Local Sensing to Global Hotspots: Cross-Attention Graph Fusion for Congestion-Aware Multi-Agent Path Finding

Shuvra Neel Roy
TCS Research
Tata Consultancy Services (TCS)
Kolkata, India
shuvraneel.roy@tcs.com

Arup Kumar Sadhu
TCS Research
Tata Consultancy Services (TCS)
Kolkata, India
arupk.sadhu@tcs.com

Ranjan Dasgupta
TCS Research
Tata Consultancy Services (TCS)
Kolkata, India
ranjan.dasgupta@tcs.com

***Abstract*—How can decentralized agents (here robots) predict and mitigate global traffic jams? This paper introduces a unified framework that forecasts warehouse-wide congestion costmaps using only locally-sensed data and utilizes these predictions to optimize multi-agent path finding (MAPF). We model the environment using two graph representations: a dynamic RoboGraph, derived from local robot interactions, and a static NavGraph, representing the warehouse topology. A Graph Fusion module, implemented via cross-attention, projects sparse, learned robot dynamics onto the global map to predict congestion hotspots. We then propose the congestion-aware MAPF (CA-MAPF) algorithm that leverages these predictions to steer agents through low-density zones. Simulation results demonstrate that our model scales linearly with fleet size and, when integrated into the planner, significantly improves system throughput compared to standard baselines.**

***Index Terms*—Congestion costmap; Temporal Graph Neural Network; robot graph; navigation graph**

I. Introduction

Modern Automated Warehouse Management Systems (AWMS) rely on the coordination of hundreds of loosely coupled collaborative robots [1]. While decentralization enhances robustness, it introduces a critical vulnerability: congestion. When robots contend for the same workspace without knowledge of global traffic flows, bottlenecks emerge, cascading into system-wide delays [2]. This disparity establishes a significant research gap: developing a methodology to generate accurate, global congestion forecasts when robots are restricted to sensing only their immediate neighborhood. To bridge this local sensing to global prediction gap, this paper introduces a novel deep learning architecture that learns to predict warehouse-wide congestion costmaps using only the decentralized data locally available from all robots.

Existing approaches largely treat congestion reactively [3], [4] or rely on centralized, global information for prediction [1], [2], [5], [6], [7]. This conflicts with the decentralized reality of modern fleets. To bridge the gap between local sensing and global efficiency, we propose a twofold framework. First, we introduce a deep learning architecture that predicts global congestion costmaps using only decentralized local data. Second, we utilize these predictions to drive a CA-MAPF algorithm.

Our framework relies on a dynamic RoboGraph ($\mathcal{G}_R(t)$) to capture robot interactions and a static NavGraph ($\mathcal{G}_N$) for map topology. A cross-attention Graph Fusion module projects sparse robot states onto the dense map, generating a Spatio-Temporal Robot Density (STRD) forecast. We proposed the Congestion-Weighted A* (CWA*), a low-level path planner that integrates STRD forecast into a hybrid cost function, allowing agents to proactively avoid future bottlenecks due to congestion, which result in the CA-MAPF algorithm. The key contributions of this paper are:

- A local-sensing-to-global-prediction network using cross-attention Graph Fusion.
- The proposed CA-MAPF algorithm, which utilizes learned costmaps to improve system throughput.

Simulation results demonstrate linear inference scaling and superior throughput over standard MAPF baseline [8].

This paper is structured as follows. Section II introduces preliminary definitions and the problem statement. Section III highlights the training and algorithm details of our approach. Section IV presents the simulation results and analysis. Finally, Section V concludes the paper with relevant future direction.

II. Problem Formulation

We consider a decentralized fleet of R loosely coupled robots operating within a warehouse environment. The environment and agents are modeled using two distinct graph representations: a static navigation graph and a dynamic robot graph.

A. Graph Representation

1) Static NavGraph ($\mathcal{G}_N$): The warehouse topology is represented by a static graph defined in Definition 2.1.

Definition 2.1: The *NavGraph* is indicated by $\mathcal{G}_N = (N, \mathcal{E}_N, \mathbf{X}_N)$, where N is set of nodes of the navigation graph $\mathcal{G}_N$, $\mathcal{E}_N$ is the edge connectivity between the nodes of $\mathcal{G}_N$, and $\mathbf{X}_N$ is the feature vector of each node. Here, $\mathbf{X}_N$ is given by the tuple $< p_n^x, p_n^y, l_n, u_n >, \forall n \in N$, where $< p_n^x, p_n^y >$ is the Cartesian coordinate of n^{th} node, l_n being the node type

(e.g., [goal, pick, isle, charging-point]), and u_n is a unique node descriptor.

2) Dynamic RoboGraph ($\mathcal{G}_R(t)$): The evolving state of the fleet is captured by a dynamic graph with temporal features. RoboGraph is defined in Definition 2.2.

Definition 2.2: The *RoboGraph* is indicated by $\mathcal{G}_R(t) = (R, \mathcal{E}_R(t), \mathbf{X}_R(t))$, where R indicates set of robots involved in forming $\mathcal{G}_R(t)$. $\mathcal{E}_R(t)$ is the edge connections between the nodes of $\mathcal{G}_R(t)$, and $\mathbf{X}_R(t)$ is the robot (or node) feature vector at time t. The node feature vector $\mathbf{X}_R(t)$ of $\mathcal{G}_R(t)$ is given by the tuple $< p_i^x, p_i^y, v_i^x, v_i^y, g_i^x, g_i^y, P_i^x, P_i^y, u_i >, \forall i \in R$. Here, $< p_i^x, p_i^y >$ is the Cartesian coordinate of robot i, $< v_i^x, v_i^y >$ are the velocity components along x-axis and y-axis respectively, $< g_i^x, g_i^y >$, is the Cartesian coordinate of goal, $< P_i^x, P_i^y >$ are the set of planned waypoints, and u_i is the unique identifier for robot i.

We propose two types of *RoboGraph*: one is used for interaction among robots as shown in Fig. 1(b) and another is used for ground-truth data generation [Fig. 1(a)] and is used as privileged information during training.

3) Ground Truth Generation: We calculate the *Spatio-Temporal Robot Density* (STRD) signal, ρ_i for robot $i \in R$ by considering all neighbours $\mathcal{N}_i(t)$ within a sensing radius d_{strd}, (Fig. 1(a)). The STRD quantifies local crowdedness by (1).

$$\rho_i(i, \mathcal{N}_i(t), t) = \frac{\mathcal{N}_i(t)}{\mathcal{N}_i(t)+1} \sum_{i=1, i\neq k}^{\mathcal{N}_i(t)} \frac{1}{||p_{ik}(t)||}, \tag{1}$$

where $||p_{ik}(t)||$ is the Euclidean distance to neighbor k. This metric is then projected onto the NavGraph ($\mathcal{G}_N$) to form the prediction labels τ^N. The labels τ^N are assigned following the rule defined in (2):

$$\tau^N(n,t) = \begin{cases} \rho_i(i, \mathcal{N}_i(t), t), & \text{if } n \text{ is occupied by } i, \\ 0, & \text{Otherwise.} \end{cases} \tag{2}$$

4) Inference Input: The edges $\mathcal{E}_R(t)$ of the input RoboGraph are formed using a stricter threshold d_{edge} (where $d_{edge} < d_{strd}$) as shown in Fig. 1(b). An edge exists between robots i and k only if $||p_{ik}(t)|| \leq d_{edge}$. This separation ensures the model learns to infer broad STRD signals (derived from d_{strd}) using only sparse, efficient local connections (derived from d_{edge}).

Specifically, a wider-ranging STRD is derived from the broadly connected RoboGraph $\tilde{\mathcal{G}}_R(t)$ (using threshold d_{strd}), while maintaining a sparse, computationally efficient graph structure $\mathcal{G}_R(t)$, defined by the tighter d_{edge} for GNN processing.

B. Problem Statement

The problem is twofold: global congestion prediction from local history, and subsequent CA-MAPF algorithm.

1) Local-to-Global Prediction: By Definition 2.2 $\mathbf{X}_R(t) \in \mathcal{G}_R(t)$ represents the decentralized local data available from the fleet at time t. Given a history of local observations over horizon $t_H : t$, we aim to learn a mapping function $\mathcal{F}_\theta$ that learns the evolution of the STRD signal over a future horizon $t : t_F$ and projects it onto the NavGraph $\mathcal{G}_N$, which we infer as the global congestion costmap τ^N, which is expressed by (3).

$$\tilde{\tau}^N_{t:t+t_F} = \mathcal{F}_\theta(\mathcal{G}_{R,t-t_H:t}, \mathcal{G}_N). \tag{3}$$

The objective is to accurately map the sparse, floating robot observations $\mathbf{X}_R(t)$ onto the dense, grid-aligned nodes of $\mathcal{G}_N$.

2) Congestion-Aware Planning: Utilizing the predicted costmap τ^N, we seek to compute optimal paths for all robots that minimize the collision probability and maximize the system throughput, thereby mitigating the "congestion collapse" phenomenon observed in standard MAPF baseline [8].

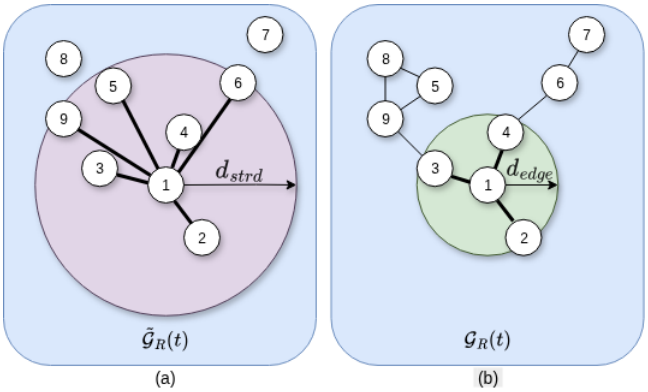

Fig. 1: (a) $\tilde{G}_R(t)$ defined by threshold d_{strd}. For robot $i = 1$, neighbors $\mathcal{N}_i(t) = \{2, 3, 4, 5, 6, 9\}$ form the edge set $\{(1,2), (1,3), (1,9), (1,5), (1,4), (1,6)\}$, used to compute STRD ρ_i by (1). (b) $G_R(t)$ defined by threshold d_{edge}. The neighbor set is reduced to $\mathcal{N}_i(t) = \{2, 3, 4\}$ with edges $\{(1,2), (1,3), (1,4)\}$. As these edges are a subset of $\tilde{G}_R(t)$, $G_R(t)$ is a spanning subgraph of $\tilde{G}_R(t)$ and is adopted as the primary RoboGraph hereafter.

III. Methodology

The proposed methodology consists of two primary components: a deep neural network that predicts global congestion costmaps from decentralized robot data, and a low level congestion-weighted A* (CWA*) planner that leverages these predictions resulting a high-level CA-MAPF to optimize fleet throughput.

A. Network Architecture

The architecture, illustrated in Fig. 2, processes the dynamic RoboGraph and static NavGraph through parallel encoders before fusing them to generate a unified congestion prediction.

1) Temporal Graph Encoder: This subsection highlights learning of the complex spatio-temporal dynamics from the *RoboGraph* $\mathcal{G}_R(t)$. It is implemented as a Recurrent Graph Neural Network (R-GNN), which processes the sequence of historical graph snapshots.

The primary objective of the *RoboEnc* is to process the features of each robot node into vector representations, or

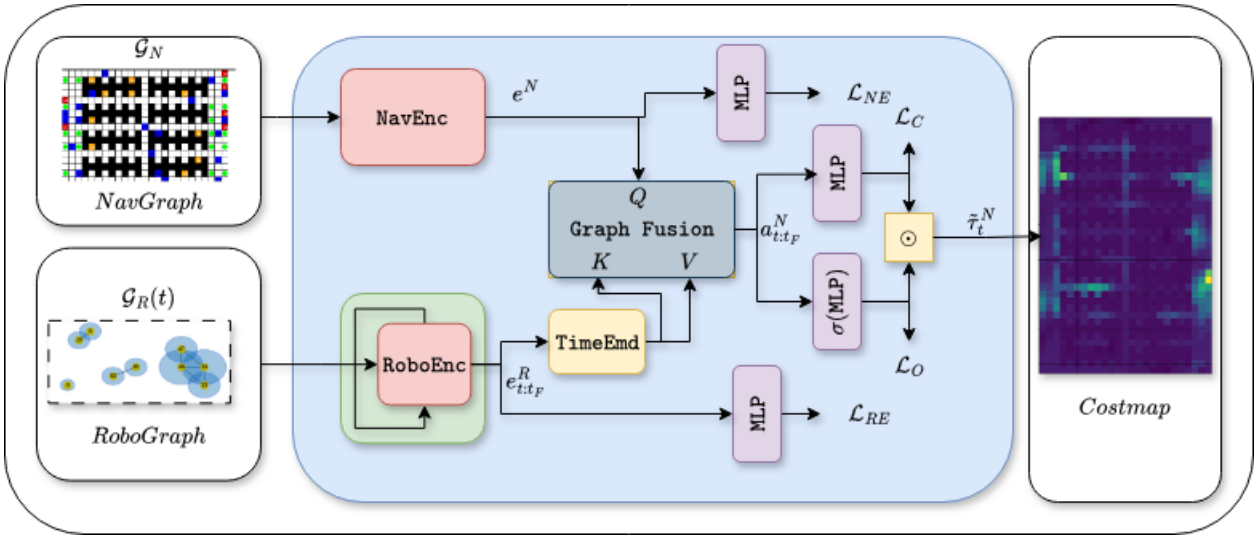

Fig. 2: Block-diagram of the proposed neural network. The radius of the circles in *RoboGraph* are indicative of the STRD values of each robot at time t. The proposed network uses a TGNN as the *RoboEnc* to learn the temporal nature of the STRD value upto t_F timestep. It is then projected onto the NavGraph via *GraphFusion*. The resultant congestion cost value is indicative of future cost of movement on the NavGraph.

embeddings, that capture both the robot's state and the relational dynamics of its local neighborhood. As a seq2seq model [9], [10], it encodes the historical graph sequence (e.g., from $(t-t_H{+}1)$ to t) to generate a sequence of future hidden-state embeddings, $\mathbf{e}^R_{t\in[t_0,t_F]}$, for the entire prediction horizon by (4).

These future embeddings are then passed through a time-distributed multilayer perceptron (MLP), which predicts the robot's future state $\tilde{\mathbf{Y}}^R_{ENC}(t)$ for each time step t in the horizon $[t_0, t_F]$ by (5). This labels $\mathbf{Y}^R_{ENC}(t)$ for the predicted state vector consists of the robot's future Cartesian coordinate $(p^x_i(t), p^y_i(t))$ and its future STRD $\rho_i(.)$.

$$\mathbf{e}^R_{t_H:t_F} = \texttt{RoboEnc}(\mathbf{G}_R(t)) \tag{4}$$

$$\tilde{\mathbf{Y}}^R_{ENC}(t) = \texttt{MLP}(\mathbf{e}^R_t), \tag{5}$$

The *RoboEnc* is trained using the mean squared error (MSE) loss function, $\mathcal{L}_{RE}$. This loss measures the error between the predicted states $\tilde{\mathbf{Y}}^R_{ENC}(t)$ and the labels or ground-truth states $\mathbf{Y}^R_{ENC}(t)$, averaged over the prediction horizon $[t_0, t_F]$.

$$\mathcal{L}_{RE} = \frac{1}{t_F}\sum_{t=t_0}^{t_F} \texttt{MSE}(\tilde{\mathbf{Y}}^R_{ENC}(t), \mathbf{Y}^R_{ENC}(t)), \tag{6}$$

2) Static Graph Encoder: In a typical map, the navigable space for the robot is conceptualized as a static graph. The nodes in the static graph represent areas on the navigable map with connections marking adjacency between neighboring nodes. In the present context, this warehouse map comprises grid-like nodes classified into regions such as obstacles (e.g., racks and walls), pick locations, drop-off points, and idling zones. Obstacles are non-navigable and thus excluded from the graph representation. Since the node properties and the connectivity of nodes in warehouse do not change with time, it is considered a static graph. We have designated this graph as the NavGraph $\mathcal{G}_N$.

The static graph encoder (*NavEnc*) processes the *NavGraph* ($\mathcal{G}_N$) defined in Definition 2.1. This module is used to generate node embeddings $\mathbf{e}^N : \mathbb{R}^{|N|\times E_d}$, where E_d is the embedding dimension. These embeddings are generated by applying a stack of graph convolutional networks (GCNs) to the node features of $\mathcal{G}_N$ which learns the spatial features and structure of the map. The resulting node representations serve as positional anchors, for grounding the location information within the $\mathcal{G}_R(t)$ embeddings. The embeddings $\mathbf{e}^N$ are computed by (7).

$$\mathbf{e}^N = \texttt{NavEnc}(\mathcal{G}^N), \tag{7}$$

In (7), `NavEnc` represents the NavGraph encoder. Embeddings are learned through a self-supervised learning scheme, where an MLP is conditioned to regress the initial node features $\mathbf{X}_N$ from the node embeddings $\mathbf{e}^N$ during training. The MLP layer has input size equal to the embedding dimension (E_d) and output size equal to the length of the input feature vector $\mathbf{X}_N$. The predicted node feature $\tilde{\mathbf{Y}}_N$ is given by (8).

$$\tilde{\mathbf{Y}}_N = \texttt{MLP}(\mathbf{e}^N). \tag{8}$$

The encoder learns map information by minimizing the reconstruction error between the original node features $\mathbf{X}_N$ and predicted node features $\tilde{\mathbf{Y}}_N$. During training, 50% of $\mathcal{G}_N$'s node features are randomly masked out to prevent overfitting and ensure robust embedding generation. The unsupervised loss function for the static graph encoder is defined by (9).

$$\mathcal{L}_{NE} = \frac{1}{|N|}\sum_{n\in N} \texttt{MSE}\left(\mathbf{x}^n_N, \tilde{\mathbf{y}}^n_N\right), \tag{9}$$

where $MSE(\mathbf{x}_N^n, \tilde{\mathbf{y}}_N^n)$ represents the mean squared error between the original node features $\mathbf{x}_N^n$ and the predicted features $\tilde{\mathbf{y}}_N^n$. Normalizing by $|N|$, the number of nodes in $\mathcal{G}_N$, ensures training stability. Once computed, these embeddings can be reused for downstream tasks as long as the map remains unchanged.

3) Graph Fusion: The *RoboEnc* and *NavEnc* modules produce two distinct representations.

- *RoboGraph* Embeddings ($\mathbf{e}_t^R$): RoboGraph embeddings are dynamic, sparse (low-dimensional, $|R|$ nodes), and contain the temporal STRD value. However, RoboGraph embeddings suffer from inherent positional inaccuracies.
- *NavGraph* Embeddings ($\mathbf{e}^N$): NavGraph embeddings are static, dense (high-dimensional, N nodes, where $N \gg R$), and provide a precise, spatial grounding of the warehouse map.

The core challenge is to accurately project the sparse, dynamic STRD value from $\mathcal{G}_R(t)$ onto the dense, static nodes of $\mathcal{G}_N$. This is achieved using a *Graph Fusion* module, which is implemented as a cross-attention network [11]. This mechanism allows each node in the *NavGraph* to look at all the nodes in the *RoboGraph* and selectively pull in the most relevant density information.

First, we enrich the *RoboGraph* embeddings with temporal context by generating time embeddings $\mathbf{k}_t^R$ for the prediction horizon $t \in [t_0, t_F]$ by (10). These are concatenated to create the final robot-state representation $\mathbf{A}_t^R$ by (11).

$$\mathbf{k}_{t_0:t_F}^R \leftarrow \texttt{TimeEmb}(t_0 : t_F). \tag{10}$$

$$\mathbf{A}_{t_0:t_F}^R \leftarrow \texttt{concat}(\mathbf{k}_{t_0:t_F}^R, \mathbf{e}_{t_0:t_F}^R). \tag{11}$$

Next, these representations are fed into a cross-attention layer to perform fusion for each timestep t in the prediction horizon, following these steps:

- **Query** (Q): The static *NavGraph* embeddings, $\mathbf{e}^N$. Each map node "asks": *What robot density is relevant to my location?*
- **Key** (K): The time-enriched *RoboGraph* embeddings, $\mathbf{A}_t^R$. Each robot node "answers" with its *state and position*.
- **Value** (V): The time-enriched *RoboGraph* embeddings, $\mathbf{A}_t^R$. Each robot node provides its STRD value to be projected.

The output of this operation is a new set of attended *NavGraph* embeddings, $\mathbf{g}_{t_0:t_F}^N$. These embeddings now effectively fuse the static map topology from $\mathbf{e}^N$ with the predicted dynamic density information from $\mathbf{A}_t^R$ by (12). This fusion process anchors the floating robot-centric density signals to the precise, grid-aligned nodes of the map, creating a robust, unified representation for the final prediction task.

$$\mathbf{g}_{t_0:t_F}^N \leftarrow \texttt{GraphFusion}(Q = \mathbf{e}^N, K = V = \mathbf{A}_{t_0:t_F}^R) \tag{12}$$

4) Congestion estimation: The attended embeddings $\mathbf{g}_{t_0:t_F}^N$, produced by the *GraphFusion* module, are used as the final representation for the prediction heads as shown in Fig. 2. Two separate time-distributed MLPs were employed to derive the final congestion costmap as shown in Fig. 2.

As shown in Fig. 2, we train a regression head by optimizing MSE loss ($\mathcal{L}_C$) to predict the continuous congestion value ($\tilde{\boldsymbol{\kappa}}_n^C(t)$) for each node $n \in N$ for timesteps $t_0 : t_F$ w.r.t the ground-truth STRD value τ^N computed by (1). Second, a separate classification head with a sigmoid activation is used to predict the binary node occupancy $\tilde{y}_n^O(t)$. This head is trained against the ground-truth occupancy $y_n^O(t)$ which is obtained after applying the step function over τ^N. This head is trained by employing Binary Cross-Entropy (BCE) loss, $\mathcal{L}_O$.

The final predictive congestion costmap $\tilde{\tau}_t^N$ is then calculated by an element-wise multiplication of $\tilde{y}_n^O(t)$ and $\tilde{\kappa}_n^C(t)$ as expressed by (13).

$$\tilde{\tau}_n^N(t) = \tilde{y}_n^O(t) \cdot \tilde{\kappa}_n^C(t) \tag{13}$$

This design ensured that a high congestion cost was only applied to nodes that the model also predicted were likely to be occupied, providing a sparse and actionable costmap.

A challenge in this domain is the inherent class imbalance, as most map nodes are unoccupied. To address this, the training process was designed with two mechanisms. First, the primary prediction losses, $\mathcal{L}_C$ and $\mathcal{L}_O$, were dynamically masked to keep equal numbers of occupied and unoccupied nodes. This focused the model's learning on relevant, non-zero regions. Second, a minor L2 regularization term, $\mathcal{L}_{Reg}$, was introduced to penalize any false-positive predictions on known unoccupied nodes.

5) Two-Phase Training Strategy: Training the complete, multi-modal network from a random initialization was found to be unstable. The complex cross-attention module overfits to noisy embeddings before the *RoboEnc* and *NavEnc* learns to produce coherent spatio-temporal and spatial representations respectively. To ensure stable convergence, a two-phase training strategy is implemented as explained below.

Phase 1: *Encoder Pretraining:* In phase 1, the *RoboEnc* and *NavEnc* are first trained independently to learn stable initial representations. The *RoboEnc* was trained on its state prediction task (minimizing $\mathcal{L}_{RE}$) and the *NavEnc* on its self-supervised reconstruction task (minimizing $\mathcal{L}_{NE}$) with 40% feature dropout. The combined pretraining loss is given by (14).

$$\mathcal{L}_{Pre} = \mathcal{L}_{RE} + 0.5 \cdot \mathcal{L}_{NE} \tag{14}$$

Phase 2: *End-to-End Joint Training:* In this phase, after the encoders are pretrained, the entire model was trained jointly. The focus of this phase was to fine-tune the full system for the final prediction task. The total loss $\mathcal{L}$ is the weighted sum of all loss components as expressed by (15).

$$\mathcal{L} = \delta \cdot \mathcal{L}_C + \gamma \cdot \mathcal{L}_O + \alpha \cdot \mathcal{L}_{RE} + \beta \cdot \mathcal{L}_{NE} + \epsilon \cdot \mathcal{L}_{Reg} \tag{15}$$

The hyperparameters were set to $\delta = 0.7$ and $\gamma = 0.4$, prioritizing the main prediction tasks. The original encoder losses,

$\mathcal{L}_{RE}$ and $\mathcal{L}_{NE}$, were retained with small weights ($\alpha = 0.02$, $\beta = 0.02$) to act as regularizers, preventing the encoders from drifting from their robust pretrained representations. The false-positive regularization $\mathcal{L}_{Reg}$ was weighted with $\epsilon = 0.0001$.

B. Congestion-Aware CBS Planning

To mitigate the predicted congestion, we integrate the learned costmaps (τ^N) into the MAPF framework (here, Conflict-Based Search (CBS)) [12], [13]. CBS is a two-level algorithm: a High-Level search that resolves conflicts among agents, and a Low-Level search that plans individual paths. We keep the High-Level conflict resolution mechanism intact to guarantees completeness. Our modification is strictly to the Low-Level Planner, replacing the standard A^* with the proposed CWA^*. We decompose the A^* cost function $f(n) = g(n) + h(n)$ to address both the reactive and strategic congestion.

1) Hybrid Step Cost ($g(n)$): In standard A^*, the step cost is usually taken as 1. We define the cost of traversing a node at time t using a decoupled time horizon, which is defined as $g(n) = 1 + Costmap(n, t)$ where t denotes path length from the start node m to current node n.

- Reactive Horizon ($t \leq t_F$) : Uses explicit, short-term crowd predictions from the congestion model, given by $\tilde{\tau}^N(n, t) \in \mathbb{R}$. This allows agents to perform "micro-waits" to let dynamic crowds pass.
- Strategic Horizon ($t > t_F$) : Uses an Exponential Moving Average (EMA) over historical average density $\hat{\tau}_t^N(n)$ evaluated by (16).

$$\hat{\tau}_t^N(n) = \frac{1}{t_F} \sum_{t}^{t_F} \tilde{\tau}^N(n, t). \tag{16}$$

This forms a "Static Risk Map", i.e $EMA(\hat{\tau}_t^N(n))$ which penalizes traversing historically high traffic zones (e.g. central intersections) even if they are currently empty, acting as a low-pass filter for risk.

EMA introduces a tunable hyperparameter "*Density Decay Factor*" denoted by η and is used to calculate $EMA(\hat{\tau}_t^N(n))$ given by $\hat{\tau}_t^{EMA} = \eta \times \hat{\tau}_t^N(n) + (1 - \eta) \times \hat{\tau}_{t-1}^{EMA}$. This η controls the rate of change of the static risk map over time.

2) Risk Inflated Heuristic ($h(n)$): To steer the search globally, we relax the admissibility constraint of standard A^*. We introduce another hyperparameter, which is the density-dependent "*Inflation Factor*" λ as a scaling factor to the risk function, given as $h = h_{eucl}(n) + \lambda \times Risk(n)$. The above formulation transforms the solver into a "*Congestion*" Weighted A^* search. While strictly suboptimal regarding path length ($h \geq h^*$), it is effectively "optimal" regarding time-to-destination in high-density scenarios. In our study, we use the same EMA weighted costmap ($\hat{\tau}_t^{EMA}$) as the function to represent $Risk(n)$. By inflating the estimated cost of congested regions, the heuristic effectively increases the gradient of the search space, naturally flowing agents like fluid through low-density lanes rather than high-density isles. Algorithm 1 uses $\hat{\tau}_t^{EMA}$, calculated at every time step and is sent to the low level planner, for all agents planning at time step t. In Algorithm 1, we denote $\hat{\tau}_t^N(n)$ at time t as Costmap $\mathcal{C}$ and $\hat{\tau}_t^{EMA}$ as Weighted Costmap $\mathcal{C}_w$.

Algorithm 1: Congestion Weighted A* Search

Input: Start s, Goal g, Map $\mathcal{M}$, Costmap $\mathcal{C}$, Weighted Costmap $\mathcal{C}_w$, Heuristic Inflation λ , Prediction Horizon t_F;
Output: Path Q, Cost J^a;
Initialize: $Open \leftarrow \{s\}$, $g(s) \leftarrow 0$, $f(s) \leftarrow h(s)$, $g(\text{others}) \leftarrow \infty$;
while $Open \neq \emptyset$ **do**
  $n \leftarrow \arg\min_{m \in Open} f(m)$;
  $Open \leftarrow Open \setminus \{n\}$;
  if $n = g$ **then**
    return $[Q, g(n)]$; ▷ Reconstruct Q via parent pointers
  end
  for *each neighbor* m *of* n **do**
    $h(m) \leftarrow dist(m, g) + \lambda * \mathcal{C}_w(g)$
    $t_{hop} \leftarrow$ *len*(*path_from_start*(m, s))
      ▷ Reconstruct path via parent pointers
    if $t_{hop} < t_F$ **then**
      $g(m) \leftarrow 1 + \mathcal{C}(m, t_{hop})$
    end
    else
      $g(m) \leftarrow 1 + \mathcal{C}_w(m)$
    end
    $cost \leftarrow g(n) + g(m)$;
    if $cost < g(m)$ **then**
      $parent(m) \leftarrow n$; $g(m) \leftarrow cost$;
      $f(m) \leftarrow cost + h(m)$;
      if $m \notin Open$ **then** $Open \leftarrow Open \cup \{m\}$;
    end
  end
end
return Failure;

IV. Simulation And Results

This section details simulation setup and performance metrics. Finally, based on the performance metrics results are presented with analysis.

A. Simulation Setup

This section illustrates data generation for the simulation and configuration of the system to conduct the simulations.

1) Simulation And Data Generation: We generate our training and validation datasets using an open-source, python-based Multi-Agent Path Finding (MAPF) simulator [8]. The simulator employs a Conflict-Based Search (CBS) solver [12], [13] with an A^* based low-level planner. We collected data from three different warehouse environments with varying map sizes, defined by their number of navigable nodes (N = [168, 286, 583]). The number of pickup and goal locations for each environment are $[32, 64, 128]$ and $[8, 16, 32]$ respectively.

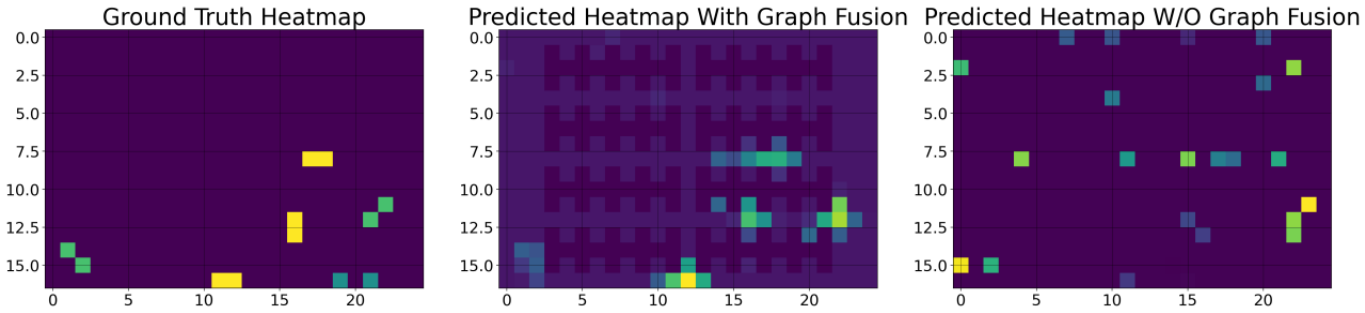

Fig. 3: Comparison of Results at $t = t_F = 10$. The left plot is the Ground truth labels for STRD on the grid based navigation map. The middle plot is Predicted heatmap after Graph Fusion and the third plot is prediction from RoboEnc plotted to nearest neighbor nodes of the NavGraph.

2) Task and Data Parameters: For each environment, we simulate robot fleets of increasing size (R = [8, 24, 52], respectively) executing pickup and delivery tasks. Data was collected from 10 random initializations per environment, with 80% used for training and 20% for validation.

3) Model Hyperparameters: Based on the methodology in Section IV, we set the sensing threshold for STRD calculation to $d_{strd} = 4$ units and the more restrictive edge-formation threshold for the *RoboGraph* to $d_{edge} = 2$ units. The model was trained using a look back history of $t_H = 20$ timesteps to predict a future horizon of $t_F = 10$ timesteps. The hidden dimension E_d is chosen as 64 for the overall analysis. For the static encoder, we choose a stack of GCNConv and linear layers with depth of 3 stacks. For dynamic encoders, we keep the $k-$hop property.

4) Training Details: The models were trained on a system with an Intel i9 14900HX CPU, 32 GB RAM, and an NVIDIA RTX 4090 GPU. We employed the two-phase training strategy described in Section III-e. The encoders were first pretrained for 10 epochs with a learning rate of 0.01. The full model was then trained end-to-end with a batch size of 1 and a learning rate of 0.0035, using the joint loss function (15).

5) Algorithm Testing: We assess the performance of our proposed Algorithm 1 via long-horizion task simulation. Here, we randomly sampled tasks and fed them into both (CBS with CWA^* and with A^*) the algorithms and measuring task completion. We run 5 simulations per environment (refer Section IV-A1) and averaged the results. Metric used for assessment is defined in Section IV-B. We set *Density Decay Factor* $\eta = 0.05$ and *Inflation Factor* $\lambda = 2.0$ for the subsequent simulation results.

B. Evaluation Metrics

To assess our model's performance, we use the following performance metrics.

1) Density Loss ($\mathcal{L}_C$): The mean square error (MSE) between the predicted continuous congestion value ($\tilde{\kappa}_n^C(t)$) and the ground-truth STRD (τ^N). This measures the accuracy of the predicted congestion magnitude.

2) Occupancy Loss ($\mathcal{L}_O$): The Binary Cross-Entropy (BCE) loss between the predicted binary occupancy ($\tilde{y}_n^O(t)$) and the ground-truth occupancy ($y_n^O(t)$). This measures the model's ability to correctly identify which nodes will be occupied.

3) Validation Loss: The total joint loss is computed on the held-out validation set by (15). This is our primary metric for model selection and to monitor for overfitting.

4) F1-Score: The F1-score for the binary occupancy classification task, providing a balanced measure of precision and recall.

To validate the resilience of CWA^* against saturation, we focus on metrics that capture system-wide throughput rather than individual agent speed.

5) Differential Throughput Gain: We define system capacity by the Cumulative Task Completion. Let the Cumulative Task Completion at time t for CBS with CWA* and CBS with A* be $N_{CWA^*}(t)$ and $N_{A^*}(t)$ respectively. To isolate the impact of congestion awareness, we analyse the Differential Throughput Gain $\Delta_{map}(t)$ expressed by (17).

$$\Delta_{map}(t) = N_{CWA^*}(t) - N_{A^*}(t), \tag{17}$$

where map = [small, medium, big] with $N = [168, 283, 583]$ respectively.

C. Results and Analysis

Simulation results analysis focuses on five key questions: (1) Which temporal graph encoder is most effective for the RoboEnc? (2) How critical is the GraphFusion module to the model's accuracy? (3) How well does the model's inference time scale with fleet and map size? (4) How sensitive is the model to its parameter count? (5) How CA-MAPF algorithm perform in small, medium, and big maps?

1) Ablation Study (RoboEncoder Architecture): To select the optimal temporal encoder for the RoboEnc module, we benchmarked four different TGNN architectures available in the PyTorch Geometric Temporal library [14]. The results are summarized in Table I

As shown in Table I, the DyGrEncoder [17] achieves the lowest overall Validation Loss (0.2808), indicating the best

TABLE I: Comparison of different temporal graph encoders for the RoboEnc module. Best performance in each column is in bold.

RoboEnc	$\mathcal{L}_{RE}$	**Validation Loss**	$\mathcal{L}_C$	$\mathcal{L}_O$
GConvGRU [9]	0.1371	0.3734	0.1704	0.0767
T-GCN [15]	0.1319	0.3174	0.1567	0.0847
GCLSTM[16]	**0.0971**	0.3129	**0.1396**	0.0878
DyGrEncoder[17]	0.1147	**0.2808**	0.1432	**0.0847**

generalization on unseen data. While GCLSTM [16] shows a strong ability to learn the congestion magnitude (lowest Density Loss), its higher validation loss suggests a tendency to overfit. Based on its superior generalization, we selected **DyGrEncoder** as the primary recurrent module for all subsequent experiments.

2) Qualitative Analysis (Efficacy of Graph Fusion): Fig. 3 provides a direct, qualitative validation of our core hypothesis: that the GraphFusion module is essential for grounding the decentralized, robot-centric predictions onto the global map. In Fig. 3, the Ground Truth heatmap (left) shows sparse, specific congestion hotspots. The Predicted Heatmap W/O Graph Fusion (right) represents an ablation where the learned *RoboEnc* states are plotted at the nearest map node. This model correctly identifies which robots will be congested but fails to localize the congestion, scattering its predictions. In stark contrast, our full model With Graph Fusion (center) successfully uses the cross-attention mechanism to project the learned STRD value onto the correct *NavGraph* nodes, resulting in a prediction that is both accurate and spatially precise.

3) Scalability Analysis (Inference Time): For any model to be viable in a real-world warehouse, its inference time must be low and must not grow exponentially with the fleet size. Fig. 4 plots the average inference time (ms) against an increasing number of robots $R = [8, 24, 52]$ for our three different maps with $N = [168, 286, 583]$.

We observe two critical results. First, the inference time scales linearly with the number of robots, with a modest slope of approximately **0.61 ms** per additional robot. Second, the inference speed is largely independent of the warehouse size, increasing by only **0.014 ms** per new node added to the NavGraph. This demonstrates the exceptional efficiency of our approach. The RoboGraph computations are localized (scaling with R), while the NavGraph computations are static and can be pre-computed, making the model highly scalable for large, real-world deployments.

4) Parameter Sensitivity Analysis: Finally, we analyzed the impact of model size (total parameters) by varying the hidden dimension size from 32 to 512. The result is shown in Table II.

We observe a clear "sweet spot" at **4.1M** parameters, which achieves the lowest validation loss and the highest F1-score. As seen in Table II, models smaller than this are under-parameterized. Crucially, the model with 18.3M parameters begins to overfit: its validation loss (blue line) increases while its training loss (orange line) continues to decrease. This

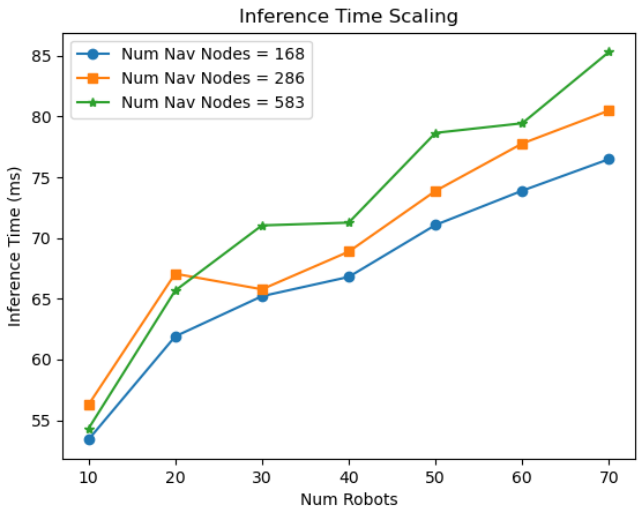

Fig. 4: Inference Time vs number of robots (Num robots) varying $\mathcal{G}_N$ nodes.

confirms that a model around 4.1M parameters provides the optimal balance between model capacity and generalization for this task.

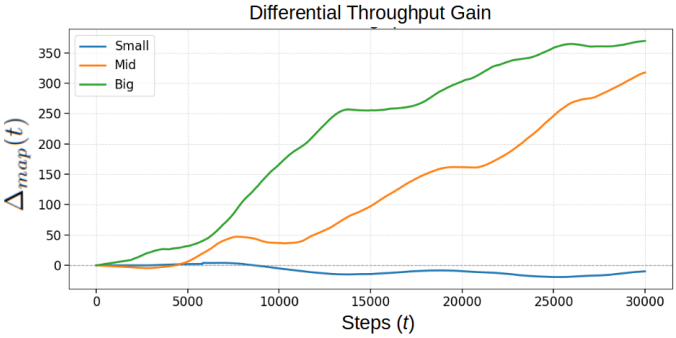

Fig. 5: Comparison of Differential Throughput Gain ($\Delta_{map}(t)$) of the proposed algorithm for 3 different maps with N = [168 (Small), 286 (Mid), 583 (Big)]. Positive elevation slope is indicative of increased Differential Throughput Gain over traditional approach. Results of 5 runs are averaged for 3 different maps.

5) Algorithm Performance Analysis: We observe a linearly growing $\Delta_{map}(t)$ for Medium [Blue] and Big [Orange] warehouses in Fig 5, which indicates a constant efficiency advantage. This is observed in Fig 5, where $\Delta_{map}(t)$ curve grows linearly as more tasks are added. This shows our method clearly provides throughput gain as new tasks are assigned. We believe that the growing divergence is the result of congestion awareness because of CWA^*. Our algorithm maintains a steeper completion rate by spatially distributing traffic, by proactively routing agents through underutilized areas, thereby delaying the onset of saturation at central corridors and drop zones. In contrast, the baseline system suffers from rapid

TABLE II: Impact of model parameter count on performance. The 4.1M parameter model provides the best trade-off.

Parameter Count	Train Loss↓	Validation Loss↓	Pretrain Loss↓	Robo Encoder Loss↓	Occupancy Loss↓	F1 Score↑
90K	0.112	0.379	0.138	0.103	0.174	0.42
300K	0.097	0.328	0.113	0.118	0.118	0.46
1.1M	0.071	0.296	**0.109**	0.097	**0.078**	0.55
4.1M	**0.063**	**0.291**	0.122	**0.094**	0.079	**0.59**
18.3M	0.069	0.304	0.118	0.099	0.096	0.58

saturation. As delays accumulate, agents remain in high-traffic zones for longer durations, which obstructs incoming agents and compounds the congestion. This recursive process adversely affects the system throughput. However, in smaller warehouses (see Fig 5) addition of congestion awareness has negligent to detrimental effects, wherein we suspect costmaps are inadvertedly causing confusion for the low level planner. A more thorough inspection is required for such map.

V. Conclusion and Future Direction

This paper presented a unified framework for predicting and mitigating global warehouse congestion using only decentralized, local sensing. By synthesizing agent interactions via a dynamic *RoboGraph* and map topology via a static *NavGraph*, our cross-attention *GraphFusion* module effectively projects sparse, robot-centric data onto dense global costmaps. Simulation results validate the architecture's efficiency, demonstrating that inference time scales linearly with fleet size while remaining largely independent of map dimensions, a critical requirement for large-scale real-world deployments. Furthermore, the integration of these predictive costmaps into our CA-MAPF algorithm confirmed the framework's practical utility; by closing the loop between forecasting and navigation, the system successfully steers agents through low-density zones, yielding measurable improvements in overall task throughput.

While the current validation relies on a Conflict-Based Search (CBS) implementation, future research will focus on testing the model's transferability to alternative solvers, such as Priority-Based Search or fully decentralized planners, to ensure robustness under varying traffic dynamics. Additionally, we aim to extend this predictive paradigm beyond the scope of path planning. Future work will investigate the integration of global congestion forecasts into higher-level warehouse orchestration, specifically for dynamic task scheduling and priority assignment, to further optimize system-wide efficiency and responsiveness.

References

[1] G. Yu and M. Wolf, "Congestion prediction for large fleets of mobile robots," 2023.

[2] C. Street, S. Pütz, M. Mühlig, N. Hawes, and B. Lacerda, "Congestion-aware policy synthesis for multirobot systems," *IEEE Transactions on Robotics*, vol. 38, no. 1, pp. 262–280, 2021.

[3] M. Zhang, R. Batta, and R. Nagi, "Modeling of workflow congestion and optimization of flow routing in a manufacturing/warehouse facility," *Management Science*, vol. 55, no. 2, pp. 267–280, 2009.

[4] S. AlHalawani and N. J. Mitra, "Congestion-aware warehouse flow analysis and optimization," in *Advances in Visual Computing*, G. Bebis, R. Boyle, B. Parvin, D. Koracin, I. Pavlidis, R. Feris, T. McGraw, M. Elendt, R. Kopper, E. Ragan, Z. Ye, and G. Weber, Eds. Cham: Springer International Publishing, 2015, pp. 702–711.

[5] Y. Li, R. Yu, C. Shahabi, and Y. Liu, "Diffusion convolutional recurrent neural network: Data-driven traffic forecasting," *arXiv preprint arXiv:1707.01926*, 2017.

[6] L. Bai, L. Yao, C. Li, X. Wang, and C. Wang, "Adaptive graph convolutional recurrent network for traffic forecasting," *Advances in neural information processing systems*, vol. 33, pp. 17 804–17 815, 2020.

[7] H. Liu, C. Zhu, D. Zhang, and Q. Li, "Attention-based spatial-temporal graph convolutional recurrent networks for traffic forecasting," in *International Conference on Advanced Data Mining and Applications*. Springer, 2023, pp. 630–645.

[8] Charles, "Multi-agent pickup and delivery," https://github.com/lodz97/multi-agent_pickup_and_delivery, 2021.

[9] Y. Seo, M. Defferrard, P. Vandergheynst, and X. Bresson, "Structured sequence modeling with graph convolutional recurrent networks," Springer, pp. 362–373, 2018.

[10] Y. Li, D. Tarlow, M. Brockschmidt, and R. Zemel, "Gated graph sequence neural networks," 2017. [Online]. Available: https://arxiv.org/abs/1511.05493

[11] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," 2023. [Online]. Available: https://arxiv.org/abs/1706.03762

[12] J. Kottinger, S. Almagor, and M. Lahijanian, "Conflict-based search for explainable multi-agent path finding," *Proceedings of the International Conference on Automated Planning and Scheduling*, vol. 32, p. 692–700, Jun. 2022. [Online]. Available: http://dx.doi.org/10.1609/icaps.v32i1.19859

[13] G. Sharon, R. Stern, A. Felner, and N. Sturtevant, "Conflict-Based search for optimal multi-agent path finding," *Proc. Conf. AAAI Artif. Intell.*, vol. 26, no. 1, pp. 563–569, Sep. 2021.

[14] B. Rozemberczki, P. Scherer, Y. He, G. Panagopoulos, M. S. Astefanoaei, O. Kiss, F. Béres, N. Collignon, and R. Sarkar, "Pytorch geometric temporal: Spatiotemporal signal processing with neural machine learning models," *CoRR*, vol. abs/2104.07788, 2021. [Online]. Available: https://arxiv.org/abs/2104.07788

[15] L. Zhao, Y. Song, C. Zhang, Y. Liu, P. Wang, T. Lin, M. Deng, and H. Li, "T-gcn: A temporal graph convolutional network for traffic prediction," *IEEE Transactions on Intelligent Transportation Systems*, vol. 21, no. 9, p. 3848–3858, Sep. 2020. [Online]. Available: http://dx.doi.org/10.1109/TITS.2019.2935152

[16] J. Chen, X. Wang, and X. Xu, "Gc-lstm: Graph convolution embedded lstm for dynamic link prediction," 2021. [Online]. Available: https://arxiv.org/abs/1812.04206

[17] A. Taheri and T. Berger-Wolf, "Predictive temporal embedding of dynamic graphs," in *2019 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM)*, 2019, pp. 57–64.