Scalable Multi-Agent Path Finding using Collision-Aware Dynamic Alert Mask and a Hybrid Execution Strategy

Bharath Muppasani, Ritirupa Dey, Biplav Srivastava, Vignesh Narayanan

University of South Carolina, USA

bharath@email.sc.edu, deyr@email.sc.edu, vignar@sc.edu, biplav.s@sc.edu

Abstract

Multi-agent pathfinding (MAPF) remains a critical problem in robotics and autonomous systems, where agents must navigate shared spaces efficiently while avoiding conflicts. Traditional centralized algorithms that have global information, such as Conflict-Based Search (CBS), provide high-quality solutions but become computationally expensive in large-scale scenarios due to the combinatorial explosion of conflicts that need resolution. Conversely, distributed approaches that have local information, particularly learning-based methods, offer better scalability by operating with relaxed information availability, yet often at the cost of solution quality. To address these limitations, we propose a hybrid framework that combines decentralized path planning with a lightweight centralized coordinator. Our framework leverages reinforcement learning (RL) for decentralized planning, enabling agents to adapt their planning based on minimal, targeted alerts–such as static conflict-cell flags or brief conflict tracks–that are dynamically shared information from the central coordinator for effective conflict resolution. We empirically study the effect of the information available to an agent on its planning performance. Our approach reduces the inter-agent information sharing compared to fully centralized and distributed methods, while still consistently finding feasible, collision-free solutions–even in large-scale scenarios having higher agent counts.

1 Introduction

Multi-Agent Path Finding (MAPF) addresses the fundamental challenge of computing collision-free trajectories for multiple agents navigating a shared environment. Its effective solving is crucial for deploying a wide array of multi-agent systems, from automated warehouses and robotic swarms to autonomous vehicle coordination (Sharon et al. 2015). Despite its practical significance, MAPF is computationally demanding, classified as NP-hard in its general form, which can render traditional centralized search methods intractable as the number of agents or the complexity of the environment increases (Ren et al. 2025; Sartoretti et al. 2019).

Extensive research in MAPF has explored various coordination paradigms, each with different implications for information availability and system performance. Centralized approaches, such as Conflict-Based Search (CBS) (Sharon et al. 2015) and its efficiency-focused variant ICBS (Boyarski et al. 2015), typically assume full observability of the environment and agent states, helping them generate optimal plans. However, as the number of agents or the size of the environment increases, joint planning and conflict resolution become computationally expensive. Additionally, requiring agents to share full information raises privacy concerns, limiting applicability in confidential settings motivating strategies that reduce reliance on global information. Early decoupled methods like M^* (Wagner and Choset 2011) begin with individual plans and only coordinate agents in conflict, offering scalability but often sacrificing optimality or completeness. More recently, Multi-Agent Reinforcement Learning (MARL) has emerged to address coordination and partial observability in dynamic settings. For example, PRIMAL (Sartoretti et al. 2019) trains agents to plan using partial views, learning implicit coordination, while methods like FOLLOWER (Skrynnik et al. 2024) reduce explicit communication by relying on global heuristic maps. These decentralized methods improve scalability but may reduce

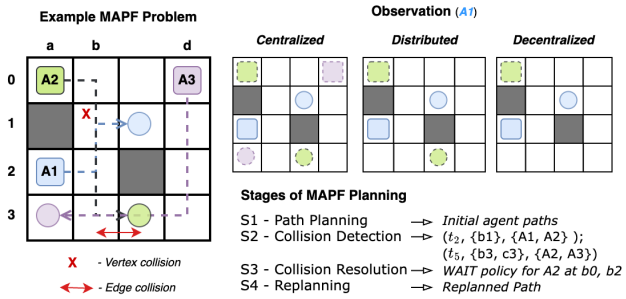


Figure 1: An example MAPF problem and our four-stage planning pipeline. **Left:** Three agents (A1, A2, A3) navigate a grid with static obstacles (dark gray). The diagram illustrates a future vertex collision (red X), where two agents would occupy the same cell, and an edge collision (red arrows), where agents would swap adjacent cells. **Top Right:** The varying levels of information available to Agent A1 under centralized (all agent positions and goals), distributed (nearby agent positions and goals), and decentralized (only nearby agent positions) paradigms. **Bottom Right:** The four stages of our framework, from initial Path Planning (S1) to Collision Detection (S2), Resolution (S3), and Replanning (S4).

solution quality and leave conflicts unresolved due to limited information.

To address these limitations, we propose a novel hybrid MAPF framework that combines decentralized planning with a lightweight centralized coordinator, focusing on minimizing inter-agent information sharing while maintaining solution feasibility. Our findings show that minimal, targeted alerts are sufficient, reducing the total information load by an estimated **∼93%** compared to a continuous-observation distributed paradigm (see Appendix A.1). In our approach, agents primarily rely on decentralized, reinforcement learning-based neural network planners that operate on local observations (e.g., position coordinates), eliminating the need for global information as well as local egocentric maps. A central coordinator oversees agent trajectories, intervening selectively by dynamically sharing targeted information to prompt localized re-planning when conflicts are anticipated. This work's contributions are centered on this selective coordination strategy. We introduce **(1) a novel hybrid framework** with this on-demand alert mechanism and provide **(2) a comprehensive empirical evaluation** demonstrating that our planning policy generalizes robustly from a simple training regimen to larger, more complex environments. Our evaluation is guided by two key research questions: ***(RQ1)*** Given the observation constraints of a decentralized setup, can an effective MAPF algorithm be created with one agent knowing nothing about other agents? If not, what information must it need at a minimum? and ***(RQ2)*** How does the proposed hybrid method compare to leading alternative search- and learning-based approaches in terms of performance, solution quality, and scalability?

2 Background and Literature Review

2.1 Multi-Agent Path Finding

Let $G = (V, E)$ be an undirected graph, where V is the set of vertices (grid cells) and $E \subseteq V \times V$ is the set of edges connecting adjacent cells. A team of n agents $A = \{a_1, \ldots, a_n\}$ must move from start vertices $s_i \in V$ to goal vertices $g_i \in V$, where $(s_i, g_i) \neq (s_j, g_j), \forall i \neq j : i, j \in \{1, ..., n\}$. Time is discretized into steps $t = 0, 1, 2, \ldots$, and at each step, an agent may either move along an edge or wait. A path for agent a_i is a sequence $\pi_i = (v_0^i, v_1^i, \ldots, v_{T_i}^i)$ with $v_0^i = s_i$ and $v_{T_i}^i = g_i$. A solution to MAPF is $\Pi = \{\pi_1, \ldots, \pi_n\}$, and it is collision-free if for all $i \neq j$ and all t, $v_t^i \neq v_t^j$ (vertex collision free) and $(v_t^i, v_{t+1}^i) \neq (v_{t+1}^j, v_t^j)$ (edge collision free). The primary goal in standard MAPF is typically to find a path set Π that is collision-free (Sharon et al. 2015). Common efficiency objectives include minimizing the makespan, $\max_i T_i$, minimizing the sum of individual completion times (sum-of-costs), $\sum_i T_i$, or minimizing the number of collisions.

An important consideration in defining a MAPF problem instance arises from agents potentially reaching their targets at different time steps, thus requiring a clear definition of how an agent behaves after it has reached its target g_i at time T_i, but while other agents may still be en route. Two common settings are used to address this scenario (Stern et al. 2019a). The first setting, typically referred to as *stay at target*, considers that an agent, upon reaching its goal, remains at that vertex until all other agents in the team have also reached their respective targets. This waiting agent continues to occupy its target vertex, potentially causing conflicts if another agent's plan requires traversing or occupying that vertex at any time $t \geq T_i$ up to the completion time of the entire solution. The second setting, *disappear at target*, assumes an agent is effectively removed from the environment immediately upon reaching its target. Consequently, its path concludes at time T_i, and it poses no further collision risk nor occupies any vertex for $t > T_i$. In our work, we adopt the second setting, *disappear at target*. This setting is particularly relevant for applications like drone delivery, where an agent's task is considered complete upon arrival at a target location.

2.2 Coordination Paradigms

As formalized by Sharon et al. (Sharon et al. 2015), solution approaches to MAPF problems can be categorized based on their coordination strategy. We extend this by focusing on three key elements: state observability (global vs. local), communication (allowed vs. none), and control (centralized vs. decentralized). In *centralized* paradigms, a global planner with full observability of the entire state controls all agents, and communication is implicit through this central controller. By contrast, other approaches grant agents local control over their actions. Within this category, we draw a key distinction based on communication: In *distributed* approaches, agents that plan their own paths are allowed to explicitly exchange information, such as local observations, intended paths, or goals, with other agents to achieve cooperative behavior. In *decentralized* MAPF, as defined in this work, inter-agent communication during execution is eliminated. Agents plan and act entirely independently, relying only on their own local information. Finally, *hybrid* frameworks, like the one we propose, combine these elements. They typically employ a central coordination mechanism that has full observability to detect conflicts and selectively intervene, while agents otherwise plan in a decentralized manner. In our work, we adopt a hybrid framework with a customized information sharing mechanism as described in Section 3.

2.3 Literature Review

Search-based Methods: Research on MAPF has evolved from centralized to more scalable methods. Early advancements include M^*, which dynamically couples agent searches upon conflict for optimality and completeness (Wagner and Choset 2011), and Increasing Cost Tree Search (ICTS), which employs cost allocation and then checks for conflict-free solutions from sets of single-agent paths, yielding significant speedups over joint-space search (Sharon et al. 2013). A pivotal development was CBS, where agents independently plan paths using A^*, and a centralized detector resolves conflicts by branching with new constraints for involved agents, ensuring optimality (Sharon et al. 2015). To enhance scalability while preserving optimality, variants such as ICBS integrate cardinality heuristics and meta-agent

merging to accelerate convergence (Boyarski et al. 2015). For very large instances, suboptimal repair strategies like Large Neighborhood Search (LNS) generate initial solutions (often via prioritized planning) and iteratively repair colliding agent subsets under a centralized control loop (Li et al. 2022). However, these methods face scalability challenges as the number of agents increases and often depend on extensive global information for coordination.

Learning-based Methods: In parallel, learning-based approaches address partial observability and communication constraints. PRIMAL and PRIMAL2 train decentralized policies via a combination of imitation learning from an expert centralized planner and reinforcement learning (PRIMAL2 enhancing local observations for better coordination), effectively implementing a centralized training and decentralized execution (CTDE) paradigm (Sartoretti et al. 2019; Damani et al. 2021b). SCRIMP introduces a transformer-based communication module enabling agents with limited fields of view to coordinate effectively, achieving decentralized coordination without a central execution-time coordinator (Wang et al. 2023). To function in real-world settings, each agent must carry onboard sensing, e.g., RGB-D cameras or LiDAR scanners, and implement an inter-agent communication protocol for goal disclosure, which introduces additional computational overhead and raises privacy concerns. While offering adaptability and decentralized execution, learning-based methods can require significant training data, may struggle with generalization, and often provide weaker guarantees on solution quality or completeness when relying purely on local information.

Hybrid Methods: Hybrid planning-learning paradigms seek the strengths of both worlds. Authors in (Skrynnik et al. 2024) introduce FOLLOWER, which involves each agent using a congestion-aware A^* planner to determine its subgoals, then executing a fully decentralized RL policy for lifelong replanning. More recently, hybrid frameworks like LNS2+RL, introduced by Li et al., have emerged, augmenting LNS by employing MARL for early, localized conflict resolution, then relying on efficient search-based planning for final refinement (Wang et al. 2025). These existing hybrid approaches, while innovative, can involve complex integrations and may still face trade-offs between optimality, scalability, or specific information dependencies (like global congestion maps (Skrynnik et al. 2024)).

To enable a structured comparison of diverse MAPF techniques (search-based, learning-based, and hybrid), we adopt the four-stage MAPF pipeline: S1 (Agent Planning), S2 (Collision Detection), S3 (Collision Avoidance Policy), and S4 (Agent Replanning). Within this structure, CBS uses decentralized A* search for S1/S4 and centralized high level solver for collision resolution in S2/S3 for optimality (Sharon et al. 2015). LNS employs a centralized loop for all four stages, managing path generation, detection, repair, and subset replanning, thereby trading optimality for scalability (Li et al. 2022). Learning-based planners like PRIMAL and SCRIMP typically use local observations for decentralized S1-S4 execution, coordinating without a central controller (Sartoretti et al. 2019; Wang et al. 2023); however, practical deployments can require sensors (eg., camera, LiDAR etc.) and communication, adding computational overhead and increase sensing cost. Hybrid methods such as FOLLOWER use global congestion maps with A^* search for S1, then decentralized RL policy for S2-S4, often without further communication (Skrynnik et al. 2024). LNS2+RL applies centralized LNS for S1-S3, while S4 uses a hybrid policy (MARL or prioritized planning) for subsets, balancing key trade-offs (Wang et al. 2025). Refer to Table 2 in the Appendix Section A.2 for a detailed categorization of these methodologies.

Together, these works highlight key limitations of existing MAPF paradigms: centralized methods face scalability and privacy concerns; purely learning-based approaches compromise solution quality; and current hybrid solutions often still depend on significant centralized communication or coordination overhead with higher sensing cost. The hybrid framework introduced in Section 3, which is the focus of our work, is specifically designed to mitigate these limitations. It strategically reduces inter-agent information sharing to maintain solution feasibility, thereby addressing the aforementioned information management challenges while also aiming to preclude the additional sensing costs and computational overhead common in learning-based methods.

3 Methodology

3.1 Approach Summary

We propose a hybrid coordination framework that leverages decentralized RL-based path planning together with centralized collision detection and control. While conceptually similar to the high-level loop of Conflict-Based Search (CBS), our framework differs significantly in its resolution step: it does not build a constraint tree or perform backtracking, instead issuing targeted alerts that prompt heuristic-based replanning from the local RL agent. The proposed method's four-stage MAPF pipeline is detailed below.

S1: Decentralized Path Planning Each agent a_k generates an independent trajectory

$$\rho_k = \pi_\theta(s_k, g_k) = (v_0^k, \ldots, v_{\tau_k}^k), \tag{1}$$

where $v_0^k = s_k$, $v_{\tau_k}^k = g_k$, and π_θ is a parameterized RL policy trained to minimize path length and local collision risk over a planning horizon H. No information about other agents is exchanged during this stage. Let τ_i denote the set of makespan of each agent and $\tau = \{\tau_1, \ldots, \tau_n\}$ be the set of all such makespans.

S2–S3: Centralized Collision Detection and Control A central module takes as input the independent trajectories of all the agents $\rho = \{\rho_1, \ldots, \rho_n\}$ along with the makespan set τ, and identifies all vertex and edge conflicts, using

$$\begin{aligned} C(\rho, \tau) = \{(t_j, \Delta_c, A_c) \mid (v_{t_j}^k = v_{t_j}^l := \Delta_c) \vee \\ ((v_{t_j}^k, v_{t_{j+1}}^k) = (v_{t_j+1}^l, v_{t_j}^l) := \Delta_c)\} \end{aligned} \tag{2}$$

$$\begin{aligned} A_c = \{\{a_k,\, a_l\} : \rho_k|\{t_j\} = \rho_l|\{t_j\} = \Delta_c \vee \rho_k| \\ \{(t_j, t_{j+1})\} = \rho_l|\{(t_j, t_{j+1})\} = \Delta_c, \forall k \neq l\}, \end{aligned} \tag{3}$$

where $\rho_k|\{t_j\}$ and $\rho_k|\{(t_j, t_{j+1})\}$ denotes the position of the k^{th} agent at step t_j and steps (t_j, t_{j+1}), respectively.

For each conflict $c = (t_j, \Delta_c, A_c) \in C(\rho, \tau)$, which occurs at timestep t_j, the controller issues an alert $\mathcal{A}$ defined as

$$\mathcal{A}(c) = \mu_{c_k} = (a_{c_k}, t_{j-r}, \Delta_c), r \geq j, \tag{4}$$

where policy μ_{c_k} is used to select an agent $a_{c_k} \in A_c$. Example policies for this agent selection is described in Section 3.2. Once an alert is issued to an agent, the selected agent is prompted to perform a constrained replanning of its trajectory for a rollout window $\{t^{c_k}_{j-r}, \ldots, t^{c_k}_j, \ldots, \tau_{c_k}\}$, where r is the rewind window by avoiding the collision set Δ_c.

S4: Decentralized Replanning Upon receiving a collision alert, agent a_{c_k} truncates its path at $v^{c_k}_{t_{j-r}}$, denoted as

$$\rho_{c_k|t_{j-r-1}} = (v^{c_k}_0, \ldots, v^{c_k}_{t_{j-r-1}}). \tag{5}$$

The agent then generates a new path segment ρ'_{c_k} from $v^{c_k}_{t_{j-r}}$ to its goal g_{c_k} by applying its RL policy π_θ. This replanning incorporates new constraints derived from the alert. For static obstacle avoidance *(S4.1)*, the policy is typically derived by incorporating a fixed constraint, wherein the agent is supposed to avoid forbidden cells, as captured in Eq. 6, where Δ_c represents the set of static cells identified in the alert, and a segment of the new path is generated as:

$$\rho'_{c_k} = \pi_\theta(v^{c_k}_{t_{j-r}}, g_{c_k} \mid v^{c_k}_{t_i} \notin \Delta_c, i \geq j - r). \tag{6}$$

Alternatively, for dynamic obstacle avoidance *(S4.2)*, the replanning must account for the predicted movements of other agents involved in the collision. In this case, the RL policy is conditioned on the sub-paths of the conflicting agents

$$\rho'_{c_k} = \pi_\theta(v^{c_k}_{t_{j-r}}, g_{c_k} \mid v^{c_k}_t \neq v^{c_l}_t, v^{c_l}_t \in \rho_{c_l}, \\ \forall l \neq k, a_{c_l} \in A_c, \forall t \in [t_{j-r}, t_{j+r}]). \tag{7}$$

The information guiding the replanning–whether it constitutes minimal details such as static obstacle constraints (Eq. 6), or more detailed sub-path information about colliding agents' paths treated as dynamic obstacles (Eq. 7)–is integrated into the RL agent's decision-making process (by modifying its state representation, more details in the Section 3.2) to guide it towards a conflict-free solution. The agent's new complete trajectory $\rho^{new}_{c_k}$ is formed by concatenating the initial segment with the newly planned one, given by

$$\rho^{new}_{c_k} = \rho_{c_k|t_{j-r}} \| \rho'_{c_k}. \tag{8}$$

This updated trajectory is submitted to the central controller. This iterative cycle of detection (S2), control (S3), and selective replanning (S4) continues until a globally conflict-free solution is achieved ($C(\rho, \tau) = \emptyset$).

Our hybrid MAPF framework manages a precise flow of information, essential to support our claims of reduced information sharing and adaptive planning. The process begins with fully decentralized agent planning (S1), where each RL-driven agent uses only its local information before submitting its intended path ρ_k to a central coordinator. This coordinator, leveraging its global observation of all submitted paths, performs collision detection (S2) to identify the set of all potential conflicts $C(\rho, \tau)$. Following detection, the central control module (S3) issues targeted alerts; it selects a specific agent a_{c_k} and determines the rewind point t_{j-r} from which the agent must replan, effectively issuing an alert $\mathcal{A}(c)$. S3 also governs the level of information to be shared for the subsequent decentralized replanning (S4). Initially, the alerted agent a_{c_k} attempts to resolve the conflict using minimal information, typically by treating its own conflicting path segment Δ_c as a static obstacle (S4.1). If this localized, low-information approach proves insufficient, S3 then shares additional information with the agent–specifically, a subset of the path of other directly conflicting agents–for a more informed replan (S4.2). This tiered strategy—from purely local information in S1, to targeted alerts from S3, and then to progressively detailed yet still localized conflict information for S4—ensures that inter-agent information sharing is demand-driven and sparse. Agents communicate their revised plans $\rho^{new}_{c_k}$ back to the coordinator, and this iterative cycle of detection and selective, adaptive replanning continues until a globally conflict-free solution is achieved ($C(\rho, \tau) = \emptyset$). This methodology substantiates our framework's ability to ensure feasibility while operating with significantly reduced information exchange.

3.2 Policy Representation

(S1 & S4) Decentralized Path Planning: In our proposed framework, we address the challenges of decentralized multi-agent pathfinding in dynamic environments by integrating collision awareness directly into the agent's observation. Our approach leverages an RL-based decentralized planner that receives a multi-channel grid observation.

Observation Space: Each agent's observation is a tensor $\mathbf{s} \in \mathbb{R}^{H \times W \times 4}$, where H and W are the height and width of the grid. The four channels correspond to: (a) *ObstacleMap* - A binary map (1 if a static obstacle is present, 0 otherwise). (b) *AgentMap* - A binary map marking the agent's current position. (c) *GoalMap* - A binary map indicating the goal position. (d) *AlertMask* - A dynamic collision alert map (initially all zeros, updated during execution). This alert mask is updated online by a centralized collision detection module that simulates timely collision alerts. In addition, the observation includes low-dimensional features comprising a unit vector pointing from the agent's current position to the goal and the Euclidean distance.

Action Space: The agent operates in a discrete action space, $\mathcal{A} = \{0, 1, 2, 3, 4\}$, corresponding to movements in the four cardinal directions (e.g., Up, Down, Left, Right, WAIT).

Reward Structure Our reward function, similar to PRIMAL (Sartoretti et al. 2019), promotes efficient, collision-free navigation and discourages unproductive behaviors. Agents receive +20 for reaching the goal. Penalties include: -3 for collisions with obstacles, -2 for timeout, -0.02 per time step taken (to encourage efficiency), and -0.1 for a 'WAIT' action, a penalty that encourages path progression over stationary waiting. An additional -0.05 penalty is applied if the agent is near a dynamic obstacle.

(S2) Rule-Based Collision Detection: The collision detection module (S2) functions as a deterministic, rule-based system. It takes the set of all current agent trajectories $\rho = \{\rho_1, \ldots, \rho_n\}$ and their makespans τ as input. For each timestep $t = \{0, 1, ..., T_M\}$ upto the maximum makespan (T_M), the module systematically scans for two primary types of conflicts: vertex conflicts (where $v_t^k = v_t^\ell$ for $k \neq \ell$) and edge conflicts (where $(v_t^k, v_{t+1}^k) = (v_{t+1}^\ell, v_t^\ell)$ for $k \neq \ell$). Upon detecting a conflict, characterized as $c = (t, v, A_c)$ where A_c is the set of agents involved, S2 reports this conflict c to the S3 control module for resolution. The detection process for all conflicts is computationally efficient, with a time complexity of $O\big(\sum_k |\rho_k|\big)$ per cycle, linear in the sum of all agent path lengths.

(S3) Heuristic-Based Collision Avoidance Control: Upon notification of a conflict $c = (t, v, A_c)$ from S2, the S3 control module formulates and issues an alert $\mathcal{A}(c)$ to a selected agent. This involves choosing an agent $a_{c_k} \in A_c$ to replan, determining its replan start point t_{j-r} by selecting an appropriate rewind value r, and specifying the replanning approach. We consider three agent selection policies (from A_c): (i) *Random* choice (i.e., $a_{c_k} \sim \text{UniformRandom}(A_c)$), (ii) selecting the agent *Farthest* from its goal (g_a) based on Manhattan distance $d_{\text{Manh}}(v^a_{t_{j-r}}, g_a)$, or (iii) identifying the agent with the *Fewest Future Collisions (FFC)*, i.e., $a_{c_k} = \arg\min_{a \in A_c} \big|\{\tilde{c} \mid \tilde{c} \in C(\rho, \tau),\, a \in A_{\tilde{c}}\}\big|$. In our work, we present the results with ***FFC*** agent selection policy. We present the comparative results considering other policies in the Appendix Section A.4. S3 then dictates the replanning formulation, choosing either *Static Obstacle Replanning* (where a_{c_k} avoids its own problematic vertices v_c) or *Dynamic Obstacle Replanning* (where a_{c_k} avoids specified sub-path trajectories $v_t^{c_l} \in \rho_{c_l}, \forall l \neq k, a_{c_l} \in A_c, \forall t \in \{t_{j-r}, \ldots, t_{j+r}\}$ of other conflicting agents $a_\ell \in A_c$). Our framework initially employs *Static Obstacle Replanning* for resolving a conflict; if this approach proves insufficient to resolve the conflict, *Dynamic Obstacle Replanning* is utilized. Abalation study considering individual replanning strategies and their implication on our framework's performance is presented in the Appendix Table 5.

4 Experimental Setup

In this section we first describe how we train the per-agent navigation policy used in stage S1, then detail the maps, problem instances, and competing planners used to evaluate stages S2–S4, and define the performance metrics used to evaluate.

Training Procedure We train a parametrized RL model $Q_\theta(s, a)$ using the DDQN algorithm(Van Hasselt, Guez, and Silver 2016), incorporating prioritized experience replay (PER) and ε-greedy exploration, to enable an agent to navigate to a specified goal in the presence of both static and dynamic obstacles. Each dynamic obstacle possesses a hidden goal and follows precomputed trajectories, executing only valid moves; this allows for the simulation of online collision alerts during inference from the S3 stage. Training is performed for 30 000 episodes on an 11×11 maze grid, with each episode capped at a maximum of $T_{\max} = 50$ steps. The curriculum for environmental complexity is scheduled as follows: during the first 500 episodes, static-obstacle density $\rho_s = 0.10$ and dynamic-obstacle count $n_d = 0$; from episode 500 to 2999, $\rho_s = 0.10$ and $n_d = 1$; from episode 3000 to 5999, $\rho_s = 0.20$ and $n_d = 2$; and for episode 6000 onward, $\rho_s = 0.30$ and $n_d = 4$.

Input observations are normalized to zero mean and unit variance before being fed to the network. Transitions (s_t, a_t, r_t, s_{t+1}) are stored in a prioritized experience replay buffer of size 10^6. At each learning step, a mini-batch of 128 transitions is sampled. The agent selects actions using an ε-greedy policy, where ε decays from $\varepsilon_0 = 1.0$ to 0.01 over the course of training according to $\varepsilon_{t+1} = \max(0.01, 0.999\,\varepsilon_t)$. Crucially, during both exploration and exploitation (e.g., when calculating the target Q-value), only actions a_t deemed valid by an action mask $m_t \in \{0, 1\}^{|\mathcal{A}|}$ are considered for efficient exploration (Damani et al. 2021b).

The Q-network parameters θ are updated via gradient descent using the Adam optimizer with a learning rate $\alpha = 3 \times 10^{-4}$. The discount factor is $\gamma = 0.97$. A separate target network, with parameters θ^-, provides stable targets and is updated by copying θ every 300 steps. Updates minimize the prioritized, importance-weighted Bellman error. The target value y_t and TD-error δ_t are computed as: $y_t = r_t + \gamma\, Q_{\theta^-}\big(s_{t+1}, \arg\max_{a'} Q_\theta(s_{t+1}, a') \text{ s.t. } m_t(a') = 1\big)$, $\delta_t = y_t - Q_\theta(s_t, a_t)$, The loss function is then calculated as $L(\theta) = \mathbb{E}_{i \sim p(i)}\big[w_i\, \delta_i^2\big]$, where the sampling probability $p(i) \propto |\delta_i|^\alpha$ (α is a hyperparameter for PER) and w_i are importance-sampling weights that correct for the bias introduced by prioritized sampling.

In addition to the DDQN model, a Proximal Policy Optimization (PPO) based model was also trained; details regarding its architecture (including the one used for DDQN model) and training procedure, along with comparative performance, are provided in the Appendix Section A.3. The DDQN model was selected due to its superior performance over the PPO model - results are presented in Table 3.

Evaluation Scenarios and Baselines We evaluate our framework's performance and scalability across three distinct grid configurations, representing two primary map types: maze and warehouse. All evaluations use ten randomly generated problem instances per configuration. For the maze map type, we test on an 11×11 grid with approximately 45% static obstacles (5 to 20 agents) and a larger 21×21 grid with approximately 35% static obstacles (32, 64, and 96 agents). For the warehouse map type, we utilize a 25×25 grid layout with approximately 24% static obstacles as shelves (32, 64, and 96 agents). Our proposed hybrid framework is evaluated through two variants, Alert-BFS and Alert-A*, which are named based on their RL policy rollout strategy (*Breadth-First Search* or *weighted A-star search*, respectively). The impact of our specific replanning strategies is further analyzed in an ablation study in Appendix Table 5. These are compared against search-based planners, Conflict-Based Search (CBS) (Sharon et al. 2015) and Improved CBS (ICBS) (Boyarski et al. 2015), and lead-

ing learning-based online planners PRIMAL (Damani et al. 2021a) and SCRIMP (Wang et al. 2023) with the MAPF formulation of *disappear at goal*. While our framework variants operate within our proposed four-stage pipeline (S1–S4), the baseline methods are executed as standalone algorithms. Per-instance time limits are 50 seconds for 11×11 mazes, 30 minutes for 21×21 mazes, and 1 hour for 25×25 warehouse scenarios. For online planners, the step limits are 256 for the 11×11 maze, and 512 for both the 21×21 maze and the 25×25 warehouse, as detailed in Table 1. Additional results on various other map configurations are provided in the Appendix Section A.4.

Performance Criteria We evaluate our MAPF framework using standard metrics. *Success Rate (SR)* is the proportion of instances solved within defined time limits. For successful instances, *Makespan (MS)* measures the time until the last agent reaches its goal. The number of *Collisions (CO)* measures the total count of path disagreements detected and successfully resolved by the framework to produce the final collision-free solution. A non-zero CO in a successful run reflects the planner's total conflict resolution workload. Finally, total *Time (T)* measures the computational cost until a solution is found or the time limit is exceeded. Learning-based methods are evaluated by the average number of steps taken in the environment.

5 Results and Discussion

Here we present an empirical study, evaluating our central hypothesis: *a MAPF framework with strategically reduced information sharing can achieve robust performance.* We address our **RQs** concerning minimal information needs and comparative performance against approaches with different information paradigms using the experimental results presented in Table 1.

(RQ1): *Given the observation constraints of a decentralized setup, can an effective MAPF algorithm be created with one agent knowing nothing about other agents? If not, what information must it need at a minimum?* ***Ans.*** No—if agents have zero knowledge of one another, MAPF fails, losing completeness guarantees; but solutions are possible with targeted, minimal information sharing. Table 5 in the appendix presents comparative results for various inter-agent information sharing settings.

Now, we turn our attention to the strategically limited minimal information sharing case. In our framework, strategically informed alerts convey either (S4.1) static constraints (the conflicting cell) or (S4.2) slightly richer, yet still localized, dynamic constraints (the sub-path of only the directly conflicting agent(s) for a short horizon). Table 1 shows performance comparison of considered methodologies across 11x11 maze (a), 21x21 maze (b), and 25×25 (c) warehouse maps, respectively. As shown in Table 1, the performance of our framework, where agents initially plan with only local information and receive minimal, targeted alerts from a central coordinator upon conflict, achieves high Success Rates (SR). Specifically, on the 11×11 and 21×21 maze grids, SRs are high (90%-100%). On the more structured 25×25 warehouse map (Table 1c), Alert-BFS and Alert-A* also achieve 100% SR for 32 and 64 agents, with Alert-A* maintaining a 60% SR in the dense agent scenario (96 agents), demonstrating effectiveness across different environment types with this minimal information sharing strategy. This success with minimal inter-agent information contrasts sharply with alternatives. Centralized planners (*CBS*, *ICBS*) demand *complete global knowledge* of all agents' paths, a requirement that, while enabling optimality, leads to their low SR due to the combinatorial explosion in conflict resolution within practical time limits. In contrast, distributed learning frameworks (*PRIMAL* & *SCRIMP*) often operate under the assumption that each planning agent possesses *full observation, of other agents and their goals, in a local field of view*. While this localized observation can facilitate decentralized decision-making, it implies a continuous and potentially substantial information gathering (e.g., via onboard cameras or LiDAR sensors) and processing requirement for each agent.

The success of our method across maze and warehouse environments underscores that this on-demand information is sufficient for high SR, thereby avoiding the continuous, dense local information exchange. Our quantitative analysis, detailed in the Appendix A.1, shows that this on-demand alert mechanism reduces the total information load by $\sim$**93%** compared to a continuous-observation distributed paradigm in this scenario.

(RQ2): *How does the proposed hybrid method compare to leading alternative search- and learning-based approaches in terms of performance, solution quality, and scalability?* ***Ans.*** Our hybrid framework exhibits a compelling balance of performance and scalability with distinct trade-offs against established search-based and learning-based methods, as detailed in Table 1.

Our Python-based framework demonstrates higher SR and scalability compared to search-based optimal solvers. *CBS/ICBS* (C++ based) shows constantly lower SR with increasing environment complexity, failing to solve any instances on the 21×21 maze (for 32+ agents) and the 25×25 warehouse map. While *CBS/ICBS* achieves optimal MS when they solve (Table 1a), their reported Collision (CO) counts (Table 1a) for such cases are very high; our methods report significantly fewer collisions when they succeed on these smaller maps.

PRIMAL exhibits low SR across all tested configurations, often hitting step limits without success. *SCRIMP* is a strong learning-based performer with fast inference times and good MS. On the 11×11 and 21×21 mazes, its SR is comparable or better than our methods in some low-density cases, but can decline in denser scenarios (21×21, 64 agents: *SCRIMP* 20% SR vs. *Alert-BFS* 90% SR). On the 25×25 warehouse map, *SCRIMP* maintains 100% SR for 32 and 64 agents, and achieves 90% SR for 96 agents, outperforming Alert-BFS (10% SR) and Alert-A* (60% SR) at this highest agent count in terms of success. However, SCRIMP also demonstrates significantly lower CO values than our methods across all successful warehouse instances attributing to its efficient multi-agent training. A critical factor for online planners is the offline training investment. For instance, *PRIMAL*'s extensive training on varied map sizes in-

Table 1: Comparison of performance metrics for different methods on an 11×11 grid (5–20 agents, $\sim$45% obstacles), a 21×21 grid (32, 64, and 96 agents, $\sim$35% obstacles), and a 25×25 warehouse grid (32, 64, 96 agents, $\sim$24% obstacles). Results are averaged over 10 problem instances per configuration. Key metrics include Success Rate (SR), Makespan (MS), Collisions (CO), and Time (T), with Time averaged over all trials. For learning-based methods, the average steps taken while solving the problem instances are presented in brackets along with average wall-clock time taken.

(a) 11x11 maze with 50 sec. time limit per problem instance (256 steps for learning-based methods)

	SR (↑)			MS (↓)			CO (↓)			T (s) (↓)		
Methods	5-10	11-15	16-20	5-10	11-15	16-20	5-10	11-15	16-20	5-10	11-15	16-20
Alert-BFS	**100.00%**	**100.00%**	**98.00%**	17.8	23.6	27.4	9	53	191	1.1	4.2	11.8
*Alert-A**	**100.00%**	96.00%	82.00%	16.1	20.7	24.0	10	59	173	3.3	12.0	25.8
CBS	85.00%	34.00%	2.00%	**13.8**	**14.3**	17.0	559	8785	3895	8.3	30.7	35.3
ICBS	85.00%	26.00%	10.00%	14.0	14.8	**12.8**	885	3486	1466	5.9	25.1	28.6
PRIMAL	61.67%	48.00%	28.00%	71.7	89.9	104.8	-	-	-	1.4 (142)	3.4 (176)	6.1 (214)
SCRIMP	95.00%	70.00%	60.00%	19.3	28.6	38.8	**1**	**2**	**8**	**0.4 (31)**	**1.6 (97)**	**3.9 (124)**

(b) 21x21 maze with 30 min. time limit per problem instance (512 steps for learning-based methods)

	SR (↑)			MS (↓)			CO (↓)			T (min) (↓)		
Methods	32	64	96	32	64	96	32	64	96	32	64	96
Alert-BFS	**100.00%**	**90.00%**	0.00%	86.6	131.4	-	528	18909	-	2.7	23.9	30.0
*Alert-A**	**100.00%**	80.00%	0.00%	**41.4**	**72.0**	-	457	**11379**	-	3.3	23.5	30.0
CBS	0.00%	0.00%	0.00%	-	-	-	-	-	-	30.0	30.0	30.0
ICBS	0.00%	0.00%	0.00%	-	-	-	-	-	-	30.0	30.0	30.0
PRIMAL	0.00%	0.00%	0.00%	-	-	-	-	-	-	0.8 (512)	2.1 (512)	1.8 (512)
SCRIMP	**100.00%**	20.00%	**10.00%**	65.9	135.0	**94.0**	**15**	**220**	**357**	**0.1 (66)**	**1.0 (437)**	**2.4 (470)**

(c) 25x25 warehouse with 1 hour time limit per problem instance (512 steps for learning-based methods)

	SR (↑)			MS (↓)			CO (↓)			T (min) (↓)		
Methods	32	64	96	32	64	96	32	64	96	32	64	96
Alert-BFS	**100.00%**	**100.00%**	10.00%	94.3	125.9	**96.0**	237	6379	26553	5.2	28.0	59.5
*Alert-A**	**100.00%**	**100.00%**	60.00%	**40.5**	**43.7**	**50.8**	211	3129	16277	4.7	19.2	51.8
CBS	0.00%	0.00%	0.00%	-	-	-	-	-	-	60.0	60.0	60.0
ICBS	10.00%	0.00%	0.00%	38.0	-	-	123107	-	-	57.9	60.0	60.0
PRIMAL	0.00%	0.00%	0.00%	-	-	-	-	-	-	1.0 (512)	3.1 (512)	6.6 (512)
SCRIMP	**100.00%**	**100.00%**	**90.00%**	60.2	49.0	101.7	**2**	**14**	**161**	**0.1 (60)**	**0.3 (49)**	**1.1 (143)**

volved approximately 20 days on significant computational resources (Damani et al. 2021a). *SCRIMP*, with its sophisticated transformer-based architecture and PPO training, also implies a substantial training regimen across diverse configurations to achieve its reported performance (Wang et al. 2023). In contrast, our RL planning model, trained only on 11×11 maze grids, was developed with a comparatively modest regimen: training was conducted on a Mac Mini equipped with an Apple M4 Pro chip (14-core CPU, 20-core GPU, 16-core Neural Engine) and 48GB unified memory, running for approximately 6 to 8 hours. Its subsequent effective performance when tested on different map types (map and warehouse) and larger grid sizes (up to 25×25) indicates strong generalization from this simpler training setup.

6 Conclusions

Our hybrid framework combines decentralized RL-based planning with only minimal, targeted coordinated alerts–static conflict cell flags or brief conflict tracks–to maintain better SR across various challenging scenarios. While it incurs more collisions (CO) and sometimes longer makespan (MS) compared to highly specialized methods like SCRIMP, and its SR declines in extremely dense scenarios, it often delivers better overall feasibility and scalability, in scenarios with increasing agent counts, than purely search-based methods and PRIMAL by strategically sharing minimum inter-agent information.

Effective multi-agent coordination with reduced information exchange, as explored in this research, will lead to privacy-aware automated systems, enabling broader use in complex privacy-preserving applications like autonomous driving. Despite these promising results, our work has limitations. The current framework, optimized for *disappear at target* setting, may face challenges in *stay at target* behavior. Future work will investigate the minimal information necessary to effectively manage these *stay at target* scenarios, aiming to extend the framework's applicability without compromising its core principle of sparse information exchange. Additionally, an ablation study of richer `AlertMask` encodings, like dynamic occupancy maps, could improve replanning efficiency.

References

Boyarski, E.; Felner, A.; Stern, R.; Sharon, G.; Betzalel, O.; Tolpin, D.; and Shimony, E. 2015. Icbs: The improved conflict-based search algorithm for multi-agent pathfinding. In *Proceedings of the International Symposium on Combinatorial Search*, volume 6, 223–225.

Damani, M.; Luo, Z.; Wenzel, E.; and Sartoretti, G. 2021a. PRIMAL _2: Pathfinding via reinforcement and imitation multi-agent learning-lifelong. *IEEE Robotics and Automation Letters*, 6(2): 2666–2673.

Damani, M.; Luo, Z.; Wenzel, E.; and Sartoretti, G. 2021b. PRIMAL2: Pathfinding via Reinforcement and Imitation Multi-Agent Learning—Lifelong. *IEEE Robotics and Automation Letters*, 6(2): 2666–2673.

He, K.; Zhang, X.; Ren, S.; and Sun, J. 2016. Deep Residual Learning for Image Recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 770–778.

Li, J.; Chen, Z.; Harabor, D.; Stuckey, P. J.; and Koenig, S. 2022. MAPF-LNS2: Fast Repairing for Multi-Agent Path Finding via Large Neighborhood Search. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, 10256–10265.

Ren, J.; Eric, E.; Kumar, T. K. S.; Koenig, S.; and Ayanian, N. 2025. Empirical Hardness in Multi-Agent Pathfinding: Research Challenges and Opportunities. In *Blue Sky paper at 24th International Conference on Autonomous Agents and Multiagent Systems*.

Sartoretti, G.; et al. 2019. PRIMAL: Pathfinding via Reinforcement and Imitation Multi-Agent Learning. *IEEE Robotics and Automation Letters*, 4(3): 2559–2566.

Schulman, J.; Levine, S.; Abbeel, P.; Jordan, M. I.; and Moritz, P. 2015a. Trust Region Policy Optimization. In *Proceedings of the International Conference on Machine Learning*, 1889–1897. PMLR.

Schulman, J.; Moritz, P.; Levine, S.; Jordan, M. I.; and Abbeel, P. 2015b. High-Dimensional Continuous Control Using Generalized Advantage Estimation. *arXiv preprint arXiv:1506.02438*.

Schulman, J.; Wolski, F.; Dhariwal, P.; Radford, A.; and Klimov, O. 2017. Proximal Policy Optimization Algorithms. *arXiv preprint arXiv:1707.06347*.

Sharon, G.; Stern, R.; Felner, A.; and Sturtevant, N. R. 2015. Conflict-Based Search for Optimal Multi-Agent Pathfinding. *Artificial Intelligence*, 219: 40–66.

Sharon, G.; Stern, R.; Goldenberg, M.; and Felner, A. 2013. The Increasing Cost Tree Search for Optimal Multi-Agent Pathfinding. *Artificial Intelligence*, 195(C): 470–495.

Skrynnik, A.; Andreychuk, A.; Nesterova, M.; Yakovlev, K.; and Panov, A. 2024. Learn to Follow: Decentralized Lifelong Multi-Agent Pathfinding via Planning and Learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, 17541–17549.

Stern, R.; Sturtevant, N.; Felner, A.; Koenig, S.; Ma, H.; Walker, T.; Li, J.; Atzmon, D.; Cohen, L.; Kumar, T.; et al. 2019a. Multi-agent pathfinding: Definitions, variants, and benchmarks. In *Proceedings of the International Symposium on Combinatorial Search*, volume 10, 151–158.

Stern, R.; Sturtevant, N. R.; Felner, A.; Koenig, S.; Ma, H.; Walker, T. T.; Li, J.; Atzmon, D.; Cohen, L.; Kumar, T. K. S.; Boyarski, E.; and Bartak, R. 2019b. Multi-Agent Pathfinding: Definitions, Variants, and Benchmarks. *Symposium on Combinatorial Search (SoCS)*, 151–158.

Van Hasselt, H.; Guez, A.; and Silver, D. 2016. Deep reinforcement learning with double q-learning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 30.

Wagner, G.; and Choset, H. 2011. M*: A Complete Multirobot Path Planning Algorithm with Performance Bounds. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 3260–3267.

Wang, Y.; Duhan, T.; Li, J.; and Sartoretti, G. A. 2025. LNS2+RL: Combining Multi-agent Reinforcement Learning with Large Neighborhood Search in Multi-agent Path Finding. In *Proceedings of the AAAI Conference on Artificial Intelligence*.

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

A Appendix

Appendix Contents

A.1 A Quantitative Model of Information Load

To quantitatively support our claim of reduced information sharing, we developed a simplified model to estimate the "Total Information Load" of different coordination paradigms. This model is based on a simple **Information Unit (IU)**, defined as the data required to represent one agent's state (e.g., its coordinates) at a single timestep.

Model Assumptions Our calculation is based on the **11x11 maze scenario with 20 agents**, using metrics from the successful *Alert-BFS* runs in Table 1a.

1. **Scenario Data:** We use a 20-agent instance (N=20) solved with a makespan (T) of **27 timesteps** and requiring the resolution of **191 conflicts** (C).
2. **Average Path Length (L):** We assume an average initial path length of **25 steps** per agent.
3. **Agent Density (D):** For a distributed method like PRIMAL with a 10x10 FoV on an 11x11 grid, each agent has near-global vision. Therefore, each agent continuously observes all other N-1 agents. We set the average density (D) to **19 neighbors**.
4. **Alert Size (I_{alert}):** We assume each conflict alert sent by our coordinator is a minimal, static alert costing **1 IU**.

Comparative Calculation

Distributed Method (e.g., PRIMAL) The information load is the continuous sensing and processing of all visible neighbors by every agent at every timestep. The formula is $\text{Info}_{\text{Distributed}} = N \times T \times D$.

$$\begin{aligned}\text{Info}_{\text{Dist.}} &= 20 \times 27 \times 19 \\ &= \textbf{10,260 IU}\end{aligned}$$

Our Hybrid Method (Alert-X) The load is the one-time path sharing in addition to all subsequent targeted alerts. The formula is $\text{Info}_{\text{Alert-X}} = (N \times L) + (C \times I_{\text{alert}})$.

$$\begin{aligned}\text{Info}_{\text{Alert-X}} &= (20 \times 25) + (191 \times 1) \\ &= 500 + 191 \\ &= \textbf{691 IU}\end{aligned}$$

Conclusion This analysis highlights the difference in information architecture. The distributed method requires each of its 20 agents to continuously sense and process a heavy stream of local data, amounting to a total load of **10,260 IU**. In contrast, our hybrid method offloads this burden to a central coordinator, resulting in a total information load of only **691 IU**.

This represents a **$\sim$93% reduction in the total information load**, quantifying the efficiency of our on-demand alert system. While our method has a central coordinator, the burden on each individual agent is drastically lower, as it does not require constant, high-bandwidth sensing of its environment.

A.2 Literature review

Conflict-Based Search (CBS) *S1 – Agent Planning: Decentralized*: Each agent independently computes its path from the start to the goal using a single-agent pathfinding algorithm (e.g., A^*). This decentralized planning allows for efficient initial path computation without considering other agents.

S2 – Collision Detection: Centralized: After individual paths are planned, CBS centrally examines these paths to detect conflicts, such as two agents occupying the same location at the same time (vertex conflicts) or swapping positions simultaneously (edge conflicts). This centralized detection ensures systematic identification of all potential conflicts.

S3 – Collision Avoidance Policy: Centralized: Upon detecting a conflict, CBS resolves it by adding constraints to the agents' paths. Specifically, it creates two new branches in a constraint tree, each imposing a restriction on one of the conflicting agents to avoid the conflict. This centralized policy ensures optimal conflict resolution by exploring different constraint combinations (Sharon et al. 2015).

S4 – Agent Replanning: Decentralized: With new constraints in place, only the agents affected by these constraints replan their paths. Each affected agent independently computes a new path that adheres to the added constraints, maintaining the decentralized nature of the replanning process.

Large Neighborhood Search (LNS) *S1 – Agent Planning: Centralized*: LNS-based methods, such as MAPF-LNS2, begin with a centralized planning phase where initial paths for all agents are computed using a fast, suboptimal solver like Prioritized Planning (PP). This initial solution may contain conflicts but serves as a starting point for further refinement (Li et al. 2022).

S2 – Collision Detection: Centralized: The system centrally identifies conflicts (e.g., vertex or edge collisions) in the initial set of paths. This global analysis ensures that all potential conflicts are detected and can be addressed in subsequent steps.

S3 – Collision Avoidance Policy: Centralized: LNS employs a centralized strategy to resolve conflicts by selecting subsets of agents (the "neighborhood") involved in collisions and replanning their paths. Techniques like Safe Interval Path Planning with Soft Constraints (SIPPS) are used to minimize the number of conflicts during this replanning phase.

S4 – Agent Replanning: Centralized: The selected subset of agents undergoes centralized replanning to resolve conflicts, while the paths of other agents remain unchanged. This process iterates, with different neighborhoods selected in each iteration, until a conflict-free set of paths is achieved or a predefined time limit is reached.

PRIMAL and PRIMAL2 *S1 – Agent Planning: Distributed*: In both PRIMAL and PRIMAL2, each agent independently plans its path using policies learned through a combination of RL and IL. These policies are trained to enable agents to navigate towards their goals based on local observations without centralized coordination (Sartoretti et al. 2019; Damani et al. 2021b).

S2 – Collision Detection: Distributed: Agents detect potential collisions based on their local observations of the environment. They do not rely on a centralized system to identify conflicts but instead use their learned policies to anticipate and respond to nearby agents.

S3 – Collision Avoidance Policy: Distributed: Collision avoidance is handled through the agents' learned behaviors. In PRIMAL2, enhancements such as improved observation types have been introduced to facilitate better implicit coordination among agents, especially in dense and structured environments.

S4 – Agent Replanning: Distributed: Agents continuously replan their paths in response to changes in their local environment. This reactive planning allows them to adapt to dynamic scenarios, such as new goal assignments in lifelong MAPF settings.

SCRIMP *S1 – Agent Planning: Distributed*: Each agent independently plans its path using a policy learned through a combination of RL and IL. Agents rely on a small local FOV (as small as 3×3) and a transformer-based communication mechanism to share information with nearby agents, enabling them to make informed decisions despite limited local observations (Wang et al. 2023).

S2 – Collision Detection: Distributed: Agents detect potential collisions based on their local observations and the messages received from neighboring agents through the communication mechanism. This decentralized approach allows agents to anticipate and respond to nearby agents without centralized coordination.

S3 – Collision Avoidance Policy: Distributed: Collision avoidance is handled through the agents' learned policies, which incorporate a state-value-based tie-breaking strategy. This strategy enables agents to resolve conflicts in symmetric situations by assigning priorities based on predicted long-term collective benefits and distances to goals.

S4 – Agent Replanning: Distributed: Agents continuously replan their paths in response to changes in their local environment, leveraging intrinsic rewards to encourage exploration and mitigate the long-term credit assignment problem. This decentralized replanning allows agents to adapt to dynamic scenarios effectively.

Learn to Follow (FOLLOWER) *S1 – Agent Planning: Decentralized*: Each agent independently plans its path to the assigned goal using a heuristic search algorithm (e.g., A*). To mitigate congestion, the planner incorporates a heatmap-based cost function that penalizes frequently occupied areas, encouraging agents to choose less crowded paths. A sub-goal (waypoint) is selected along the planned path to guide short-term movement (Skrynnik et al. 2024).

S2 – Collision Detection: Decentralized: Agents detect potential collisions based on their local observations. They do not rely on a centralized system to identify conflicts but instead use their learned policies to anticipate and respond to nearby agents.

S3 – Collision Avoidance Policy: Decentralized: Collision avoidance is handled through the agents' learned behaviors. A neural network-based policy guides the agent toward its sub-goal while avoiding collisions. The policy is trained using reinforcement learning, leveraging local observations without requiring global state information or inter-agent communication.

S4 – Agent Replanning: Decentralized: Agents continuously replan their paths in response to changes in their local environment. This reactive planning allows them to adapt to dynamic scenarios, such as new goal assignments in lifelong MAPF settings.

LNS2+RL *S1 – Agent Planning: Centralized*: LNS2+RL begins by centrally generating initial paths for all agents using a fast, suboptimal method like Prioritized Planning (PP). This initial solution may contain collisions but serves as a starting point for further refinement (Wang et al. 2025).

S2 – Collision Detection: Centralized: The system centrally identifies conflicts (e.g., vertex or edge collisions) in the initial set of paths. This global analysis ensures that all potential conflicts are detected and can be addressed in subsequent steps.

S3 – Collision Avoidance Policy: Hybrid: LNS2+RL employs a hybrid strategy for collision avoidance: **Early Iterations:** Utilizes a MARL policy to replan paths for subsets of agents involved in conflicts. This decentralized component allows agents to learn cooperative behaviors to avoid collisions. **Later Iterations:** Switches to a centralized PP algorithm for replanning, aiming to quickly resolve any remaining conflicts.

S4 – Agent Replanning: Hybrid: Replanning in LNS2+RL is conducted in a hybrid manner: **Early Iterations:** Selected subsets of agents undergo decentralized replanning using the MARL policy, promoting cooperative behavior. **Later Iterations:** Replanning shifts to a centralized approach using PP, focusing on efficiency and resolving any remaining conflicts.

A.3 Training Methods

PPO Training Procedure We train a single-agent navigation policy π_θ to reach a specified goal in the presence of both static and dynamic obstacles. Dynamic obstacles are each assigned a hidden goal and follow a precomputed trajectory, executing only valid moves; this setup allows us to generate online collision alerts during inference (cf. stage S3 of our fix–collisions algorithm).

Training is performed with Proximal Policy Optimization (PPO) (Schulman et al. 2017) on an 11×11 maze grid for $30{,}000$ episodes. Throughout the first $15{,}000$ episodes, we linearly increase the static-obstacle density from 0% to 30% and the number of dynamic obstacles from 0 to 4.

At each time step t, we compute the discounted return

$$R_t = \sum_{l=0}^{T-t} \gamma^l \, r_{t+l},$$

and the advantage estimate using Generalized Advantage Estimation (GAE) (Schulman et al. 2015b):

$$A_t = R_t - V_\theta(s_t),$$

where $\gamma = 0.95$ and $V_\theta(s_t)$ is the value function.

The PPO surrogate objective is

$$\begin{aligned} L_{\mathrm{PPO}}(\theta) = -\mathbb{E}_t\big[\min\big(r_t A_t,\, \mathrm{clip}(r_t, 1-\varepsilon, 1+\varepsilon)\, A_t\big)\big] \\ + c_v\, \mathbb{E}_t\big[\big(V_\theta(s_t) - R_t\big)^2\big] - c_e\, \mathbb{E}_t\big[H\big(\pi_\theta(\cdot \mid s_t)\big)\big], \end{aligned}$$

Table 2: Related Literature Categorization into – S1: Agent Planning; S2: Collision Detection; S3: Collision avoidance policy; S4: Agent Replanning (SB: Search Based; LB: Learning Based)

Method	**S1**	**S2**	**S3**	**S4**
Our*	Decentralized	Centralized	Centralized	Decentralized
(SB) CBS	Decentralized	Centralized	Centralized	Centralized
(SB) LNS	Centralized	Centralized	Centralized	Centralized
(LB) PRIMAL & PRIMAL-2	Distributed	Distributed	Distributed	Distributed
(LB) SCRIMP	Distributed	Distributed	Distributed	Distributed
(LB) Learn to Follow	Decentralized	Decentralized	Decentralized	Decentralized
(LB) LNS2+RL	Centralized	Centralized	Hybrid	Hybrid

where $r_t = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}$, $\varepsilon = 0.2$, $c_v = 0.5$, and c_e are linearly annealed from 0.05 down to 0.01 over the first $5{,}000$ episodes. PPO inherits many of the stability guarantees of trust-region methods (Schulman et al. 2015a).

To encourage the agent to distinguish valid from invalid moves, we add a binary cross-entropy loss

$$L_{\text{valid}}(\theta) = \mathbb{E}_t\big[\text{BCE}\big(z_t, m_t\big)\big],$$

where $z_t \in \mathbb{R}^{|\mathcal{A}|}$ are the network logits, $m_t \in \{0,1\}^{|\mathcal{A}|}$ is the action-validity mask, and

$$\text{BCE}(z,m) = -\sum_{i=1}^{|\mathcal{A}|}\big[m_i \log\sigma(z_i) + (1-m_i)\log\big(1-\sigma(z_i)\big)\big].$$

The full objective is

$$L(\theta) = L_{\text{PPO}}(\theta) + \lambda_{\text{valid}}\, L_{\text{valid}}(\theta),$$

where λ_{valid} is chosen empirically. During both training and inference, we sample only actions flagged as valid by m_t, which accelerates convergence and improves safety.

After training, the policy π_θ is used in stage S1 to generate initial trajectories and in stage S4 to replan whenever the centralized collision detector (stage S2) issues an alert.

Neural Network Architecture

ResNetDQN Network Architecture for DDQN Training ResNetDQN is a residual-network architecture for approximating the action-value function $Q^\pi(s,a)$ in grid-based environments trained using the DDQN algorithm (Van Hasselt, Guez, and Silver 2016). It combines a deep convolutional stem with residual blocks (He et al. 2016), early fusion of low-dimensional features, hierarchical downsampling, and a late-fusion MLP to output one Q-value per action.

Network Overview

1. ***Input:*** $(B, 6, H, W)$ tensor comprising four binary masks (obstacle, agent, goal, dynamic) plus normalized x- and y-coordinate channels, and a $(B, 3)$ low-dim vector of $\{\texttt{direction}, \texttt{distance}\}$.
2. ***Conv Stem & Residual Blocks:*** 3×3 conv (6→32 channels, stride=1, pad=1) + BatchNorm + ReLU, then two `ResidualBlock(32→32)` with dilation rates 1 and 2.
3. ***Early Fusion:*** Project low-dim (3→16), tile to $(B, 16, H, W)$, concat with conv features → 48 channels, then 3×3 conv (48→32) + BatchNorm + ReLU.
4. ***Downsampling Stage 1:*** 3×3 conv (32→64, stride=2, pad=1) + BatchNorm + ReLU, followed by two `ResidualBlock(64→64)` with dilation rates 2 and 4.
5. ***Downsampling Stage 2:*** 3×3 conv (64→128, stride=2, pad=1) + BatchNorm + ReLU, followed by two `ResidualBlock(128→128)` with dilation rates 4 and 1.
6. ***Global Pooling & Late Fusion:*** AdaptiveAvgPool2d → 128-dim vector $\mathbf{u}$. In parallel, map low-dim→256, map $\mathbf{u} \to 256$, concat →512 →256 via FC + ReLU.
7. ***Output:*** Linear layer (256→$|\mathcal{A}|$) produces Q-values.
8. ***Initialization:*** All conv and linear weights: Xavier-uniform; all biases: zero.

Implementation and Packages

- `torch`, `torch.nn`, `torch.nn.functional`: define modules, layers, activations.
- `AdaptiveAvgPool2d`, `BatchNorm2d`, `Conv2d`, `Linear`: core building blocks.

ResNet-based Actor–Critic Architecture for PPO Training The PPOActorCritic model shares the same ResNet encoder as ResNetDQN (conv stem, residual blocks, fusion, downsampling, pooling) (He et al. 2016), producing a 256-dim embedding. It splits into two GRU-based heads for policy (actor) and value (critic), trained via PPO (Schulman et al. 2017).

Network Overview

1. ***Shared Encoder:*** Follows Steps 1–4 from ResNetDQN, yielding a 256-dim hidden $\mathbf{h}_{\text{shared}}$.

2. ***Actor Head:*** GRUCell(256→256) updates hidden state $\mathbf{h}_t^{\pi}$; Linear(256→$|\mathcal{A}|$) produces action logits.
3. ***Critic Head:*** GRUCell(256→256) updates hidden state $\mathbf{h}_t^{V}$; Linear(256→1) produces scalar state-value estimate.
4. ***Initialization:*** Same Xavier-uniform for all weights, zero biases.

Implementation and Packages

- `torch`, `torch.nn`, `torch.nn.functional`: define encoder, GRUs, heads.
- `GRUCell`, `Linear`: recurrent and output modules.

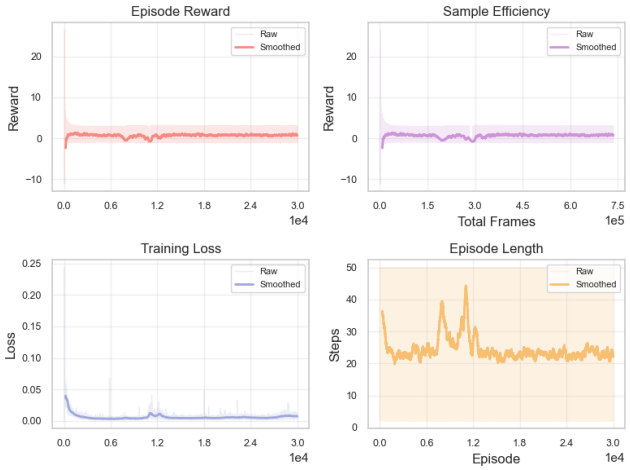


Figure 2: The plot illustrates the training performance of the *Double Deep Q-Network (DDQN)* algorithm. Episode rewards (red), sample efficiency measured by rewards per total frames (purple), training loss (blue), and episode length (orange) are presented across episodes. Smoothed curves represent moving averages, enhancing the visibility of underlying performance trends.

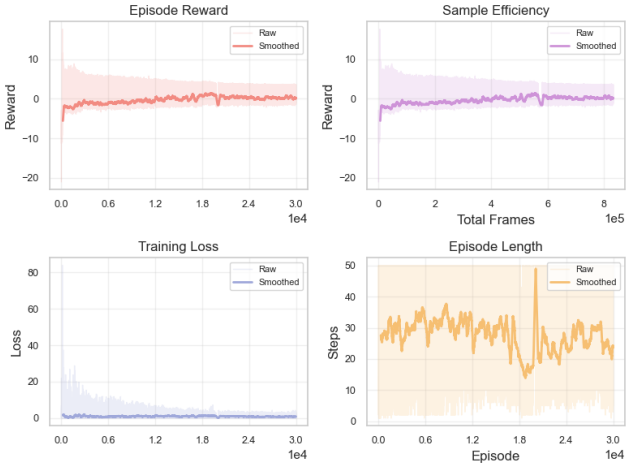


Figure 3: The plot illustrates the training performance of the *Proximal Policy Optimization (PPO)* algorithm, capturing episode rewards (red). Episode rewards (red), sample efficiency measured by rewards per total frames (purple), training loss (blue), and episode length (orange) are presented across episodes. Smoothed curves represent moving averages, enhancing the visibility of underlying performance trends.

Training Results

Model Comparison

Agent Selection Policy Comparison

A.4 Additional Experiments

We present a model comparison trained using DDQN and PPO reinforcement learning algorithms, Table 3, in Section A.3. This section details further empirical evaluations and an ablation study of our framework.

An ablation study of different agent selection policies is detailed in Table 4. We have considered the information sharing setting/replanning strategy as *Static & Dynamic Obstacle Replanning*.

Furthermore, a comparison of replanning settings for static and dynamic obstacles is provided in Table 5, which reflects different information sharing settings as detailed in our main paper. Here, we considered the agent selection policy as *Fewest Future Collisions (FFC)*.

We also present extended analyses on MovingAI lab's MAPF benchmark dataset (Stern et al. 2019b) – random map $32 \times 32 - 10$ environment with various agent counts [8, 16, 32, 64, 96], results presented in Section A.4.

Table 3: Performance comparison of reinforcement learning methods (BFS-DQN, BFS-PPO, A*-DQN, A*-PPO) across different grid environments and agent counts. Results are averaged over 10 problem instances per configuration. Evaluated metrics include Success Rate (SR), Makespan (MS), Collisions (CO), and Time (T), with the Time metric averaged over all trials. ***Shows that the model trained using DDQN, selected for our main paper, yields better results over PPO model.***

(a) 11x11 maze grid, ∼45% static obstacles (5–20 agents). Time in seconds.

	SR (↑)			MS (↓)			CO (↓)			T (s) (↓)		
Methods	5-10	11-15	16-20	5-10	11-15	16-20	5-10	11-15	16-20	5-10	11-15	16-20
BFS-DDQN	**100.00%**	**100.00%**	**98.00%**	17.8	23.6	27.4	**9**	53	191	**1.1**	**4.2**	**11.8**
BFS-PPO	**100.00%**	94.00%	74.00%	22.1	28.6	28.6	**9**	**48**	**125**	4.9	16.5	31.8
A*-DDQN	**100.00%**	96.00%	82.00%	**16.1**	**20.7**	**24.0**	10	59	173	3.3	12.0	25.8
A*-PPO	**100.00%**	96.00%	74.00%	16.6	21.4	25.2	10	60	171	3.9	14.4	30.4

(b) 21x21 maze grid, ∼35% static obstacles (32, 64, and 96 agents). Time in minutes.

	SR (↑)			MS (↓)			CO (↓)			T (min) (↓)		
Methods	32	64	96	32	64	96	32	64	96	32	64	96
BFS-DDQN	**100.00%**	**90.00%**	0.00%	86.6	131.4	-	528	18909	-	**2.7**	23.9	33.5
BFS-PPO	**100.00%**	0.00%	0.00%	87.6	-	-	463	-	-	11.2	30.2	31.1
A*-DDQN	**100.00%**	80.00%	0.00%	41.4	**72.0**	-	457	11379	-	3.3	**23.5**	33.4
A*-PPO	**100.00%**	0.00%	0.00%	**39.6**	78.3	-	**407**	**10761**	-	6.2	27.0	30.2

(c) 25x25 warehouse grid, ∼24% static obstacles (32, 64, and 96 agents). Time in minutes.

	SR (↑)			MS (↓)			CO (↓)			T (min) (↓)		
Methods	32	64	96	32	64	96	32	64	96	32	64	96
BFS-DDQN	**100.00%**	**100.00%**	10.00%	94.3	125.9	96.0	237	6379	26553	5.2	28.0	59.5
BFS-PPO	**100.00%**	40.00%	0.00%	93.6	130.0	-	231	3740	-	15.2	57.3	60.2
A*-DDQN	**100.00%**	**100.00%**	**60.00%**	**40.5**	43.7	**50.8**	211	3129	**16277**	**4.7**	**19.2**	**51.8**
A*-PPO	**100.00%**	**100.00%**	40.00%	41.6	**42.4**	51.5	**179**	**3117**	17382	5.7	30.0	55.4

Table 4: Performance comparison of Agent Selection Policies (Random, Farthest, Fewest Future Collisions - FFC) for Alert-BFS and Alert-A* methods across different grid environments and agent counts. Metrics include Success Rate (SR), Makespan (MS), Collisions (CO), and Time (T). Results are averaged over 10 problem instances per configuration. Best performance across policies for each method and agent group is bolded. *Using Static & Dynamic obstacle replanning*. ***Shows that, among the choices [random, farthest & fewest future collisions (FFC)], the FFC policy, selected for our main paper, yields best results across the different map configurations.***

(a) 11x11 maze, ∼45% static obstacles (5–20 agents). Time in seconds (s).

	SR (↑)			MS (↓)			CO (↓)			T (s) (↓)		
Methods	5-10	11-15	16-20	5-10	11-15	16-20	5-10	11-15	16-20	5-10	11-15	16-20
Policy: Random Agent Selection												
Alert-BFS	98.33%	98.00%	82.00%	20.5	24.6	32.7	20	108	517	2.0	5.8	22.6
Alert-A*	98.33%	96.00%	50.00%	17.8	23.6	24.3	16	122	259	4.3	15.7	35.6
Policy: Farthest Agent Selection												
Alert-BFS	**100.00%**	94.00%	90.00%	19.8	24.6	30.3	14	91	387	1.2	6.9	17.8
Alert-A*	98.33%	84.00%	62.00%	18.9	23.3	26.5	14	96	224	4.6	18.3	35.1
Policy: Fewest Future Collisions (FFC)												
Alert-BFS	**100.00%**	**100.00%**	**98.00%**	17.8	23.6	27.4	**9**	**53**	191	**1.1**	**4.2**	**11.8**
Alert-A*	**100.00%**	96.00%	82.00%	**16.1**	**20.7**	**24.0**	10	59	**173**	3.3	12.0	25.8

(b) 21x21 maze grid, ∼35% static obstacles (32, 64, and 96 agents). Time in minutes (min).

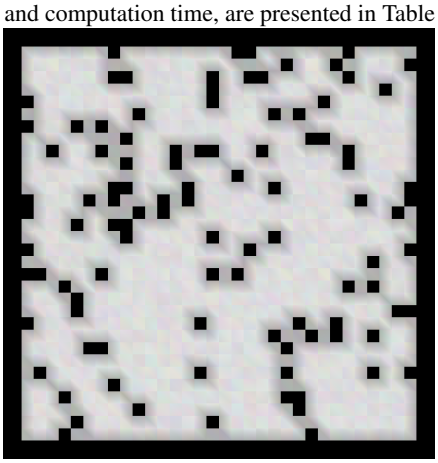
	SR (↑)			MS (↓)			CO (↓)			T (min) (↓)		
Methods	32	64	96	32	64	96	32	64	96	32	64	96
Policy: Random Agent Selection												
Alert-BFS	**100.00%**	0.00%	0.00%	99.2	-	-	1670	-	-	4.1	30.1	30.4
Alert-A*	**100.00%**	0.00%	0.00%	48.8	-	-	931	-	-	6.1	30.2	30.2
Policy: Farthest Agent Selection												
Alert-BFS	**100.00%**	0.00%	0.00%	93.5	-	-	1216	-	-	3.1	30.2	30.2
Alert-A*	**100.00%**	10.00%	0.00%	47.5	**64.0**	-	697	14654	-	5.3	29.8	30.3
Policy: Fewest Future Collisions (FFC)												
Alert-BFS	**100.00%**	**90.00%**	0.00%	86.6	131.4	-	528	18909	-	**2.7**	23.9	30.5
Alert-A*	**100.00%**	80.00%	0.00%	**41.4**	**72.0**	-	**457**	**11379**	-	3.3	**23.5**	30.4

(c) 25x25 warehouse grid, ∼24% static obstacles (32, 64, and 96 agents). Time in minutes (min).

	SR (↑)			MS (↓)			CO (↓)			T (min) (↓)		
Methods	32	64	96	32	64	96	32	64	96	32	64	96
Policy: Random Agent Selection												
Alert-BFS	**100.00%**	20.00%	0.00%	119.0	139.0	-	722	23853	-	12.4	57.9	60.1
Alert-A*	**100.00%**	**100.00%**	0.00%	42.4	54.6	-	302	5788	-	6.0	37.9	60.1
Policy: Farthest Agent Selection												
Alert-BFS	**100.00%**	60.00%	0.00%	83.4	126.0	-	455	9836	-	8.5	47.6	60.8
Alert-A*	**100.00%**	**100.00%**	0.00%	45.0	59.4	-	268	4644	-	6.4	34.9	60.1
Policy: Fewest Future Collisions (FFC)												
Alert-BFS	**100.00%**	**100.00%**	10.00%	94.3	125.9	96.0	237	6379	26553	5.2	28.0	59.5
Alert-A*	**100.00%**	**100.00%**	**60.00%**	**40.5**	**43.7**	**50.8**	**211**	**3129**	**16277**	**4.7**	**19.2**	**51.8**

Inter-agent Information Sharing Comparison

Map Configuration: $32 \times 32 - 10$ *Random Map*
The random map $32 \times 32 - 10$ is obtained from the Moving AI Lab's MAPF benchmark dataset (Stern et al. 2019b). Agent configurations tested include scenarios with 8, 16, 32, 64, and 96 agents. Each configuration was tested across 10 different problem instances (unique start and goal locations for all agents). The detailed performance metrics, including success rates, makespan, collisions, and computation time, are presented in Table 6.



Benchmark Dataset Evaluation

Table 5: Comparison of replanning settings (Only Static, Only Dynamic, Static & Dynamic Obstacles) for Alert-BFS and Alert-A* methods across different grid environments and agent counts. Metrics include Success Rate (SR), Makespan (MS), Collisions (CO), and Time (T). Results are averaged over 10 problem instances per configuration. Best performance across settings for each method and agent group is bolded. *Using FFC agent selection policy.* ***Shows that, among the replanning strategies, Static + Dynamic obstacle replanning strategy (selected for our main paper), yields best results across the different map configurations.***

(a) 11x11 maze grid, ∼45% static obstacles (5–20 agents). Time in seconds (s).

	SR (↑)			MS (↓)			CO (↓)			T (s) (↓)		
Methods	5-10	11-15	16-20	5-10	11-15	16-20	5-10	11-15	16-20	5-10	11-15	16-20
Setting 1: Only Static Obstacle Replanning (No inter-agent information)												
Alert-BFS	95.00%	92.00%	84.00%	17.5	23.0	25.4	8	51	205	1.4	4.6	14.2
Alert-A*	95.00%	90.00%	78.00%	15.6	20.3	23.7	8	55	177	3.2	11.6	26.0
Setting 2: Only Dynamic Obstacle Replanning												
Alert-BFS	63.33%	24.00%	8.00%	19.3	21.0	26.8	16	89	171	4.9	17.9	34.7
Alert-A*	50.00%	12.00%	2.00%	14.0	13.8	15.0	6	25	49	22.9	41.4	52.1
Setting 3: Static & Dynamic Obstacle Replanning												
Alert-BFS	**100.00%**	**100.00%**	**98.00%**	17.8	23.6	27.4	9	53	191	**1.1**	**4.2**	**11.8**
Alert-A*	**100.00%**	96.00%	82.00%	16.1	20.7	24.0	10	59	173	3.3	12.0	25.8

(b) 21x21 maze grid, ∼35% static obstacles (32, 64, and 96 agents). Time in minutes (min).

	SR (↑)			MS (↓)			CO (↓)			T (min) (↓)		
Methods	32	64	96	32	64	96	32	64	96	32	64	96
Setting 1: Only Static Obstacle Replanning (No inter-agent information)												
Alert-BFS	**100.00%**	10.00%	0.00%	86.6	140.0	-	528	12476	-	5.8	30.0	30.0
Alert-A*	**100.00%**	20.00%	0.00%	**41.4**	**59.0**	-	**457**	**9310**	-	6.7	30.0	30.0
Setting 2: Only Dynamic Obstacle Replanning												
Alert-BFS	0.00%	0.00%	0.00%	-	-	-	-	-	-	30.0	30.0	30.0
Alert-A*	0.00%	0.00%	0.00%	-	-	-	-	-	-	30.0	30.0	30.0
Setting 3: Static & Dynamic Obstacle Replanning												
Alert-BFS	**100.00%**	**90.00%**	0.00%	86.6	131.4	-	528	18909	-	**2.7**	23.9	30.0
Alert-A*	**100.00%**	80.00%	0.00%	**41.4**	72.0	-	**457**	11379	-	3.3	**23.5**	30.0

(c) 25x25 warehouse grid, ∼24% static obstacles (32, 64, and 96 agents). Time in minutes (min).

	SR (↑)			MS (↓)			CO (↓)			T (min) (↓)		
Methods	32	64	96	32	64	96	32	64	96	32	64	96
Setting 1: Only Static Obstacle Replanning (No inter-agent information)												
Alert-BFS	**100.00%**	**100.00%**	0.00%	95.8	117.4	-	230	6097	-	10.0	40.0	60.2
Alert-A*	**100.00%**	**100.00%**	**60.00%**	42.2	**41.6**	56.0	**204**	3258	16694	8.9	32.3	53.3
Setting 2: Only Dynamic Obstacle Replanning												
Alert-BFS	0.00%	0.00%	0.00%	-	-	-	-	-	-	61.7	62.4	60.6
Alert-A*	0.00%	0.00%	0.00%	-	-	-	-	-	-	58.6	61.6	61.8
Setting 3: Static & Dynamic Obstacle Replanning												
Alert-BFS	**100.00%**	**100.00%**	10.00%	94.3	125.9	96.0	237	6379	26553	5.2	28.0	59.5
Alert-A*	**100.00%**	**100.00%**	**60.00%**	**40.5**	43.7	**50.8**	211	**3129**	**16277**	**4.7**	**19.2**	**51.8**

Table 6: Comparison of performance metrics for different methods on a 32×32 random map. Results are averaged over 10 problem instances per configuration and presented in two subtables based on agent counts. Key metrics include Success Rate (SR), Makespan (MS), Collisions (CO), and Time (T(min)), with Time averaged over all trials. For learning-based methods like PRIMAL and SCRIMP, the average steps taken (episode length) are presented in brackets along with average wall-clock time taken. For learning based methods, the environment step limit is set to 512. Best performing values for each metric per agent count are highlighted in bold.

(a) 32x32 random map (8, 16, and 32 agents). The time limit per problem instance is 1 hour. For learning based methods, the environment step limit is set to 512.

	SR (↑)			MS (↓)			CO (↓)			T (min) (↓)		
Methods	8	16	32	8	16	32	8	16	32	8	16	32
Alert-BFS	**100.00%**	**100.00%**	**100.00%**	62.5	76.2	104.9	2	9	64	0.2	0.6	2.4
Alert-A*	**100.00%**	**100.00%**	**100.00%**	42.7	45.5	48.1	2	8	68	0.2	0.9	2.5
CBS	**100.00%**	**100.00%**	90.00%	40.2	43.6	47.0	1	6	35079	**0.01**	**0.01**	13.0
ICBS	**100.00%**	**100.00%**	90.00%	40.2	43.6	47.0	1	6	15438	**0.01**	**0.01**	10.7
PRIMAL	0.00%	0.00%	0.00%	-	-	-	-	-	-	0.1 (512)	0.3 (512)	1.0 (512)
SCRIMP	**100.00%**	**100.00%**	**100.00%**	**39.4**	**43.3**	**47.7**	**0**	**0**	**0**	**0.01** (39)	**0.01** (43)	**0.03** (48)

(b) 32x32 random map (64 and 96 agents). The time limit per problem instance is 1 hour. For learning based methods, the environment step limit is set to 512.

	SR (↑)		MS (↓)		CO (↓)		T (min) (↓)	
Methods	64	96	64	96	64	96	64	96
Alert-BFS	**100.00%**	**100.00%**	136.5	154.4	1194	7079	12.9	33.8
Alert-A*	**100.00%**	**100.00%**	53.2	**52.0**	1158	5912	13.2	30.7
CBS	0.00%	0.00%	-	-	-	-	60.1	60.1
ICBS	0.00%	0.00%	-	-	-	-	60.3	60.2
PRIMAL	0.00%	0.00%	-	-	-	-	3.6 (512)	8.4 (512)
SCRIMP	**100.00%**	90.00%	**52.1**	55.8	**2**	**3**	**0.2** (52)	**0.5** (101)