}_{\mathrm{aux}} + \lambda_{\mathrm{KL}}\mathcal{L}_{\mathrm{KL}}, \tag{7}$$

where $\mathcal{L}_{\mathrm{aux}}$ is the auxiliary clean-state prediction term in Eq. (3). The KL term corresponds to the posterior-matching component of the variational bound $\mathcal{L}_{\mathrm{vb}}$. Let $q(\mathbf{x}_0, \mathbf{c})$ denote the expert data distribution over clean joint action tensors and MAPF conditions. The KL term is

$$\mathcal{L}_{\mathrm{KL}} = \mathbb{E}_{q(\mathbf{x}_0,\mathbf{c})}\mathbb{E}_{k\sim\mathrm{Unif}(\{1,\dots,K\})}\mathbb{E}_{q(\mathbf{x}_k|\mathbf{x}_0)}\left[\mathrm{KL}(q(\mathbf{x}_{k-1} \mid \mathbf{x}_k, \mathbf{x}_0) \,\|\, p_\theta(\mathbf{x}_{k-1} \mid \mathbf{x}_k, \mathbf{c}))\right]. \tag{8}$$

Under the $\mathbf{x}_0$-parameterization, $p_\theta(\mathbf{x}_{k-1} \mid \mathbf{x}_k, \mathbf{c})$ is constructed from Eq. (2) using $\tilde{p}_\theta(\mathbf{x}_0 \mid \mathbf{x}_k, \mathbf{c})$. In practice, $\mathcal{L}_{\mathrm{aux}}$ is a token-level cross-entropy loss on clean action prediction, while $\mathcal{L}_{\mathrm{KL}}$ aligns the induced reverse transition with the analytic posterior.

We further add a task-oriented auxiliary loss $\mathcal{L}_{\mathrm{task}}$ for goal progress, conflict reduction, and action validity, with exact forms provided in Appendix D.3. The final objective is

$$\mathcal{L} = \mathcal{L}_{\mathrm{gen}} + \mathcal{L}_{\mathrm{task}}, \tag{9}$$

where $\mathcal{L}_{\mathrm{gen}}$ remains the primary expert-imitation objective. For stable optimization, we first train with $\mathcal{L}_{\mathrm{gen}}$ alone and then gradually increase the weight of $\mathcal{L}_{\mathrm{task}}$.

**Agent scalability.** DiffLNS is not architecturally tied to a fixed team size. Although the diffusion initializer is trained on smaller teams, it can be applied to larger teams at inference time because forward corruption and reverse denoising operate tokenwise on the joint action tensor. Increasing the number of agents changes only the number of action tokens, not the form of the diffusion process.

The denoiser shares parameters across agents and is permutation-equivariant along the agent dimension:

$$f_\theta(P\mathbf{x}_k, Pc) = Pf_\theta(\mathbf{x}_k, c), \tag{10}$$

where $P$ is a permutation matrix acting on the agent axis. During denoising, each action token is predicted from the corrupted joint plan and the MAPF condition, using both global scene context and local interaction cues from other agents. Sparse social attention implements this locality bias through dynamic neighborhoods $\mathcal{N}_k(i)$ constructed from the current trajectory estimate. These properties make inference on larger teams well defined and provide an inductive bias for reusing local coordination patterns, but they do not guarantee distributional generalization. We therefore evaluate scaling beyond the training agent range empirically and show that DiffLNS remains effective on substantially larger teams in dense MAPF settings.

### 3.3 Iterative Multi-Sample Repair with LNS2

The diffusion initializer produces structured plans that capture useful global coordination patterns, but these plans may still contain unfinished paths, residual conflicts, or invalid moves. We therefore use LNS2 as a repair stage to convert these drafts into valid MAPF solutions.

**Preprocessing diffusion drafts.** Empirically, we observe that in many diffusion drafts, some agents already reach their goals, while many others terminate close to their goals. Before passing each sampled draft to LNS2, we therefore apply a short preprocessing step: redundant suffixes are removed after an agent reaches its goal, and unfinished trajectories are completed with an obstacle-aware shortest-path suffix from the last valid position to the goal. Here, invalid actions refer to action attempts that move an agent out of the map or into an obstacle. This preprocessing step preserves useful global structure while completing all trajectories to their goals before repair. RepairBatch applies LNS2 to all candidates in the current batch under the per-candidate repair and remaining evaluation budgets, then returns their repair outcomes together.

**Algorithm 1** Iterative Repair with D3PM and LNS2

**Input:** MAPF instance $\mathcal{I}$, initializer $p_\theta$, batch size $M$, rounds $R$, time budget $B$

**for** $r = 1, \dots, R$ **do**
  **if** runtime $\geq B$ **then return** failure
  sample $\{\hat{\mathbf{x}}_0^{(m)}\}_{m=1}^{M} \sim p_\theta(\cdot \mid \mathcal{I})$
  $\Pi^{(m)} \leftarrow \textsc{Preprocess}(\hat{\mathbf{x}}_0^{(m)}), m = 1, \dots, M$
  $\{\mathbf{x}^{(m)}\}_{m=1}^{M} \leftarrow$
    $\textsc{RepairBatch}(\{\Pi^{(m)}\}_{m=1}^{M}, \mathcal{I}, B)$
  $\mathcal{S}_r \leftarrow \{\mathbf{x}^{(m)} : \text{feasible}(\mathbf{x}^{(m)})\}$
  **if** $\mathcal{S}_r \neq \emptyset$ **then return** $\arg\min_{\mathbf{x} \in \mathcal{S}_r} \text{SOC}(\mathbf{x})$
**end for**
**return** failure

**LNS2 repair.** Given the preprocessed draft, LNS2 performs neighborhood-based replanning to restore feasibility. Starting from the diffusion-generated trajectories, it iteratively selects subsets of agents for replanning while keeping the remaining trajectories fixed. This local repair process progressively removes residual vertex conflicts, edge-swap conflicts, and other inconsistencies until a feasible global solution is obtained or the repair budget is exhausted.

**Multi-sample selection and iterative retry.** For each MAPF instance, the diffusion initializer samples multiple drafts under the same condition, allowing different global coordination patterns to be explored. Each round completes repair of its entire candidate batch before deciding whether to terminate. If the batch yields feasible repaired candidates, DiffLNS returns the one with the smallest SOC. Otherwise, it samples a new batch and repeats the procedure until either the maximum number of repair rounds is reached or the time budget is exhausted. This iterative strategy improves robustness by exploring diverse structured initializations within the given evaluation budget.

## 4 Experiments

**Baselines and metrics.** We compare DiffLNS with LNS2 (Li et al., 2022), LNS2+RL (Wang et al., 2025), HMAGAT (Jain et al., 2026), and LaCAM3 (Okumura, 2024). DiffLNS, LNS2, LNS2+RL, and HMAGAT share fixed per-setting time limits; LaCAM3 serves only as a reference for DiffLNS, with each instance's budget matched to DiffLNS's actual runtime. We report success rate (SR), pairwise common-success SOC ratios, and mean elapsed runtime over all instances, including failures. Each SOC ratio divides a baseline's mean SOC by DiffLNS's mean SOC on instances solved by both; values above 1 favor DiffLNS. DiffLNS runtime includes diffusion generation and LNS2 repair. Baseline implementations, batch selection, stopping rules, budgets, hardware, and metric details are provided in Appendix C.1.

**Evaluation settings.** All evaluation instances are randomly generated with POGEMA (Skrynnik et al., 2025). We consider five environment families: *Small Random* ($10 \times 10$), *Medium Maze* ($25 \times 25$), *Medium Room* ($23 \times 23$), *Medium Warehouse* ($25 \times 25$), and *Large Maze* ($33 \times 33$). These settings emphasize dense and congested MAPF regimes, where global coordination and repair quality become major bottlenecks. Detailed per-setting statistics and more result analyses are provided in Appendix C.1.

### 4.1 Results

As shown in Fig. 3, DiffLNS matches or exceeds all baselines in SR across 20 evaluated settings, with a 95.8% mean SR across settings and a 9.65-point gain over LNS2+RL, the strongest baseline by average SR. The advantage of DiffLNS is especially clear in maze and warehouse environments, where the success rates of classical baselines decrease more sharply as congestion increases. Under the matched-time protocol, DiffLNS matches or exceeds LaCAM3 in SR across all 20 settings, with larger gains at higher agent counts.

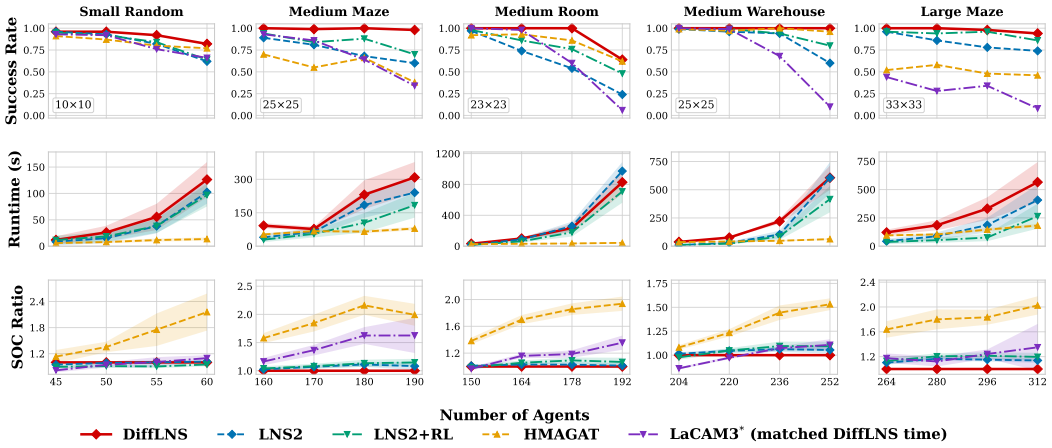



Figure 3: Performance comparison on MAPF benchmarks across different environments. The bottom row reports the ratio of each baseline's mean SOC to DiffLNS's mean SOC on their pairwise common successful instances, with DiffLNS fixed at 1. Ratios above 1 favor DiffLNS. Shaded regions indicate 95% confidence intervals for runtime and SOC ratios.

On pairwise common successful instances, the mean SOC ratios favor DiffLNS against LNS2, LNS2+RL, HMAGAT, and LaCAM3 in 18, 14, 20, and 15 of the 20 settings, respectively. Its SOC advantage is most consistent in complex maze environments and the denser room and warehouse settings, while baseline advantages are concentrated in the comparatively simpler Small Random environments and room and warehouse settings with fewer agents. These comparisons are made separately against DiffLNS because the common successful subset can differ by baseline. HMAGAT is fast and stable in some room and warehouse cases, but its paired mean SOC exceeds that of DiffLNS in all 20 settings. HMAGAT's runtime reflects both its wall-clock and 512-step episode limits: unfinished rollouts stop at whichever limit is reached first. Overall, these results show that DiffLNS combines robust feasibility with favorable paired solution quality in the tested complex and congested settings.

DiffLNS incurs higher average runtime in most evaluated settings because each candidate requires reverse-diffusion denoising followed by LNS2 repair. Runtime tends to increase with instance difficulty, as harder cases require more generated candidates and repair attempts before a feasible solution is found; detailed candidate statistics are provided in Appendix C.3. With four independent candidates per batch, ideal parallel execution can reduce candidate-processing time by up to 75% ($4\times$ speedup over serial execution), assuming balanced runtimes, sufficient resources, and negligible overhead; this is a theoretical bound. DiffLNS therefore trades additional but parallelizable computation for higher feasibility in congested settings while maintaining competitive solution quality on pairwise common successful instances.

### 4.2 Structural Analysis of Initialization Repairability

We compare D3PM and the PP initializer on 150 paired instances: 50 each from Medium Maze ($N = 190$), Medium Room ($N = 192$), and Medium Warehouse ($N = 236$). Each method generates one initialization without best-of-$k$ selection, followed by the same LNS2 repair operator with a 120 s budget. This comparison isolates initialization repairability from the benefits of multi-sample generation and selection. A conflict event is a vertex or edge conflict between an agent pair at one time step; each colliding pair is counted once regardless of event count. We measure event concentration across pairs, per-agent conflict-load Gini, and active colliding pairs during repair.

As shown in Fig. 4, D3PM begins with more colliding pairs on average (2613.3 vs. 584.8), but its conflicts are less concentrated: the top 5% of pairs account for 25.4% of events versus 40.4% for the PP initializer, and its per-agent Gini is lower (0.208 vs. 0.618). D3PM has lower top-5% concentration in 149 of 150 paired instances and lower Gini in all 150. Under identical repair, this more balanced structure is associated with a lower normalized area under the active-pair curve (AUC;

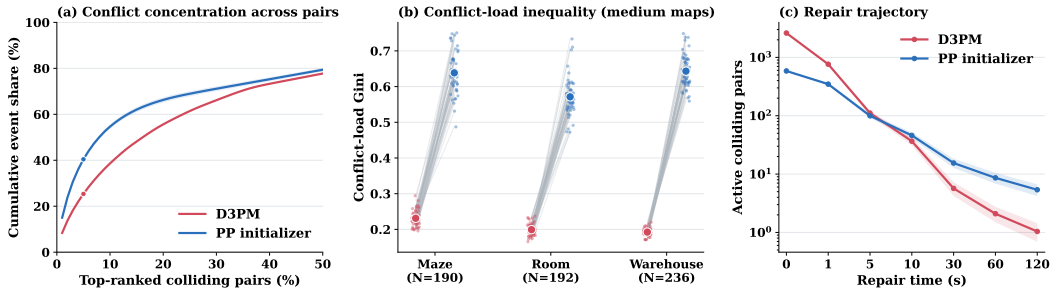



Figure 4: Initial conflict structure and dynamics under identical bounded LNS2 repair. Red: D3PM; blue: PP initializer. (a) Cumulative conflict-event share versus percentage of top-ranked colliding pairs. (b) Per-agent conflict-load Gini across three map families; light lines connect paired initializations, and large markers denote means. (c) Mean active colliding pairs over repair time (logarithmic vertical axis). Curves and shading in (a) and (c) show means and 95% stratified bootstrap confidence intervals over 150 paired instances.

0.0108 vs. 0.0360), fewer active pairs after 120 s (1.0 vs. 5.4), and higher repair success (76% vs. 44%).

These results show that raw conflict count alone does not characterize repairability. Sequential PP assigns paths in a fixed priority order: later agents must accommodate earlier paths, potentially concentrating repeated conflicts among a small set of agents or pairs. Since each LNS2 iteration replans only a bounded neighborhood while keeping other paths fixed, such tightly coupled conflicts can persist or be recreated across iterations. In contrast, D3PM distributes conflict events and per-agent loads more evenly. This structure is more compatible with bounded-neighborhood repair, consistent with its lower repair-curve AUC and higher success rate under the same operator.

### 4.3 Ablation Study on Sparse Social Attention

To evaluate the proposed sparse social attention, we compare it with a dense all-to-all variant while keeping the rest of the DiffLNS framework unchanged. We focus on the densest settings of three representative benchmark environments: Medium Maze ($N = 190$), Medium Room ($N = 192$), and Medium Warehouse ($N = 252$), where social interaction modeling is most critical.

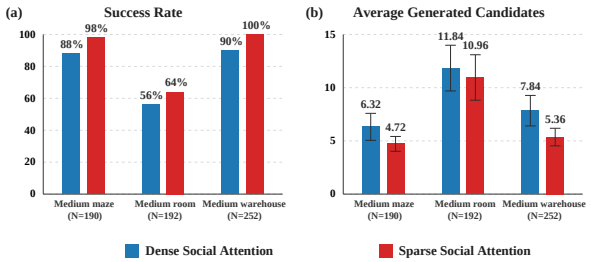



Figure 5: Ablation of sparse versus dense social attention on three dense maps. (a) Repair success rate. (b) Average number of generated candidates. Error bars indicate 95% confidence intervals.

As shown in Fig. 5, sparse social attention consistently achieves higher repair success rates while requiring fewer generated candidates across all three settings. Averaged over the three tested settings, the dense variant also incurs approximately 12% higher end-to-end runtime than the sparse variant. In congested MAPF instances, conflict-relevant interactions are primarily local, whereas dense attention distributes modeling capacity across many irrelevant agent pairs. By dynamically focusing on nearby agents that are most relevant to the current trajectory estimate, the sparse variant provides more effective social context and produces initial plans that are easier for LNS2 to repair in highly congested scenarios.

### 4.4 LIMITATIONS

DiffLNS is most useful on dense, difficult MAPF instances where initialization quality strongly affects repair; on simpler instances, pure LNS2 is often sufficient, making the extra initialization cost less necessary. Since the current denoiser is trained on $23 \times 23$ maps, generalization to substantially larger maps may be affected by distribution shift and require broader training scales. Moreover, DiffLNS remains constrained by the downstream LNS2 repair stage: results on the hardest *Medium Room* setting and the fixed-repair analysis in Section B.2 show that D3PM warm-starting improves repairability but cannot fully overcome extremely difficult repair cases under a limited budget.

## 5 RELATED WORK

**Classical MAPF solvers.** Classical MAPF algorithms formulate planning as combinatorial search. Representative methods include conflict-based search (CBS), bounded-suboptimal variants such as EECBS, and joint-configuration search methods such as LaCAM (Sharon et al., 2012; Li et al., 2021b; Okumura, 2023). Large neighborhood search (LNS) offers a scalable repair-based paradigm that starts from an initial plan and improves it by replanning agent subsets while keeping the remaining paths fixed (Li et al., 2021a; 2022). Although initialization quality becomes increasingly important on harder instances (Li et al., 2021a), most LNS-based MAPF studies focus on improving the repair process, including neighborhood selection, neural neighborhood generation, adaptive destroy strategies, and reinforcement-learning-based replanners (Huang et al., 2022; Yan & Wu, 2024; Phan et al., 2024; Wang et al., 2025). Direct studies of MAPF initialization remain limited and mainly select from existing solutions rather than generating higher-quality initializations (Huber et al., 2025). Our work instead improves the quality and diversity of initial plans provided to LNS2, targeting a complementary stage of the LNS pipeline.

**Learning-based MAPF.** Learning-based MAPF methods amortize coordination through neural policies or predictors. Early methods such as PRIMAL, DHC, and SCRIMP learn decentralized local coordination through imitation learning or multi-agent reinforcement learning (Sartoretti et al., 2019; Ma et al., 2021; Wang et al., 2023). Recent methods improve interaction modeling with graph, hypergraph, or token-based architectures, including MAGAT, HMAGAT, MAPF-GPT, and centralized predictors such as RAILGUN (Li et al., 2021c; Jain et al., 2026; Andreychuk et al., 2025; Tang et al., 2025). Although efficient at inference time, these methods typically produce limited rollouts and struggle to strictly satisfy hard MAPF constraints in dense environments. In contrast, DiffLNS generates multiple globally structured joint plans as initialization candidates and relies on LNS2 repair to ensure that returned solutions satisfy the hard MAPF constraints.

**Diffusion and generative planning.** Diffusion models have been widely adopted as trajectory priors for robot planning, motion generation, and multi-robot coordination, often in continuous state spaces with additional collision handling or planner guidance (Carvalho et al., 2023; Shaoul et al., 2025; Liang et al., 2025a;b). Their stochastic sampling naturally yields diverse candidates for a given problem instance. However, most diffusion-based planners operate over continuous trajectories, whereas grid-based MAPF is discrete and requires strict vertex and edge conflict constraints. We therefore formulate diffusion over discrete joint action tensors and use its samples to warm-start LNS2, combining multimodal generative priors with classical hard-constraint repair.

## 6 CONCLUSION

We presented DiffLNS, a hybrid MAPF framework that uses D3PM as a learned initializer for LNS2. It generates coordinated joint action drafts without requiring initial feasibility, while LNS2 enforces hard MAPF constraints. Our analysis shows that repairability depends on conflict structure as well as count: D3PM initializations exhibit lower conflict concentration and more balanced agent loads, making them more amenable to bounded-neighborhood repair. Together with diffusion-aware sparse social attention and multi-sample repair, this design improves repair success in dense and congested settings while maintaining competitive solution quality. These findings highlight initialization quality as a complementary direction to repair improvements and suggest that discrete diffusion can provide joint priors for multi-agent planning.

## AI USE STATEMENT

We used LLM-based tools to assist with manuscript drafting and polishing, including the discussion and interpretation of experimental results, and to support literature retrieval and discovery of related work. These tools also assisted with figure, table, and LaTeX formatting. The authors retain responsibility for verifying retrieved references and AI-assisted interpretations against the cited sources and experimental evidence, and take full responsibility for the final manuscript and all AI-assisted content.

## ETHICS STATEMENT

This study uses procedurally generated MAPF environments and planner-generated expert trajectories, without human participants or personal data. Within the scope of this simulation-based research, we identify no specific ethical risks.

## REPRODUCIBILITY STATEMENT

We document the evaluation settings and metrics in Section 4, with baseline implementations, hardware, time budgets, stopping rules, and metric definitions in Appendix C.1. Appendix D describes expert-data construction, the denoising architecture, training losses, the training schedule, and optimization hyperparameters. Section 4.2 specifies the paired initialization comparison and fixed repair budget, while Appendix B provides additional controlled comparisons. Detailed results and candidate-generation statistics are reported in Appendix C.

## REFERENCES

Anton Andreychuk, Konstantin Yakovlev, Aleksandr Panov, and Alexey Skrynnik. MAPF-GPT: Imitation learning for multi-agent pathfinding at scale. *Proceedings of the AAAI Conference on Artificial Intelligence*, 39(22):23126–23134, Apr. 2025. doi: 10.1609/aaai.v39i22.34477. URL https://ojs.aaai.org/index.php/AAAI/article/view/34477.

Jacob Austin, Daniel D. Johnson, Jonathan Ho, Daniel Tarlow, and Rianne van den Berg. Structured denoising diffusion models in discrete state-spaces. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan (eds.), *Advances in Neural Information Processing Systems*, volume 34, pp. 17981–17993. Curran Associates, Inc., 2021. URL https://proceedings.neurips.cc/paper_files/paper/2021/file/958c530554f78bcd8e97125b70e6973d-Paper.pdf.

J. Carvalho, A. T. Le, M. Baierl, D. Koert, and J. Peters. Motion planning diffusion: Learning and planning of robot motions with diffusion models. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2023. URL https://arxiv.org/abs/2308.01557.

M. Erdmann and T. Lozano-Perez. On multiple moving objects. In *Proceedings. 1986 IEEE International Conference on Robotics and Automation*, volume 3, pp. 1419–1424, 1986. doi: 10.1109/ROBOT.1986.1087401.

Emiel Hoogeboom, Didrik Nielsen, Priyank Jaini, Patrick Forré, and Max Welling. Argmax flows and multinomial diffusion: learning categorical distributions. In *Proceedings of the 35th International Conference on Neural Information Processing Systems*, NIPS '21, Red Hook, NY, USA, 2021. Curran Associates Inc. ISBN 9781713845393.

Taoan Huang, Jiaoyang Li, Sven Koenig, and Bistra Dilkina. Anytime multi-agent path finding via machine learning-guided large neighborhood search. *Proceedings of the AAAI Conference on Artificial Intelligence*, 36(9):9368–9376, Jun. 2022. doi: 10.1609/aaai.v36i9.21168. URL https://ojs.aaai.org/index.php/AAAI/article/view/21168.

Marc Huber, Günther R. Raidl, and Christian Blum. Learning to select promising initial solutions for large neighborhood search-based multi-agent path finding. In Alexis Quesada-Arencibia, Michael Affenzeller, and Roberto Moreno-Díaz (eds.), *Computer Aided Systems Theory – EUROCAST 2024*, volume 15172 of *LNCS*, pp. 236–250. Springer, 2025. doi: 10.1007/978-3-031-82949-9_22. URL https://www.ac.tuwien.ac.at/files/pub/huber-24.pdf.

Rishabh Jain, Keisuke Okumura, Michael Amir, Pietro Liò, and Amanda Prorok. Pairwise is not enough: Hypergraph neural networks for multi-agent pathfinding. In *International Conference on Learning Representations (ICLR)*, 2026.

Jiaoyang Li, Zhe Chen, Daniel Harabor, Peter J. Stuckey, and Sven Koenig. Anytime multi-agent path finding via large neighborhood search. In Zhi-Hua Zhou (ed.), *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI-21*, pp. 4127–4135. International Joint Conferences on Artificial Intelligence Organization, August 2021a. doi: 10.24963/ijcai.2021/568. URL https://doi.org/10.24963/ijcai.2021/568. Main Track.

Jiaoyang Li, Wheeler Ruml, and Sven Koenig. EECBS: A bounded-suboptimal search for multi-agent path finding. *Proceedings of the AAAI Conference on Artificial Intelligence*, 35(14):12353–12362, May 2021b. doi: 10.1609/aaai.v35i14.17466. URL https://ojs.aaai.org/index.php/AAAI/article/view/17466.

Jiaoyang Li, Zhe Chen, Daniel Harabor, Peter J. Stuckey, and Sven Koenig. MAPF-LNS2: fast repairing for multi-agent path finding via large neighborhood search. In *Thirty-Sixth AAAI Conference on Artificial Intelligence, AAAI 2022, Thirty-Fourth Conference on Innovative Applications of Artificial Intelligence, IAAI 2022, The Twelfth Symposium on Educational Advances in Artificial Intelligence, EAAI 2022 Virtual Event, February 22 - March 1, 2022*, pp. 10256–10265. AAAI Press, 2022. doi: 10.1609/AAAI.V36I9.21266. URL https://doi.org/10.1609/aaai.v36i9.21266.

Qingbiao Li, Weizhe Lin, Zhe Liu, and Amanda Prorok. Message-aware graph attention networks for large-scale multi-robot path planning. *IEEE Robotics Autom. Lett.*, 6(3):5533–5540, 2021c. doi: 10.1109/LRA.2021.3077863. URL https://doi.org/10.1109/LRA.2021.3077863.

Jinhao Liang, Jacob K Christopher, Sven Koenig, and Ferdinando Fioretto. Simultaneous multi-robot motion planning with projected diffusion models. In Aarti Singh, Maryam Fazel, Daniel Hsu, Simon Lacoste-Julien, Felix Berkenkamp, Tegan Maharaj, Kiri Wagstaff, and Jerry Zhu (eds.), *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 37162–37180. PMLR, 13–19 Jul 2025a. URL https://proceedings.mlr.press/v267/liang25e.html.

Jinhao Liang, Sven Koenig, and Ferdinando Fioretto. Discrete-guided diffusion for scalable and safe multi-robot motion planning, 2025b. URL https://arxiv.org/abs/2508.20095.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=Bkg6RiCqY7.

Ziyuan Ma, Yudong Luo, and Hang Ma. Distributed heuristic multi-agent path finding with communication. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 8699–8705, 2021. doi: 10.1109/ICRA48506.2021.9560748.

Alexander Quinn Nichol and Prafulla Dhariwal. Improved denoising diffusion probabilistic models. In Marina Meila and Tong Zhang (eds.), *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pp. 8162–8171. PMLR, 18–24 Jul 2021. URL https://proceedings.mlr.press/v139/nichol21a.html.

Keisuke Okumura. LaCAM: Search-based algorithm for quick multi-agent pathfinding. *Proceedings of the AAAI Conference on Artificial Intelligence*, 37(10):11655–11662, Jun. 2023. doi: 10.1609/aaai.v37i10.26377. URL https://ojs.aaai.org/index.php/AAAI/article/view/26377.

Keisuke Okumura. Engineering LaCAM*: Towards real-time, large-scale, and near-optimal multi-agent pathfinding. In Mehdi Dastani, Jaime Simão Sichman, Natasha Alechina, and Virginia Dignum (eds.), *Proceedings of the 23rd International Conference on Autonomous Agents and Multiagent Systems, AAMAS 2024, Auckland, New Zealand, May 6-10, 2024*, pp. 1501–1509. International Foundation for Autonomous Agents and Multiagent Systems / ACM, 2024. doi: 10.5555/3635637.3663010. URL https://dl.acm.org/doi/10.5555/3635637.3663010.

William Peebles and Saining Xie. Scalable diffusion models with transformers. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 4195–4205, October 2023.

Ethan Perez, Florian Strub, Harm de Vries, Vincent Dumoulin, and Aaron Courville. FiLM: Visual reasoning with a general conditioning layer, 2017. URL https://arxiv.org/abs/1709.07871.

Thomy Phan, Taoan Huang, Bistra Dilkina, and Sven Koenig. Adaptive anytime multi-agent path finding using bandit-based large neighborhood search. *Proceedings of the AAAI Conference on Artificial Intelligence*, 38(16):17514–17522, Mar. 2024. doi: 10.1609/aaai.v38i16.29701. URL https://ojs.aaai.org/index.php/AAAI/article/view/29701.

Guillaume Sartoretti, Justin Kerr, Yunfei Shi, Glenn Wagner, T. K. Satish Kumar, Sven Koenig, and Howie Choset. PRIMAL: Pathfinding via reinforcement and imitation multi-agent learning. *IEEE Robotics and Automation Letters*, 4(3):2378–2385, 2019. doi: 10.1109/LRA.2019.2903261.

Yorai Shaoul, Itamar Mishani, Shivam Vats, Jiaoyang Li, and Maxim Likhachev. Multi-robot motion planning with diffusion models. In Y. Yue, A. Garg, N. Peng, F. Sha, and R. Yu (eds.), *International Conference on Learning Representations*, volume 2025, pp. 95791–95811, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/file/eee6efe709623f36483e3fbb0bb513dd-Paper-Conference.pdf.

Guni Sharon, Roni Stern, Ariel Felner, and Nathan R. Sturtevant. Conflict-based search for optimal multi-agent path finding. In Jörg Hoffmann and Bart Selman (eds.), *Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence, July 22-26, 2012, Toronto, Ontario, Canada*, pp. 563–569. AAAI Press, 2012. doi: 10.1609/AAAI.V26I1.8140. URL https://doi.org/10.1609/aaai.v26i1.8140.

Alexey Skrynnik, Anton Andreychuk, Anatolii Borzilov, Alexander Chernyavskiy, Konstantin Yakovlev, and Aleksandr Panov. POGEMA: A benchmark platform for cooperative multi-agent pathfinding. In Y. Yue, A. Garg, N. Peng, F. Sha, and R. Yu (eds.), *International Conference on Learning Representations*, volume 2025, pp. 5576–5606, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/file/10d19888a94f390e58f922ab3937e1cb-Paper-Conference.pdf.

Roni Stern, Nathan R. Sturtevant, Ariel Felner, Sven Koenig, Hang Ma, Thayne T. Walker, Jiaoyang Li, Dor Atzmon, Liron Cohen, T. K. Satish Kumar, Roman Barták, and Eli Boyarski. Multi-agent pathfinding: Definitions, variants, and benchmarks. In Pavel Surynek and William Yeoh (eds.), *Proceedings of the Twelfth International Symposium on Combinatorial Search, SOCS 2019, Napa, California, 16-17 July 2019*, pp. 151–158. AAAI Press, 2019. doi: 10.1609/SOCS.V10I1.18510. URL https://doi.org/10.1609/socs.v10i1.18510.

Yimin Tang, Xiao Xiong, Jingyi Xi, Jiaoyang Li, Erdem Bıyık, and Sven Koenig. RAILGUN: A unified convolutional policy for multi-agent path finding across different environments and tasks. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 8645–8651, 2025. doi: 10.1109/IROS60139.2025.11247527.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.

Yutong Wang, Bairan Xiang, Shinan Huang, and Guillaume Sartoretti. SCRIMP: Scalable communication for reinforcement- and imitation-learning-based multi-agent pathfinding. In *Proceedings of the 2023 International Conference on Autonomous Agents and Multiagent Systems*, AAMAS '23, pp. 2598–2600, Richland, SC, 2023. International Foundation for Autonomous Agents and Multiagent Systems. ISBN 9781450394321.

Yutong Wang, Tanishq Duhan, Jiaoyang Li, and Guillaume Sartoretti. LNS2+RL: Combining multi-agent reinforcement learning with large neighborhood search in multi-agent path finding. *Proceedings of the AAAI Conference on Artificial Intelligence*, 39(22):23343–23350, Apr. 2025. doi: 10.1609/aaai.v39i22.34501. URL https://ojs.aaai.org/index.php/AAAI/article/view/34501.

Zhongxia Yan and Cathy Wu. Neural neighborhood search for multi-agent path finding. In B. Kim, Y. Yue, S. Chaudhuri, K. Fragkiadaki, M. Khan, and Y. Sun (eds.), *International Conference on Learning Representations*, volume 2024, pp. 48473–48494, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/file/d41f8403e9bb5141bc2c81fad7658185-Paper-Conference.pdf.

# A Outline

This appendix provides additional results, protocol details, and implementation details that complement the main paper. Appendix B presents a controlled PP-Multistart comparison that separates the effect of the learned diffusion initializer from the benefit of repeated PP initializer attempts, and evaluates transfer to MovingAI maps in Appendix B.5. Appendix C reports the complete main experimental results, including benchmark details, evaluation budgets, candidate-generation statistics, and per-setting numerical results. Appendix C.4 provides additional observations on the behavior of DiffLNS and the compared baselines across scene families. Finally, Appendix D describes the implementation details of the training pipeline and denoising network, including expert-data construction, the conditional denoising architecture, supervised losses, the training schedule, and optimization hyperparameters.

# B Additional Experiments

## B.1 Multi-Round PP Initializer Ablation

We further examine whether the improvement of DiffLNS can be explained by repeated PP initializer attempts. To this end, we construct a PP-Multistart baseline under a controlled fixed candidate budget, while replacing the diffusion initializer with the PP initializer. We evaluate both methods on the most congested setting of each benchmark family: *Small Random* with 60 agents, *Medium Maze* with 190 agents, *Medium Room* with 192 agents, *Medium Warehouse* with 252 agents, and *Large Maze* with 312 agents. This experiment is separate from the fixed-time main evaluation and is designed to isolate the effect of initializer quality under the same maximum number of candidate attempts. Both methods use the same candidate-generation budget: batch size 4, at most 5 batches, and at most 20 generated candidates. We report success rate (SR), mean SOC and runtime over each method's successful instances, and the average number of generated candidates.

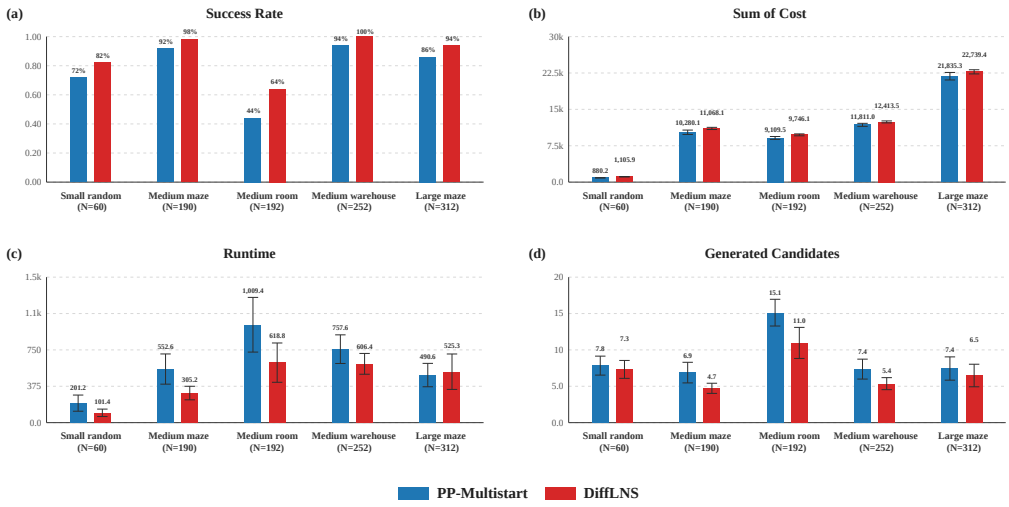



Figure 6: Comparison between DiffLNS and PP-Multistart on the most congested setting of each benchmark family. Both methods use the same multi-round candidate-generation budget. Error bars indicate 95% confidence intervals over test instances.

Fig. 6 shows that increasing the number of PP initializer attempts is not sufficient to match the robustness of DiffLNS in highly congested scenarios. Under the same multi-start budget, DiffLNS achieves higher SR on all five congested settings. The largest improvement appears on *Medium Room* with 192 agents, where DiffLNS improves SR from 44% to 64%. DiffLNS also improves SR from 72% to 82% on *Small Random*, from 92% to 98% on *Medium Maze*, from 94% to 100% on *Medium Warehouse*, and from 86% to 94% on *Large Maze*. These results indicate that learned

diffusion initialization provides more reliable repair seeds than repeated PP initializer restarts, with the advantage becoming especially important when PP-Multistart exhibits a clear drop in success rate.

The SOC results show a different trend from SR. PP-Multistart obtains lower mean SOC than DiffLNS on all five settings, suggesting that PP initializer restarts can produce shorter solutions when they succeed. However, SOC is averaged only over successful instances. Since DiffLNS solves more instances, the successful set of DiffLNS can include harder cases that PP-Multistart fails to solve. These additional solved cases may require longer coordinated paths and increase the average SOC. Therefore, the lower SOC of PP-Multistart should not be interpreted as uniformly better solution quality, but rather as a result computed on a smaller, potentially easier successful subset. The absolute SOC values also increase on larger and more congested maps, reflecting longer path lengths and greater coordination difficulty.

The runtime and candidate statistics further highlight the robustness–efficiency trade-off. The reported runtime includes both initialization and repair. PP-Multistart generally requires more generated candidates than DiffLNS. For example, it uses 15.1 candidates on average on *Medium Room*, compared with 11.0 for DiffLNS, and it also uses more candidates on *Medium Maze*, *Medium Warehouse*, and *Large Maze*. Despite this larger candidate usage, PP-Multistart still obtains lower SR, suggesting that the PP initializer produces less reliable repair seeds than diffusion-generated initializations. Runtime follows a similar pattern in most settings, with PP-Multistart requiring longer total runtime on *Small Random*, *Medium Maze*, *Medium Room*, and *Medium Warehouse*. The exception is *Large Maze*, where PP-Multistart is slightly faster, but still has lower SR and uses more candidates.

These results indicate that the benefit of DiffLNS is not merely due to using more initialization attempts. Under the same multi-start budget, the diffusion initializer provides more structured and repairable starting plans than the PP initializer, leading to higher repair robustness in dense MAPF instances while using fewer candidates in most settings.

### B.2 Initialization Comparison under Fixed LNS2 Repair

To isolate the effect of initialization quality from the full DiffLNS pipeline, we disable multi-sample generation and compare single initializations from D3PM and the PP initializer. Both initializations are passed to the same downstream LNS2 repair procedure under the same 120 s repair budget. Fig. 7 reports the repair success rate (SR) and the LNS2 repair runtime, where runtime is computed only over successful repairs and excludes the cost of initialization generation.

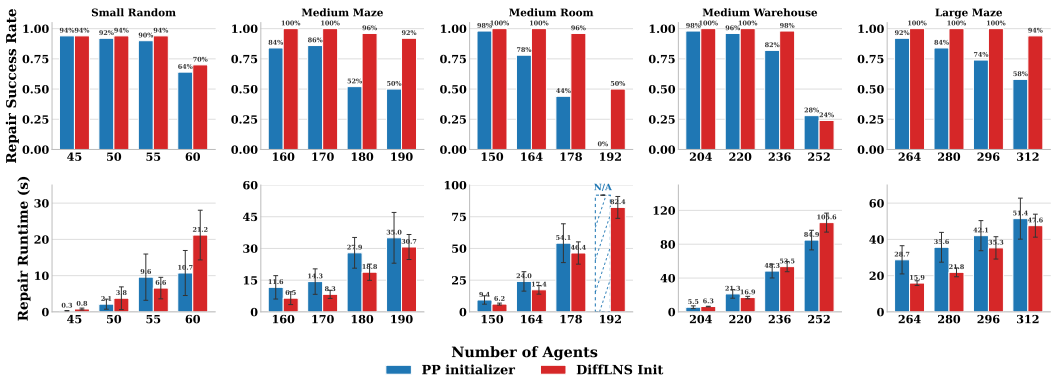



Figure 7: Initialization comparison under fixed LNS2 repair across five benchmark families. Both initialization strategies are repaired by the same LNS2 procedure under the same repair budget. Runtime is computed over successful repairs only, with N/A denoting no successful repairs. Error bars indicate 95% confidence intervals.

Fig. 7 shows that diffusion-generated initializations improve repair success in most dense and congested settings, with larger gains in highly coupled maze and room scenarios, while often leading to comparable or lower LNS2 repair time among successfully repaired instances. Overall, even a

single diffusion-generated initialization provides a more repairable starting point for LNS2, indicating that DiffLNS gains are not solely due to multi-sample generation and selection. Appendix B.1 further shows that D3PM warm-starting outperforms repeated PP initializer attempts under the same multi-start budget.

Higher repairability does not necessarily come with fewer raw conflicts before repair. In our experiments, DiffLNS initializations usually contain more vertex or edge conflicts than those from the PP initializer, yet often achieve higher repair success with comparable or lower repair time. This is consistent with prior work on selecting promising initial solutions for LNS-based MAPF (Huber et al., 2025): initialization quality should be judged by downstream LNS outcomes rather than by pre-repair statistics alone. The PP initializer uses sequential PP+SIPPS, which can produce locally low-conflict and low-cost paths under a fixed priority order but may yield globally rigid plans that are difficult to reorganize through local repair. In contrast, D3PM samples joint action plans from a learned expert trajectory prior, producing more coordinated and structured drafts that can be easier for LNS2 to repair even without fewer raw conflicts.

### B.3 Performance under Longer Time Budgets

We further evaluate whether the feasibility advantage of DiffLNS persists when all methods are given substantially longer computation budgets. We use 25 instances from each of the densest Medium Maze ($N = 190$), Medium Room ($N = 192$), and Medium Warehouse ($N = 252$) settings. DiffLNS repeatedly generates single candidates and assigns at most 120 s of LNS2 repair to each candidate, whereas LNS2, LNS2+RL, and LaCAM3 continue running within the same total wall-clock budget. We record the cumulative success rate (SR) at 1200, 1800, 2400, and 3600 s.

Table 1: Success rates under longer wall-clock budgets. All methods are evaluated on the same instances at the same budget checkpoints.

| Setting | Method | 1200 s | 1800 s | 2400 s | 3600 s |
|---|---|---|---|---|---|
| Maze | DiffLNS | **96%** | **96%** | **100%** | **100%** |
| | LNS2 | 60% | 72% | 84% | 88% |
| | LNS2+RL | 72% | 84% | 92% | 92% |
| | LaCAM3 | 76% | 84% | 96% | 96% |
| Room | DiffLNS | **68%** | **76%** | **80%** | **84%** |
| | LNS2 | 28% | 32% | 32% | 40% |
| | LNS2+RL | 40% | 48% | 52% | 52% |
| | LaCAM3 | 24% | 36% | 52% | 60% |
| Warehouse | DiffLNS | **96%** | **100%** | **100%** | **100%** |
| | LNS2 | 64% | 80% | 80% | 84% |
| | LNS2+RL | 80% | 80% | 80% | 84% |
| | LaCAM3 | 84% | 88% | **100%** | **100%** |

As shown in Table 1, the advantage of DiffLNS is not limited to a short evaluation budget. At 3600 s, DiffLNS achieves success rates of 100%, 84%, and 100% on Maze, Room, and Warehouse, respectively. It remains more successful than LNS2 and LNS2+RL on all three settings, exceeds LaCAM3 on Maze and Room, and ties LaCAM3 on Warehouse. The gap becomes smaller for some methods as the budget increases, but does not disappear in the more repair-intensive Maze and Room settings. These results support the positioning of DiffLNS as a learned initializer that improves feasibility in congested settings, rather than as a uniformly faster solver.

### B.4 Sensitivity to the Planning Horizon $T$

The planning horizon $T$ determines the length of the joint action sequence generated by the diffusion model. During training, we use a fixed horizon of $T = 72$ for all $23 \times 23$ expert instances, approximately three times the map side length. At inference, we set $T = 3\max(H, W)$, where $H$ and $W$ denote the map dimensions, and use the same value for all agent counts within a benchmark family. Thus, $T$ depends on map size rather than on the number of agents.

To examine sensitivity to this choice, we evaluate one diffusion-generated candidate at each horizon on the densest Medium Maze ($N = 190$), Medium Room ($N = 192$), and Medium Warehouse ($N = 252$) settings. Multi-sample generation and candidate selection are disabled, and every candidate receives the same 120 s LNS2 repair budget.

Table 2: DiffLNS success rate under different planning horizons.

| Setting | Map size | Planning horizon $T$ | | | |
|---|---|---|---|---|---|
| | | $T_1$ | $T_2$ | $T_3$ | $T_4$ |
| Maze | $25 \times 25$ | 50: 64% | 64: 84% | 80: **92%** | 96: 90% |
| Room | $23 \times 23$ | 46: 28% | 64: 56% | 80: 56% | 96: **60%** |
| Warehouse | $25 \times 25$ | 50: 24% | 64: **44%** | 80: 42% | 96: **44%** |

Table 2 shows that horizons close to twice the map side length are often insufficient: increasing $T$ from 50 to 80 raises the Maze success rate from 64% to 92%, while increasing it from 46 to 64 raises the Room success rate from 28% to 56%. Once the horizon is sufficiently large, performance generally stabilizes, with only modest non-monotonic fluctuations. In particular, further increasing $T$ does not consistently improve success. These results motivate using a map-scale-dependent horizon that provides adequate temporal coverage, while keeping it fixed across agent counts within each benchmark family.

### B.5 Generalization to MovingAI Maps

To assess generalization beyond POGEMA-generated environments, we evaluate DiffLNS on three standard MovingAI maps (Stern et al., 2019):[1] `maze-32-32-4` (288 agents), `room-32-32-4` (330 agents), and `random-32-32-20` (480 agents), using 20 paired instances per map.

Both methods use the same LNS2 repair settings and a 600 s total budget; SR is recorded at 120, 300, and 600 s within the same run. DiffLNS allows up to 180 s of repair per candidate, subject to the remaining budget. Its timing includes generation, path completion, repair, and validation. LNS2 uses its native first-solution runtime.

Table 3: Success rates on MovingAI maps (20 paired instances per map). Time thresholds refer to a single 600 s run.

| Map | Agents | Method | SR at 120 s | SR at 300 s | SR at 600 s |
|---|---|---|---|---|---|
| `maze-32-32-4` | 288 | DiffLNS | **100%** | **100%** | **100%** |
| | | LNS2 | 60% | 80% | 95% |
| `room-32-32-4` | 330 | DiffLNS | **100%** | **100%** | **100%** |
| | | LNS2 | 35% | 70% | 80% |
| `random-32-32-20` | 480 | DiffLNS | **65%** | **100%** | **100%** |
| | | LNS2 | 35% | 60% | 75% |

Table 3 shows that DiffLNS retains a success-rate advantage on all three tested MovingAI maps, extending its effectiveness beyond POGEMA-generated layouts. These results suggest that the learned trajectory prior captures coordination patterns that remain useful on external benchmark topologies under the same downstream LNS2 repair settings. The advantage persists on Room and Random even with the longer time limit, indicating that the benefit includes solving cases that LNS2 still fails to resolve within the evaluation budget. Successful transfer to the 480-agent Random extension further supports the applicability of learned initialization to larger teams on these external layouts.

[1] `https://movingai.com/benchmarks/mapf/index.html`

# C Complete Main Experiment Results

## C.1 Supplementary Experimental Details

This appendix supplements the main experimental section with additional protocol details and per-setting statistics for interpreting Table 8, Fig. 3, and Fig. 7. Table 4 summarizes the benchmark families, map sizes, obstacle-density ranges, and evaluated agent cardinalities. Table 5 reports the number of evaluated instances and the fixed evaluation budgets for the fixed-budget baselines.

**Baselines and implementation details.** The main experiments compare DiffLNS with four MAPF baselines: LNS2 (Li et al., 2022)[2], LNS2+RL (Wang et al., 2025)[3], HMAGAT (Jain et al., 2026)[4], and LaCAM3 (Okumura, 2024)[5]. LNS2 is a strong classical repair-based solver. LNS2+RL is a learning-augmented variant of LNS2. HMAGAT is a state-of-the-art learning-based MAPF solver; in our evaluation, we use its k-means variant and set the maximum episode length to 512 steps. LaCAM3 is a strong classical anytime solver. After finding an initial feasible solution, LaCAM3 continues to improve the solution cost until the given time limit is reached.

For DiffLNS, we generate candidates in batches, with $M = 4$ diffusion samples per batch. Each DiffLNS candidate receives at most 120 s of LNS2 repair, subject to the remaining evaluation budget. DiffLNS completes repair of the current batch and returns the feasible candidate with the lowest SOC. A new batch is sampled only if the current batch has no feasible candidate and the evaluation budget permits. For DiffLNS, LNS2, LNS2+RL, and HMAGAT, we use fixed per-setting time limits that increase with scene difficulty, ranging from 180 s to 1200 s, as reported in Table 5. An instance is counted as a failure for all fixed-budget methods if the time limit is reached before a feasible solution is found. For all LNS-based methods, including DiffLNS, LNS2, and LNS2+RL, the repair neighborhood size is fixed to 8.

**Licenses of existing assets.** We use publicly available implementations only for evaluation and baseline comparison. MAPF-LNS2 is released under the USC Research License; LNS2+RL is released under the MIT License; HMAGAT builds on MAGAT, whose included implementation is released under the MIT License; LaCAM3 is released under the MIT License; and POGEMA is released under the MIT License. We cite the corresponding papers and provide the official repository links above.

**Hardware and parallelization.** All methods were evaluated on machines equipped with NVIDIA L40S GPUs and Intel Xeon Gold 6348 CPUs at 2.60 GHz. Evaluation instances were parallelized across independent worker processes, with each process handling one MAPF instance at a time. Each worker was allocated 8 logical CPU threads. Methods requiring GPU acceleration were assigned one NVIDIA L40S GPU per worker, while CPU-only methods used the same 8-thread CPU setting without GPU acceleration. This protocol ensures that all methods are evaluated under the same per-instance CPU resource allocation, with GPU resources provided only to methods that require neural inference.

**Evaluation metrics and runtime protocol.** We report success rate (SR), the pairwise common-success SOC ratio relative to DiffLNS, and average runtime. SR is the percentage of instances solved with collision-free paths to all assigned goals within the given budget. For a setting $s$ and baseline $b$, let $\mathcal{C}_b(s)$ denote the instances solved by both $b$ and DiffLNS under their respective evaluation budgets. We define

$$R_b(s) = \frac{\frac{1}{|\mathcal{C}_b(s)|}\sum_{i\in\mathcal{C}_b(s)} \mathrm{SOC}_b(i)}{\frac{1}{|\mathcal{C}_b(s)|}\sum_{i\in\mathcal{C}_b(s)} \mathrm{SOC}_{\mathrm{DiffLNS}}(i)}. \tag{11}$$

Thus, the numerator and denominator are mean SOCs on exactly the same instances. DiffLNS is shown as the reference at 1; $R_b(s) > 1$ favors DiffLNS, and $R_b(s) < 1$ favors baseline $b$. This pairing avoids comparing averages over different successful subsets for the two methods. The subset

[2] https://github.com/Jiaoyang-Li/MAPF-LNS2
[3] https://github.com/marmotlab/LNS2-RL
[4] https://github.com/proroklab/hmagat
[5] https://github.com/Kei18/lacam3

$\mathcal{C}_b(s)$ can differ across baselines, so these ratios support separate comparisons with DiffLNS rather than a ranking among baselines. They describe solution quality conditional on joint success and are reported alongside SR. Runtime is averaged over all instances using the actual elapsed time of each run, including failed runs. For fixed-budget solvers, failed runs may terminate either at the wall-clock time limit or at a method-specific stopping condition. In particular, HMAGAT terminates when either the wall-clock limit or the 512-step episode limit is reached, whichever comes first. For DiffLNS, runtime includes both diffusion generation and downstream LNS2 repair. All reported evaluation runtimes exclude the one-time costs of neural training and expert-data generation, which are reported separately in Appendix C.2.

DiffLNS, LNS2, LNS2+RL, and HMAGAT are evaluated under fixed per-setting time budgets, where each setting is defined by a benchmark family and an agent cardinality. These budgets are assigned according to setting difficulty estimated from preliminary pilot runs: larger or more congested settings are given longer budgets, while easier settings use shorter budgets to avoid unnecessary computation. Within each setting, all fixed-budget methods use the same time limit, and timeout before finding a feasible solution is counted as failure.

LaCAM3 is evaluated under a matched per-instance time budget rather than this fixed-budget protocol. Specifically, for each instance, the time limit of LaCAM3 is set to the actual runtime of DiffLNS on the same instance. This protocol allows LaCAM3 to be used as a matched-time reference for DiffLNS in SR and the pairwise common-success SOC ratio. Since this protocol differs from the fixed-budget setting, LaCAM3 is not directly compared with the other fixed-budget methods, and its runtime is omitted from the runtime comparison.

**Benchmark statistics.** All evaluation instances are randomly generated with POGEMA (Skrynnik et al., 2025)[6]. Example test maps from the evaluated benchmark families are shown in Fig. 8. The reported agent densities are computed with respect to free cells, i.e., as the ratio between the number of agents and the number of non-obstacle cells. The instance counts shown in Table 5 are the actual numbers of evaluated MAPF instances for each agent cardinality. The listed time limits correspond to the fixed-budget methods, namely DiffLNS, LNS2, LNS2+RL, and HMAGAT. LaCAM3 is excluded from this table because it is evaluated under matched per-instance budgets determined by the runtime of DiffLNS on the same instance.

Table 4: Benchmark families used in the main experiments. Agent density is computed as the number of agents divided by the number of traversable cells. The reported agent-density ranges span the four evaluated agent cardinalities.

| Map | Size (H × W) | Obstacle Density | Agent Density | Agent Cardinalities |
|---|---|---|---|---|
| Small Random | $10 \times 10$ | 17.5% | 54.4–72.4% | 45 / 50 / 55 / 60 |
| Medium Maze | $25 \times 25$ | 27.4–36.5% | 38.0–45.4% | 160 / 170 / 180 / 190 |
| Medium Room | $23 \times 23$ | 31.9–35.0% | 42.8–54.5% | 150 / 164 / 178 / 192 |
| Medium Warehouse | $25 \times 25$ | 34.6% | 49.9–61.6% | 204 / 220 / 236 / 252 |
| Large Maze | $33 \times 33$ | 29.3–36.8% | 36.4–42.7% | 264 / 280 / 296 / 312 |

Table 5: Evaluation budgets for the main experiments. Levels 1–4 correspond to the four agent cardinalities listed in Table 4, from low to high density. Each entry reports the number of cases and the corresponding time limit. The listed time limits apply to the fixed-budget methods (DiffLNS, LNS2, LNS2+RL, and HMAGAT).

| Map | Level 1 | Level 2 | Level 3 | Level 4 |
|---|---|---|---|---|
| Small Random | 100 cases / 180 s | 100 cases / 180 s | 100 cases / 180 s | 100 cases / 240 s |
| Medium Maze | 100 cases / 240 s | 100 cases / 240 s | 50 cases / 480 s | 50 cases / 480 s |
| Medium Room | 100 cases / 240 s | 100 cases / 240 s | 50 cases / 480 s | 50 cases / 1200 s |
| Medium Warehouse | 100 cases / 240 s | 100 cases / 240 s | 50 cases / 480 s | 50 cases / 1200 s |
| Large Maze | 50 cases / 360 s | 50 cases / 360 s | 50 cases / 600 s | 50 cases / 1200 s |

[6] https://github.com/CognitiveAISystems/pogema

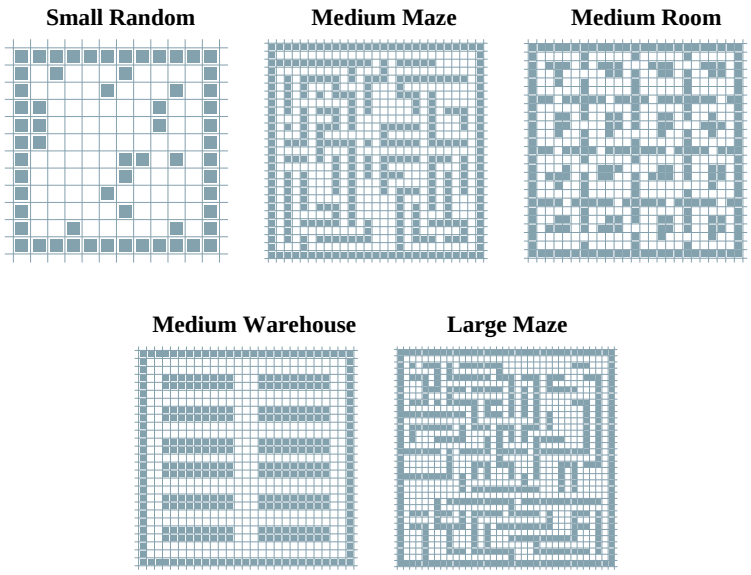



Figure 8: Example test maps generated with POGEMA for the evaluated benchmark families. These maps illustrate the obstacle layouts and structural differences across the random, maze, room, and warehouse environments used in our experiments.

### C.2 Offline Training Costs

The runtime results reported in the main experiments measure test-time computation on previously unseen instances. They do not include the one-time offline costs of generating expert demonstrations or training neural models. We report these costs separately because they are incurred once and amortized over subsequent inference, whereas classical solvers incur search computation independently for every test instance.

**Training costs of learning-based methods.** Table 6 separates the offline training costs of the three learning-based methods in our comparison. DiffLNS was trained for 400 epochs using BF16 distributed data-parallel training on eight NVIDIA A100 GPUs, requiring approximately 26 hours of wall-clock time. For LNS2+RL and HMAGAT, we report the training protocols and costs stated in their respective publications. LNS2 and LaCAM3 require no neural-network training.

Table 6: Offline training costs of the learning-based methods. GPU-hours are computed as the number of GPUs multiplied by the reported wall-clock training time.

| **Method** | **Training protocol** | **Reported compute** |
|---|---|---|
| DiffLNS | 400 epochs with BF16 DDP | 26 h on 8 NVIDIA A100 GPUs (approximately 208 GPU-hours) |
| LNS2+RL (Wang et al., 2025) | Two curriculum stages of $7 \times 10^7$ environment steps each | 136 h per stage on one RTX 3090, with 32 CPU data-collection workers (approximately 272 GPU-hours in total) |
| HMAGAT (Jain et al., 2026) | 200-epoch main training, 20-epoch quality post-training, and 50-epoch temperature-module training | $100\,\text{h} + 30\,\text{h} + 3\,\text{h}$ on one NVIDIA L40S (approximately 133 GPU-hours in total) |

These GPU-hour values provide transparent accounting but should not be interpreted as hardware-normalized training efficiency, because the methods use different GPU generations, CPU resources,

Table 7: Average number of generated candidates used by DiffLNS. For each scenario family, Levels 1–4 denote increasing agent density.

| Scenario | Level 1 | Level 2 | Level 3 | Level 4 |
|---|---|---|---|---|
| Small Random | 4.64 | 4.68 | 5.52 | 7.32 |
| Medium Maze | 4.00 | 4.16 | 4.24 | 4.72 |
| Medium Room | 4.00 | 4.00 | 4.16 | 10.96 |
| Medium Warehouse | 4.00 | 4.00 | 4.00 | 5.36 |
| Large Maze | 4.00 | 4.08 | 4.96 | 6.48 |

data pipelines, and training objectives. We therefore report offline training, expert-data generation, and test-time inference costs separately rather than combining them into a single runtime measure.

### C.3 Candidate Generation Statistics

Table 7 reports the average number of generated candidates used by DiffLNS under the iterative inference strategy. As agent density and coordination difficulty increase, DiffLNS typically needs more candidate generations to obtain a feasible repaired solution, which accounts for part of the runtime increase observed in the main results.

### C.4 Additional Observations on Main Results

Table 8 complements the main results with per-setting comparisons. SOC ratios below refer to pairwise common-success subsets and do not rank baselines against one another.

**DiffLNS.** DiffLNS matches or exceeds all compared methods in SR across the 20 settings, with particularly strong performance in congested maze and warehouse maps. In both maze families, all four baselines have SOC-ratio point estimates above 1 at every agent count, indicating lower paired mean SOC for DiffLNS. Across all settings, paired SOC-ratio point estimates favor DiffLNS against LNS2 in 18 cases, LNS2+RL in 14, HMAGAT in all 20, and LaCAM3 in 15. Thus, its feasibility gains are accompanied by lower mean SOC on many common successful subsets. These conditional quality comparisons complement SR, which captures differences in feasibility. The strong results in congested maps suggest that learned initialization is particularly useful when global coordination and repair quality are bottlenecks. These gains come with additional diffusion-generation and repair costs.

**LNS2.** LNS2 is often among the fastest fixed-budget methods, but its SR declines with congestion. Its paired SOC ratio is below 1 only on *Small Random* with 45 and 50 agents; the other 18 settings favor DiffLNS. Its runtime advantage therefore does not imply higher feasibility or lower paired SOC in the more congested settings.

**LNS2+RL.** LNS2+RL improves on LNS2's SR in harder settings, particularly dense maze and warehouse maps. We use the original paper's best-performing setting: neighborhood size $8$, switch threshold $\rho = 0.3$, stop-MARL-planning threshold $t_l = 1.2$, and maximum length $t_h = 2.2$. Its paired SOC ratio is below 1 in all four *Small Random* settings and the lowest-agent-count *Medium Room* and *Medium Warehouse* settings; the other 14 favor DiffLNS. Its repair-side improvements complement DiffLNS's learned initialization, suggesting a potential benefit from combining the two. Although learning-guided repair narrows the feasibility gap in dense settings, it does not match DiffLNS's strongest SR performance in these experiments.

**HMAGAT.** HMAGAT is relatively fast and stable in room and warehouse maps but struggles in maze environments, especially *Large Maze*. Its paired SOC ratios range from 1.081 to 2.161, favoring DiffLNS in all 20 settings. The 512-step episode limit can terminate unfinished rollouts early, contributing to its lower average runtime. Thus, its speed in some structured layouts is accompanied by higher paired solution costs relative to DiffLNS.

Table 8: Complete main experiment results. Higher success rate is better, while lower SOC ratios and runtime are better. Each baseline's SOC ratio is its mean SOC divided by DiffLNS's mean SOC on instances solved by both methods in that setting; DiffLNS is the reference at 1. Ratios above 1 favor DiffLNS. Common successful subsets differ by baseline, so the ratios do not rank baselines against one another. The symbols ↑ and ↓ indicate that larger and smaller values are better, respectively. For DiffLNS, LNS2, LNS2+RL, and HMAGAT, runtime is averaged over all evaluated instances using the actual elapsed time of each run, including failed runs. HMAGAT terminates when either the wall-clock limit or the 512-step episode limit is reached. The evaluation time limits are shown in parentheses in the runtime headers for each scene and agent cardinality. For LaCAM3, runtime is omitted because LaCAM3 is evaluated under per-instance budgets matched to DiffLNS.

| **Methods** | **Small Random:** $10 \times 10$ **world size with 17.5% static obstacle rate** | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | **Success Rate↑** | | | | **SOC Ratio↓** | | | | **Runtime↓** | | | |
| | N=45 | N=50 | N=55 | N=60 | N=45 | N=50 | N=55 | N=60 | N=45 (180s) | N=50 (180s) | N=55 (180s) | N=60 (240s) |
| DiffLNS | **96%** | **96%** | **92%** | **81%** | 1.000 | 1.000 | 1.000 | 1.000 | 11.8 | 26.1 | 55.5 | 126.3 |
| LNS2 | **96%** | 93% | 82% | 62% | 0.948 | 0.988 | 1.001 | 1.017 | 8.5 | 15.1 | 38.1 | 102.5 |
| LNS2+RL | 95% | 92% | 84% | 64% | 0.897 | 0.915 | 0.907 | 0.951 | 11.6 | 18.5 | 38.3 | 97.0 |
| HMAGAT | 91% | 87% | 80% | 77% | 1.133 | 1.355 | 1.752 | 2.156 | **5.9** | **7.8** | **11.6** | **13.5** |
| LaCAM3 | 93% | 92% | 76% | 66% | 0.816 | 0.945 | 1.008 | 1.098 | – | – | – | – |
| | **Medium Maze:** $25 \times 25$ **world size with 32.8% static obstacle rate** | | | | | | | | | | | |
| | **Success Rate↑** | | | | **SOC Ratio↓** | | | | **Runtime↓** | | | |
| | N=160 | N=170 | N=180 | N=190 | N=160 | N=170 | N=180 | N=190 | N=160 (240s) | N=170 (240s) | N=180 (480s) | N=190 (480s) |
| DiffLNS | **100%** | **99%** | **100%** | **98%** | 1.000 | 1.000 | 1.000 | 1.000 | 92.1 | 76.8 | 231.0 | 308.7 |
| LNS2 | 89% | 81% | 68% | 60% | 1.025 | 1.070 | 1.111 | 1.082 | 39.8 | 64.1 | 184.8 | 240.4 |
| LNS2+RL | 94% | 84% | 88% | 70% | 1.041 | 1.081 | 1.133 | 1.147 | **29.1** | **55.7** | 104.8 | 183.8 |
| HMAGAT | 70% | 55% | 66% | 38% | 1.579 | 1.846 | 2.161 | 1.993 | 52.9 | 67.4 | **65.3** | **79.1** |
| LaCAM3 | 93% | 86% | 64% | 34% | 1.164 | 1.365 | 1.625 | 1.626 | – | – | – | – |
| | **Medium Room:** $23 \times 23$ **world size with 33.5% static obstacle rate** | | | | | | | | | | | |
| | **Success Rate↑** | | | | **SOC Ratio↓** | | | | **Runtime↓** | | | |
| | N=150 | N=164 | N=178 | N=192 | N=150 | N=164 | N=178 | N=192 | N=150 (240s) | N=164 (240s) | N=178 (480s) | N=192 (1200s) |
| DiffLNS | **100%** | **100%** | **100%** | **64%** | 1.000 | 1.000 | 1.000 | 1.000 | 31.9 | 99.5 | 235.5 | 828.0 |
| LNS2 | 97% | 74% | 54% | 24% | 1.011 | 1.030 | 1.028 | 1.013 | 16.2 | 88.2 | 259.1 | 973.2 |
| LNS2+RL | 98% | 86% | 76% | 48% | 0.988 | 1.056 | 1.092 | 1.068 | **15.7** | 62.5 | 179.2 | 709.3 |
| HMAGAT | 92% | 93% | 86% | 62% | 1.382 | 1.699 | 1.856 | 1.937 | 26.6 | **31.0** | **35.4** | **43.0** |
| LaCAM3 | **100%** | 98% | 60% | 6% | 0.966 | 1.159 | 1.186 | 1.356 | – | – | – | – |
| | **Medium Warehouse:** $25 \times 25$ **world size with 34.6% static obstacle rate** | | | | | | | | | | | |
| | **Success Rate↑** | | | | **SOC Ratio↓** | | | | **Runtime↓** | | | |
| | N=204 | N=220 | N=236 | N=252 | N=204 | N=220 | N=236 | N=252 | N=204 (240s) | N=220 (240s) | N=236 (480s) | N=252 (1200s) |
| DiffLNS | **100%** | **100%** | **100%** | **100%** | 1.000 | 1.000 | 1.000 | 1.000 | 37.7 | 76.3 | 220.4 | 606.4 |
| LNS2 | 99% | 96% | 94% | 60% | 1.014 | 1.045 | 1.062 | 1.054 | **9.5** | 29.9 | 103.7 | 605.5 |
| LNS2+RL | 99% | 99% | 94% | 80% | 0.977 | 1.048 | 1.095 | 1.093 | 11.7 | **21.9** | 87.6 | 416.5 |
| HMAGAT | 99% | 97% | **100%** | 96% | 1.081 | 1.234 | 1.444 | 1.532 | 35.9 | 41.8 | **47.9** | **62.4** |
| LaCAM3 | **100%** | 99% | 68% | 10% | 0.862 | 0.972 | 1.073 | 1.107 | – | – | – | – |
| | **Large Maze:** $33 \times 33$ **world size with 33.0% static obstacle rate** | | | | | | | | | | | |
| | **Success Rate↑** | | | | **SOC Ratio↓** | | | | **Runtime↓** | | | |
| | N=264 | N=280 | N=296 | N=312 | N=264 | N=280 | N=296 | N=312 | N=264 (360s) | N=280 (360s) | N=296 (600s) | N=312 (1200s) |
| DiffLNS | **100%** | **100%** | **98%** | **92%** | 1.000 | 1.000 | 1.000 | 1.000 | 121.7 | 184.8 | 331.5 | 565.8 |
| LNS2 | 96% | 86% | 78% | 74% | 1.103 | 1.165 | 1.151 | 1.138 | 43.2 | 87.9 | 186.6 | 407.6 |
| LNS2+RL | 96% | 94% | 96% | 86% | 1.130 | 1.205 | 1.213 | 1.195 | **37.3** | **53.0** | **76.5** | 265.0 |
| HMAGAT | 52% | 58% | 48% | 46% | 1.638 | 1.800 | 1.834 | 2.025 | 95.4 | 104.0 | 146.5 | **182.3** |
| LaCAM3 | 44% | 28% | 34% | 8% | 1.175 | 1.121 | 1.242 | 1.350 | – | – | – | – |

**LaCAM3.** LaCAM3 uses per-instance budgets matched to DiffLNS's actual runtime. Its paired SOC ratio is below 1 in the two lowest-agent-count *Small Random* settings, the lowest-agent-count *Medium Room* setting, and the two lowest-agent-count *Medium Warehouse* settings. The other 15 favor DiffLNS, including all maze settings. Its SR declines sharply in dense regimes, especially *Large Maze* and the hardest *Medium Warehouse* setting, supporting DiffLNS's advantage under congestion.

# D Training and Denoising Network Details

This appendix specifies the training data, denoising architecture, losses, and optimization settings.

## D.1 Training Data and Distributed Setup

Training data are constructed from procedurally generated POGEMA (Skrynnik et al., 2025) MAPF instances. We use three scene families: maze, room, and warehouse. All training maps have size $23 \times 23$, and the dataset contains three agent cardinalities, $N \in \{32, 64, 96\}$. The dataset is evenly split across the three scene types and contains 4.5K instances with 32 agents, 9K instances with 64 agents, and 9K instances with 96 agents. For each instance, start and goal locations are randomly generated, and obstacle densities are sampled to produce diverse MAPF configurations.

LaCAM3 (Okumura, 2024) generates expert demonstrations using staged timeouts of 1, 5, 10, and 60 s, escalating only after failure; only successful rollouts are retained. Table 9 reports generation statistics. This offline cost is excluded from test-time runtime. LaCAM3 separately serves as a matched-time evaluation baseline: D3PM learns from its output trajectories without access to its search procedure, while LNS2 enforces feasibility using the same repair operator across our controlled initialization comparisons.

Table 9: Offline LaCAM3 expert-data generation statistics.

| **Statistic** | **Value** |
|---|---|
| Timeout schedule | 1 / 5 / 10 / 60 s |
| Maximum single-call budget | 60 s |
| Maximum cumulative budget | 76 s |
| Attempted instances | 22,508 |
| Retained demonstrations | 22,500 |
| Discarded instances | 8 |
| Acceptance rate | 99.964% |
| Generation wall time | 5.28 h |

Each mini-batch contains one fixed-$N$ subset; the three subsets alternate round-robin within each epoch. Training uses batch size 8 per GPU and distributed data parallelism on eight NVIDIA A100 GPUs on one node, taking approximately 26 hours for 400 epochs. PyTorch and NumPy seeds are 42. BF16 autocasting and activation checkpointing reduce memory usage.

## D.2 Model and Diffusion Configuration

The conditional clean-state predictor $\tilde{p}_\theta(\mathbf{x}_0 \mid \mathbf{x}_k, \mathbf{c})$ uses a timestep-conditioned transformer inspired by DiT (Peebles & Xie, 2023). Its MAPF-specific backbone processes agent-time action tokens through temporal attention, diffusion-aware sparse social attention, and trajectory-conditioned environment sensing.

Given a noisy joint action tensor $\mathbf{x}_k \in [0,1]^{N\times T\times C}$, where $C = 5$ corresponds to `stay`, `up`, `down`, `left`, and `right`, the network outputs action logits $f_\theta(\mathbf{x}_k, \mathbf{c}, k) \in \mathbb{R}^{N\times T\times C}$. The predicted clean distribution is

$$\tilde{p}_\theta(\mathbf{x}_0 \mid \mathbf{x}_k, \mathbf{c}) = \prod_{i=1}^{N}\prod_{\tau=1}^{T} \mathrm{Cat}(\mathbf{x}_{0,i,\tau,:}; \mathrm{softmax}(f_\theta(\mathbf{x}_k, \mathbf{c}, k)_{i,\tau,:})) \,. \tag{12}$$

We use $K = 100$ diffusion steps with a cosine noise schedule (Nichol & Dhariwal, 2021). During training, the diffusion step is sampled uniformly as $k \sim \mathcal{U}\{1, \ldots, K\}$.

**Uniform forward transition matrix.** We use the standard uniform transition kernel from the D3PM family (Austin et al., 2021):

$$\mathbf{Q}_k = \alpha_k \mathbf{I} + (1 - \alpha_k)\frac{1}{C}\mathbf{1}\mathbf{1}^\top. \tag{13}$$

The kernel gradually mixes categorical actions toward a uniform distribution without imposing an ordering on action classes.

**Condition encoding.** The MAPF condition $\mathbf{c}$ contains the obstacle map, goal map, start and goal locations, map size, and a global density feature. Let the grid map have width $W$ and height $H$, and denote the map-size vector by $\mathbf{m} = (W, H)$. Let $V_{\text{free}} = \{(u, v) : G_{u,v} = 0\}$ denote the set of traversable cells, where $G_{u,v} = 0$ indicates a free cell and $G_{u,v} = 1$ indicates an obstacle. Start and goal locations are represented in normalized map coordinates, $\mathbf{s}_i, \mathbf{g}_i \in [-1, 1]^2$. The global agent-density feature is

$$\rho = \log\left(1 + \frac{N}{|V_{\text{free}}|}\right). \tag{14}$$

The diffusion step is encoded by a sinusoidal embedding $\mathrm{PE}(k)$ (Vaswani et al., 2017). The global condition is computed as

$$\mathbf{c}_{\text{g}} = \psi_{\text{g}}\left([\phi_k(\mathrm{PE}(k)),\ \phi_m(\log \mathbf{m}) + \phi_\rho(\rho)]\right), \tag{15}$$

where $\phi_k$, $\phi_m$, $\phi_\rho$, and $\psi_{\text{g}}$ are learned projections. For each agent, the agent-wise condition is

$$\mathbf{c}_i = \psi_{\text{n}}\left([\mathbf{c}_{\text{g}}, \phi_s(\mathbf{s}_i), \phi_g(\mathbf{g}_i), \phi_r(\mathbf{g}_i - \mathbf{s}_i)]\right). \tag{16}$$

The global condition also predicts a scale gate for the map pyramid, later used to weight multi-scale map features in Eq. (32):

$$\boldsymbol{\eta} = \mathrm{softmax}(\psi_{\text{scale}}(\mathbf{c}_{\text{g}})), \qquad \boldsymbol{\eta} \in \mathbb{R}^S. \tag{17}$$

**Initial denoising tokens.** The network first embeds each noisy one-hot action token and adds start, goal, and relative-goal embeddings. The initial denoising token for agent $i$ at timestep $\tau$ is

$$\mathbf{x}_{i,\tau}^{(0)} = \phi_a(\mathbf{x}_{k,i,\tau,:}) + \phi_s(\mathbf{s}_i) + \phi_g(\mathbf{g}_i) + \phi_r(\mathbf{g}_i - \mathbf{s}_i), \qquad \mathbf{x}_{i,\tau}^{(0)} \in \mathbb{R}^D. \tag{18}$$

Here the superscript in $\mathbf{x}^{(\ell)}$ denotes the denoising-network layer index, while the subscript $k$ in $\mathbf{x}_k$ denotes the diffusion step.

**Environment feature pyramid.** The environment input is a three-channel map containing obstacle cells, free cells, and goal cells. A lightweight convolutional pyramid produces multi-scale feature maps

$$\{\mathbf{F}^{(s)}\}_{s=1}^{S} = E_{\text{map}}(\mathbf{G}; \mathbf{c}_{\text{g}}), \qquad \mathbf{F}^{(s)} \in \mathbb{R}^{D \times H_s \times W_s}. \tag{19}$$

Each convolutional stage is modulated by the global condition through FiLM (Perez et al., 2017):

$$\mathrm{FiLM}(\mathbf{F}; \mathbf{c}_{\text{g}}) = (1 + \boldsymbol{\gamma}(\mathbf{c}_{\text{g}})) \odot \mathbf{F} + \boldsymbol{\beta}(\mathbf{c}_{\text{g}}). \tag{20}$$

The pyramid uses two strided downsampling stages and a partial upsampling path, and each scale is projected to the hidden dimension $D$.

**Inferred trajectory and sparse social graph.** At every diffusion step, the network constructs a reference trajectory from the current noisy action tensor. Training and default reverse sampling use hard one-hot action tokens, which are passed directly to the network. The reference trajectory is used for interaction modeling and map-feature sampling and need not satisfy MAPF constraints. Let $\Delta(a) \in \mathbb{R}^2$ be the grid displacement of action $a$ in row-column coordinates. The token-weighted displacement is

$$\bar{\Delta}_{i,\tau} = \sum_{a=1}^{C} \mathbf{x}_{k,i,\tau,a} \Delta(a). \tag{21}$$

For one-hot inputs, this sum selects the sampled action displacement. Since start locations are normalized, the displacement is rescaled by the map size before accumulation. The inferred position is then

$$\mathbf{p}_{i,\tau}^{\text{inf}} = \mathbf{s}_i + \sum_{u=1}^{\tau} \mathrm{Norm}_{\mathbf{m}}\left(\bar{\Delta}_{i,u}\right), \tag{22}$$

where $\mathrm{Norm}_{\mathbf{m}}(\cdot)$ converts row-column grid displacement into the normalized coordinate system according to the map size $\mathbf{m} = (W, H)$. The resulting coordinates are clamped to $[-1.2, 1.2]$.

The implementation also computes the normalized entropy of the same input tensor for the radius factor in Eq. (31):

$$e_{i,\tau} = -\frac{1}{\log C} \sum_{a=1}^{C} \mathbf{x}_{k,i,\tau,a} \log \mathbf{x}_{k,i,\tau,a}. \tag{23}$$

For hard one-hot inputs, $e_{i,\tau} = 0$ up to numerical stabilization, so this factor does not provide uncertainty-dependent radius adaptation. The step-dependent social neighborhood below is controlled separately by $k$.

The sparse social graph is constructed from these reference trajectories. For each pair of agents, we compute their minimum inferred-trajectory distance:

$$d_{ij} = \min_{\tau=1,\dots,T} \left\| \mathbf{p}_{i,\tau}^{\mathrm{inf}} - \mathbf{p}_{j,\tau}^{\mathrm{inf}} \right\|_1. \tag{24}$$

Let $M_k$ be a clipped non-self neighbor count that increases with the noise level, with the non-self neighbor ratio scheduled from $0.10$ to $0.25$. For each agent $i$, $\mathcal{N}_k(i)$ contains agent $i$ itself and the $M_k$ nearest non-self agents according to $d_{ij}$. Invalid or padded neighbors are masked before the attention softmax.

**Interaction backbone.** The backbone consists of $L = 4$ interaction blocks. Each block contains temporal attention, sparse social attention, deformable environment sensing, and a feed-forward network. Adaptive layer normalization uses the agent-wise condition:

$$\mathrm{AdaLN}(\mathbf{x}_{i,\tau}; \mathbf{c}_i) = (1 + \boldsymbol{\gamma}_i) \odot \mathrm{LN}(\mathbf{x}_{i,\tau}) + \boldsymbol{\beta}_i, \qquad [\boldsymbol{\gamma}_i, \boldsymbol{\beta}_i] = \psi_{\mathrm{ada}}(\mathbf{c}_i). \tag{25}$$

For block $\ell$, the residual updates are

$$\begin{aligned}
\mathbf{x}^{(\ell,1)} &= \mathbf{x}^{(\ell)} + \mathrm{TempAttn}\left(\mathrm{AdaLN}(\mathbf{x}^{(\ell)}, \mathbf{c})\right), \\
\mathbf{x}^{(\ell,2)} &= \mathbf{x}^{(\ell,1)} + \mathrm{SocialAttn}\left(\mathrm{AdaLN}(\mathbf{x}^{(\ell,1)}, \mathbf{c}), \mathbf{p}^{\mathrm{inf}}, \mathcal{N}_k\right), \\
\mathbf{x}^{(\ell,3)} &= \mathbf{x}^{(\ell,2)} + \mathrm{EnvSense}\left(\mathrm{AdaLN}(\mathbf{x}^{(\ell,2)}, \mathbf{c}), \mathbf{p}^{\mathrm{inf}}, \mathbf{e}, \{\mathbf{F}^{(s)}\}_{s=1}^{S}, \boldsymbol{\eta}\right), \\
\mathbf{x}^{(\ell+1)} &= \mathbf{x}^{(\ell,3)} + \mathrm{FFN}\left(\mathrm{AdaLN}(\mathbf{x}^{(\ell,3)}, \mathbf{c})\right).
\end{aligned} \tag{26}$$

Here $\mathbf{p}^{\mathrm{inf}}$ and $\mathbf{e}$ denote the inferred positions and normalized action entropies defined above.

The temporal attention module is applied independently to each agent trajectory. For agent $i$, it treats $\{\mathbf{x}_{i,\tau}\}_{\tau=1}^{T}$ as a temporal token sequence and performs self-attention along the time dimension. Local window attention restricts each timestep $\tau$ to attend only to tokens $u \in \mathcal{W}(\tau)$ within the same window of size $W_t = 32$:

$$\mathrm{TempAttn}_{\mathrm{local}}(\mathbf{x}_{i,\tau}) = \sum_{u \in \mathcal{W}(\tau)} \mathrm{softmax}_u \left( \frac{\mathbf{q}_{i,\tau}^{\top} \mathbf{k}_{i,u}}{\sqrt{d}} \right) \mathbf{v}_{i,u}. \tag{27}$$

The module additionally uses sparse global attention over temporal anchors sampled every $S_t = 16$ timesteps, allowing each token to receive coarse long-range context without full attention over all $T$ timesteps.

The sparse social attention module operates at each timestep over the dynamic neighborhood $\mathcal{N}_k(i)$. Let $\mathbf{q}_{i,\tau}$, $\mathbf{k}_{j,\tau}$, and $\mathbf{v}_{j,\tau}$ be the query, key, and value projections. The attention weights are

$$\alpha_{ij}^{\tau} = \mathrm{softmax}_{j \in \mathcal{N}_k(i)} \left( \frac{\mathbf{q}_{i,\tau}^{\top} \mathbf{k}_{j,\tau}}{\sqrt{d}} + b_\theta \left( \mathbf{p}_{i,\tau}^{\mathrm{inf}} - \mathbf{p}_{j,\tau}^{\mathrm{inf}} \right) \right), \tag{28}$$

where $b_\theta(\cdot)$ is a learned geometric bias from relative inferred positions. The social update is

$$\mathrm{SocialAttn}_{i,\tau} = \sum_{j \in \mathcal{N}_k(i)} \alpha_{ij}^{\tau} \mathbf{v}_{j,\tau}. \tag{29}$$

The environment-sensing module samples map features around the inferred trajectory. Let $s = 1, \dots, S$ index the map-pyramid level and $p = 1, \dots, P$ index the sampling point. For each agent $i$ and timestep $\tau$, the module predicts a sampling offset $\boldsymbol{\delta}_{i,\tau}^{p}$ and an attention weight $\omega_{i,\tau}^{s,p}$ over scale-point pairs:

$$\boldsymbol{\delta}_{i,\tau}^{p} = \tanh(W_{\delta}^{p}\mathbf{x}_{i,\tau}), \qquad \omega_{i,\tau}^{s,p} = \mathrm{softmax}_{s,p}\left(W_{\omega}^{s,p}\mathbf{x}_{i,\tau}\right). \tag{30}$$

The sampling location $\mathbf{u}_{i,\tau}^{p}$ is centered at the inferred position $\mathbf{p}_{i,\tau}^{\mathrm{inf}}$, with the entropy factor reducing to one for hard one-hot inputs:

$$\mathbf{u}_{i,\tau}^{p} = \mathbf{p}_{i,\tau}^{\mathrm{inf}} + r_0(1 + 0.2e_{i,\tau})\boldsymbol{\delta}_{i,\tau}^{p}, \tag{31}$$

where $r_0$ is a learned base radius and $e_{i,\tau}$ is the normalized entropy defined in Eq. (23). With bilinear interpolation $\mathcal{B}(\mathbf{F}^{(s)}, \mathbf{u})$ on the $s$-th map feature $\mathbf{F}^{(s)}$, the environment context is

$$\mathbf{z}_{i,\tau} = \sum_{s=1}^{S}\sum_{p=1}^{P} \omega_{i,\tau}^{s,p}\,\eta_s\,\mathcal{B}\Big(\mathbf{F}^{(s)}, \mathbf{u}_{i,\tau}^{p}\Big), \tag{32}$$

where $\eta_s$ is the scale-gate weight from Eq. (17). The resulting context $\mathbf{z}_{i,\tau}$ is projected back to dimension $D$ to form the residual environment-sensing update used in Eq. (26).

**Output head.** After the final interaction block, the network normalizes the denoising tokens and maps each token to action logits through a linear output head:

$$\boldsymbol{\ell}_{i,\tau,:} = \mathrm{Linear}_{\mathrm{out}}\left(\mathrm{LN}\Big(\mathbf{x}_{i,\tau}^{(L)}\Big)\right) \in \mathbb{R}^{C}. \tag{33}$$

Collecting all agents and timesteps gives $f_\theta(\mathbf{x}_k, \mathbf{c}, k) \in \mathbb{R}^{N\times T\times C}$. The clean-action probability vector is obtained by normalizing the logits over the action dimension:

$$\hat{\mathbf{x}}_{0,i,\tau,:} = \mathrm{softmax}(\boldsymbol{\ell}_{i,\tau,:}). \tag{34}$$

These probability vectors parameterize the factorized clean-state distribution predicted by the denoiser:

$$\tilde{p}_\theta(\mathbf{x}_0 \mid \mathbf{x}_k, \mathbf{c}) = \prod_{i=1}^{N}\prod_{\tau=1}^{T} \mathrm{Cat}(\mathbf{x}_{0,i,\tau,:}; \hat{\mathbf{x}}_{0,i,\tau,:}), \tag{35}$$

which is then used to construct the D3PM reverse transition $p_\theta(\mathbf{x}_{k-1} \mid \mathbf{x}_k, \mathbf{c})$.

### D.3 Training Objective

For completeness, the generic D3PM variational term referenced in Eq. (3) decomposes as

$$\mathcal{L}_{\mathrm{vb}} = \mathbb{E}_{q(\mathbf{x}_0)}\Big[\mathrm{KL}(q(\mathbf{x}_K \mid \mathbf{x}_0)\,\|\,p(\mathbf{x}_K)) + \sum_{k=2}^{K}\mathbb{E}_{q(\mathbf{x}_k\mid\mathbf{x}_0)}\left[\mathrm{KL}(q(\mathbf{x}_{k-1} \mid \mathbf{x}_k, \mathbf{x}_0)\,\|\,p_\theta(\mathbf{x}_{k-1} \mid \mathbf{x}_k))\right] - \mathbb{E}_{q(\mathbf{x}_1\mid\mathbf{x}_0)}\left[\log p_\theta(\mathbf{x}_0 \mid \mathbf{x}_1)\right]\Big]. \tag{36}$$

In the conditional MAPF setting, let $q(\mathbf{x}_0, \mathbf{c})$ denote the expert data distribution over clean joint action tensors and MAPF conditions. Our main text uses the generative objective

$$\mathcal{L}_{\mathrm{gen}} = \mathcal{L}_{\mathrm{aux}} + \lambda_{\mathrm{KL}}\mathcal{L}_{\mathrm{KL}}, \tag{37}$$

where $\mathcal{L}_{\mathrm{aux}}$ is the auxiliary clean-state prediction term in Eq. (3), and $\mathcal{L}_{\mathrm{KL}}$ is the posterior-matching component extracted from the variational term $\mathcal{L}_{\mathrm{vb}}$. Instead of evaluating the full variational decomposition, we use a posterior-matching loss sampled over diffusion steps to align the learned reverse transition with the analytic discrete posterior:

$$\mathcal{L}_{\mathrm{KL}} = \mathbb{E}_{q(\mathbf{x}_0,\mathbf{c})}\mathbb{E}_{k\sim\mathrm{Unif}(\{1,\dots,K\})}\mathbb{E}_{q(\mathbf{x}_k\mid\mathbf{x}_0)}\left[\mathrm{KL}(q(\mathbf{x}_{k-1} \mid \mathbf{x}_k, \mathbf{x}_0)\,\|\,p_\theta(\mathbf{x}_{k-1} \mid \mathbf{x}_k, \mathbf{c}))\right]. \tag{38}$$

Under the $\mathbf{x}_0$-parameterization, $p_\theta(\mathbf{x}_{k-1} \mid \mathbf{x}_k, \mathbf{c})$ is induced by the predicted clean-state distribution $\tilde{p}_\theta(\mathbf{x}_0 \mid \mathbf{x}_k, \mathbf{c})$ through the analytic D3PM posterior, as in Eq. (2).

We further add the task-oriented auxiliary objective used in the main text:

$$\mathcal{L} = \mathcal{L}_{\text{gen}} + \mathcal{L}_{\text{task}}, \tag{39}$$

with

$$\mathcal{L}_{\text{task}} = \lambda_{\text{goal}}\mathcal{L}_{\text{goal}} + \lambda_{\text{vertex}}\mathcal{L}_{\text{vertex}} + \lambda_{\text{edge}}\mathcal{L}_{\text{edge}} + \lambda_{\text{valid}}\mathcal{L}_{\text{valid}}. \tag{40}$$

The default weights are

$$\lambda_{\text{KL}} = 0.02, \quad \lambda_{\text{goal}} = 0.4, \quad \lambda_{\text{vertex}} = 0.2, \quad \lambda_{\text{edge}} = 0.2, \quad \lambda_{\text{valid}} = 0.4.$$

The task losses are computed from the predicted clean-action distribution $\hat{\mathbf{x}}_0$ through its induced soft trajectory, rather than from samples generated by the full reverse diffusion process. Therefore, $\mathcal{L}_{\text{task}}$ serves as an auxiliary shaping signal, while $\mathcal{L}_{\text{gen}}$ remains the primary expert-imitation objective. We next define each task-oriented term used in Eq. (40).

**Goal-progress loss.** We use a BFS-based goal-progress loss. For each agent and timestep, let $d_{i,\tau}$ be the shortest-path distance from the predicted position to the goal on the obstacle map, and let $\tilde{d}_{i,\tau} = d_{i,\tau}/s$ be the normalized distance, where $s = \max(H_{\text{map}}, W_{\text{map}})$ is the map scale. For trajectory horizon $T$, the loss penalizes failures to make progress toward the goal:

$$\mathcal{L}_{\text{goal}} = \frac{1}{N(T-1)} \sum_{i=1}^{N} \sum_{\tau=0}^{T-2} \max\Big(\tilde{d}_{i,\tau+1} - \max\big(\tilde{d}_{i,\tau} - \tfrac{1}{s},\, 0\big),\, 0\Big). \tag{41}$$

This objective encourages each step to reduce the shortest-path distance by approximately one cell whenever possible.

**Vertex-conflict loss.** Let $\mathbf{p}_{i,\tau}$ denote the expected position of agent $i$ at timestep $\tau$, obtained from the predicted clean-action probabilities $\hat{\mathbf{x}}_0$. We penalize agent pairs whose expected positions lie within Manhattan distance $r_{\text{v}}$ of each other:

$$\mathcal{L}_{\text{vertex}} = \frac{1}{|\mathcal{S}_{\text{v}}|} \sum_{(i,j,\tau)\in\mathcal{S}_{\text{v}}} \exp(-\|\mathbf{p}_{i,\tau} - \mathbf{p}_{j,\tau}\|_1)\,, \tag{42}$$

where

$$\mathcal{S}_{\text{v}} = \{(i,j,\tau) : i \neq j,\ \|\mathbf{p}_{i,\tau} - \mathbf{p}_{j,\tau}\|_1 \leq r_{\text{v}}\}\,. \tag{43}$$

**Edge-conflict loss.** We penalize swap-like conflicts, where two agents move through the same edge in opposite directions between consecutive timesteps. For agents $i$ and $j$, this occurs when $\mathbf{p}_{i,\tau}$ is close to $\mathbf{p}_{j,\tau+1}$ and $\mathbf{p}_{i,\tau+1}$ is close to $\mathbf{p}_{j,\tau}$. We therefore define

$$d^{\rightarrow}_{ij,\tau} = \|\mathbf{p}_{i,\tau} - \mathbf{p}_{j,\tau+1}\|_1, \qquad d^{\leftarrow}_{ij,\tau} = \|\mathbf{p}_{i,\tau+1} - \mathbf{p}_{j,\tau}\|_1. \tag{44}$$

Let $\mathcal{S}_{\text{e}}$ contain all triples $(i,j,\tau)$ for which both distances are no larger than the edge-conflict radius $r_{\text{e}}$. The edge-conflict loss is

$$\mathcal{L}_{\text{edge}} = \frac{1}{|\mathcal{S}_{\text{e}}|} \sum_{(i,j,\tau)\in\mathcal{S}_{\text{e}}} \exp\left(-\frac{d^{\rightarrow}_{ij,\tau} + d^{\leftarrow}_{ij,\tau}}{2}\right). \tag{45}$$

**Action-validity loss.** Let $\mathcal{A}^{\text{valid}}_{i,\tau}$ be the set of actions that remain in bounds and do not enter obstacles from the current rollout position. We penalize probability mass assigned to invalid actions:

$$\mathcal{L}_{\text{valid}} = \frac{1}{NT} \sum_{i,\tau} \sum_{a\notin\mathcal{A}^{\text{valid}}_{i,\tau}} \hat{x}_{0,i,\tau,a}. \tag{46}$$

### D.4 Training Schedule

For the first 60 epochs, we optimize only $\mathcal{L}_{\text{gen}}$. We then introduce $\mathcal{L}_{\text{task}}$ over a 150-epoch warmup, multiplying its four loss weights by

$$\alpha(e) = 0.2 + 0.8 \cdot \min\Big(\frac{e}{150},\, 1\Big)\,. \tag{47}$$

Here $e$ counts epochs since the end of the generative-only stage.

### D.5 Optimization Hyperparameters

We use AdamW (Loshchilov & Hutter, 2019) with the settings in Table 10. The learning rate is constant; no exponential moving average is used.

Table 10: Main supervised training hyperparameters used by DiffLNS.

| **Hyperparameter** | **Value** |
|---|---|
| Training map size | $23 \times 23$ |
| Training map categories | maze, room, warehouse |
| Training agent counts | $N \in \{32, 64, 96\}$ |
| Dataset mixing mode | round-robin across fixed-$N$ subsets |
| Per-GPU batch size | 8 |
| Number of training epochs | 400 |
| Generative-only stage | 60 epochs |
| Task-loss warmup | 150 epochs |
| Optimizer | AdamW |
| Learning rate | $2 \times 10^{-4}$ |
| Weight decay | $10^{-4}$ |
| Gradient clipping | 1.0 |
| Precision | BF16 autocast |
| Random seed | 42 |
| Diffusion steps | 100 |
| Beta schedule | cosine |
| Number of actions | 5 |
| $\lambda_{\mathrm{KL}}$ | 0.02 |
| $\lambda_{\mathrm{goal}}$ | 0.4 |
| $\lambda_{\mathrm{vertex}}$ | 0.2 |
| $\lambda_{\mathrm{edge}}$ | 0.2 |
| $\lambda_{\mathrm{valid}}$ | 0.4 |
| Hidden dimension | 128 |
| Condition dimension | 128 |
| Attention heads | 4 |
| Interaction layers | 4 |
| Dropout | 0 |
| Temporal window $W_t$ | 32 |
| Temporal global stride $S_t$ | 16 |
| Environment sampling points per head | 8 |
| Environment feature scales | 3 |