NS-RTW: Neuro-Symbolic Planning for Min-Max Routing with Time Windows

Jiaxian Li[1], Yifan Guo[2], Chen Wang[3], Zhongqiang Ren[1†]

***Abstract*—This paper investigates a min-max routing problem with time windows (RTW), which seeks paths for multiple agents starting from a single depot to visit a set of target locations, each within a designated time window, while minimizing the maximum completion time among the agents. Most existing research either considers the min-sum objective (minimizing the sum of completion times) or ignores the time window constraints. Recent work explores data-driven approaches, such as reinforcement learning (RL), to address this challenge. However, it still struggles to generalize to unseen scenarios outside the training set. To address these challenges, we propose a neuro-symbolic method called NS-RTW, which combines learning-based and classical search techniques within a bi-level optimization framework, enabling end-to-end self-supervision and offering better generalization than existing methods. On synthetic datasets, our method is able to find feasible solutions for all instances, outperforming the RL baseline, which fails in complex scenarios. Among all baseline methods, ours finds the best-quality solution in 87.1% of commonly solved instances. Experiments on Solomon benchmark and real-world case further demonstrate the generalization capability and practical applicability of our method.**

***Index Terms*—Vehicle Routing Problem, Machine Learning, Traveling Salesman Problem.**

I. Introduction

THIS paper considers a routing problem which seeks a set of paths for multiple agents to visit a set of static target locations while minimizing the maximum path completion time among the agents. Additionally, each target location is associated with a time window, within which that target must be serviced by an agent (Fig. 1). Similar problems are often referred to as multiple traveling salesman problems (MTSP) [1], vehicle routing problems [2], or agent routing with time windows [3]. We simply call it routing with time windows (RTW). RTW arises in many applications such as transportation logistics [4], fresh food delivery [5], and health care [6]. MTSP is an NP-hard problem [7] and so is RTW.

A variety of methods were developed for routing problems, ranging from exact algorithms [8]–[11] that guarantee completeness and solution optimality while being limited to small-scale problems, to heuristics [2], [12]–[14], learning-based algorithms [1], [15]–[21] and hybrid algorithms [22]–[26], which can handle large-scale problems with suboptimal solutions. However, most existing research either focuses on the min-sum objective [15], [16], [22]–[26] that minimizes the sum of path costs of the agents as opposed to the min-max objective, or does not consider time window constraints when addressing the min-max objective [17]–[21]. Compared with the min-sum objective, the min-max objective is computationally more challenging and often requires dedicated algorithmic designs [27], [28]. It is also relevant to many real-world applications, such as the computer networks [29]. When addressing the min-max objective, it is difficult to allocate targets to agents in a balanced way while simultaneously optimizing the visiting orders. Time windows impose additional constraints on the visiting time of each target, directly affecting path completion times and further complicating the problem. To our knowledge, the most relevant and recent approach in the literature is a reinforcement learning (RL) approach [1], which addresses both time-window constraints and the min-max objective. However, this method often lacks generalization capabilities: it may produce poor-quality solutions, or even infeasible solutions (with time window violation) for unseen test cases different from the training set.

To address these limitations, inspired by the recent imper-

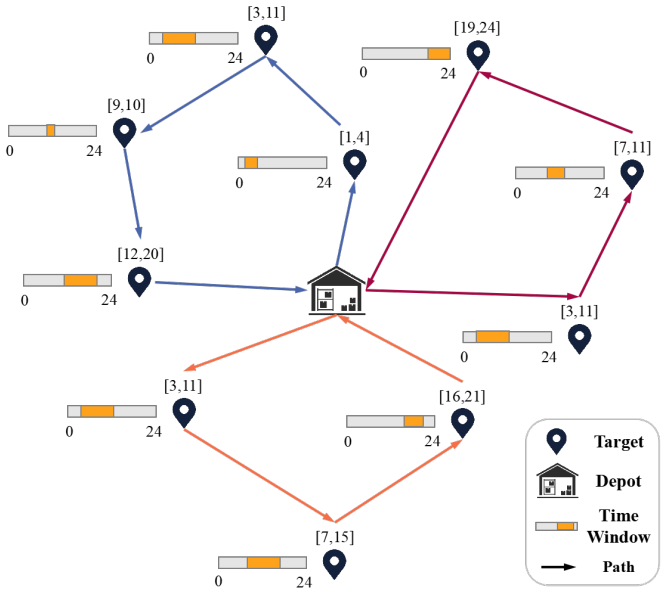

Fig. 1. Illustration of the routing problem with time windows (RTW), where a fleet of agents starts at a single depot, visits a set of targets and returns to the same depot. Each target must be visited exactly once within its required service time window. The horizontal bars adjacent to each node visualize the allowable time window (orange).

This work was supported by the Natural Science Foundation of China under Grant 62403313. (Corresponding author: Zhongqiang Ren. zhongqiang.ren@sjtu.edu.cn)

Jiaxian Li and Zhongqiang Ren are with Global College, Shanghai Jiao Tong University.

Yifan Guo is with School of Aeronautics and Astronautics, Purdue University, West Lafayette, IN 47907, USA.

Chen Wang is with Department of Computer Science and Engineering, University at Buffalo, NY 14260, USA.

ative learning [30], we propose a new hybrid neuro-symbolic (NeSy) approach NS-RTW that formulates the problem as a bi-level optimization and seeks to combine the advantages of learning and conventional search methods to find high-quality solutions for large-scale problems. NS-RTW comprises an upper-level optimization for training a target allocation network (i.e., the neural part) and a lower-level optimization that solves multiple single-agent traveling salesman problems (TSP) with time windows (TSP-TW) using classical search techniques (i.e., the symbolic part).

Specifically, the allocation network assigns each target to an agent, which decomposes the multi-agent routing into multiple simpler single-agent routing problems. Then, a classical single-agent TSP-TW solver, which can quickly produce a near-optimal path for a single-agent TSP-TW, is called to find a tour for each agent based on the allocation. Additionally, the upper-level optimization, namely the training of the allocation network, is supervised by the lower-level TSP-TW solver, which results in an end-to-end, self-supervised framework for solving RTW. In other words, NS-RTW is able to leverage learning to extract "global" features from the problem to intelligently decompose a large-scale multi-agent problem into multiple simpler single-agent ones, while using a classical solver for TSP-TW to satisfy the time window constraints. Furthermore, NS-RTW generalizes to the unseen datasets evaluated in our experiments without requiring extensive pretraining or labeled data.

There are two major technical difficulties when designing NS-RTW. The first one is to handle the time-window constraints, which impose discontinuous feasibility boundaries that are non-differentiable. We convert these constraints into soft penalty costs in the objective function when training the upper-level allocation network. These penalties are provided by the lower-level TSP-TW solver that can either rigorously respect the time window constraints or clearly report infeasibility, which provides informative guidance to the training of the upper-level allocation network. Furthermore, to effectively utilize this guidance and achieve constraint-aware allocation, we propose a graph-attention-based network with augmented spatio-temporal feature design. We also introduce a masking mechanism to explicitly prune temporally impossible transitions, thereby embedding physical constraints into the network to improve model performance.

The second challenge arises from the non-differentiable objective function when applying the NeSy method to the min-max problem. Furthermore, incorporating soft penalty terms for time windows into the cost function introduces high variance, which destabilizes the training process and hinders convergence. To address these, we integrate the REINFORCE algorithm [31] with a control variate [32] within our NeSy framework to reduce variance and ensure stable training.

To evaluate our approach, we compare it against several baseline methods, including the recent RL method [1] and classical solvers such as Google OR-Tools [33] and PyVRP [34]. We trained NS-RTW on data generated from a single training distribution and evaluated it on unseen datasets with varying target distributions and time-window lengths, the Solomon benchmark [2], and real-world scenarios derived from MAMUT [35].

On the synthetic datasets, our method consistently achieves a 100% success rate, whereas the RL baseline drops to near-zero success rates in the most challenging scenarios, highlighting the robustness of our approach. Furthermore, our method finds the best-quality solution in 87.1% of commonly solved instances while achieving a 1.5–4.8$\times$ speedup over OR-Tools, establishing a superior trade-off between solution quality and computational efficiency. Ablation studies confirm the efficacy of our neural guidance mechanism: compared to the standalone PyVRP [34], it improves solution quality by 7.30%–9.04% on average, and by up to 11.75%–13.25% when integrated with instance augmentation techniques [36]. Finally, evaluations on the Solomon benchmark and real-world cases further validate the generalization capability and practical applicability of the proposed method.

II. Related Work

A. Symbolic Routing Algorithms

Traveling Salesman Problems (TSPs) and Vehicle Routing Problems (VRPs) have a long history of research, with various approaches developed to address these challenges. Exact methods guarantee optimality, whereas approximation methods provide bounded solution-quality guarantees; both can face scalability limitations as the number of agents or targets increases [8]–[11], [37], [38]. Among them, integer linear programming [10], branch-price-and-cut algorithms [8], [9], and set-partitioning formulations [11] are several popular approaches for routing problems and their variants.

In contrast, heuristic methods often use local or population search to iteratively improve incumbent solutions; they sacrifice solution-quality guarantees but can find near-optimal solutions for large-scale problems [2], [13], [14], [39]. For example, tabu search [13] and variable neighborhood search [40] explore neighborhoods of the incumbent solution to identify improving moves [41]. Population-based approaches maintain and improve a set of solutions, including genetic algorithms [12], ant colony optimization [14], and particle swarm optimization [39]. Recent efforts also seek to mix multiple algorithms to either improve the algorithm performance or solve more complex problems [42], [43].

B. Learning-Based Routing Algorithms

Recently, there has been a notable shift towards employing deep learning techniques to address routing problems [1], [15]–[21], [44]. Such approaches either use supervised learning [44] or reinforcement learning (RL) [1], [15]–[21]. For supervised learning, the Pointer Network [44] learns approximate solutions from large datasets of solved problem instances to solve combinatorial optimization problems like TSP. However, supervised methods rely on costly, precomputed high-quality solution labels, which can hinder model performance on larger-scale problems. To overcome these issues, many recent studies use RL [1], [15]–[21], treating the routing problem as a Markov decision process and constructing a solution sequentially. RL learns a policy through trial and error, which thereby eliminates the need for labeled data. More recent

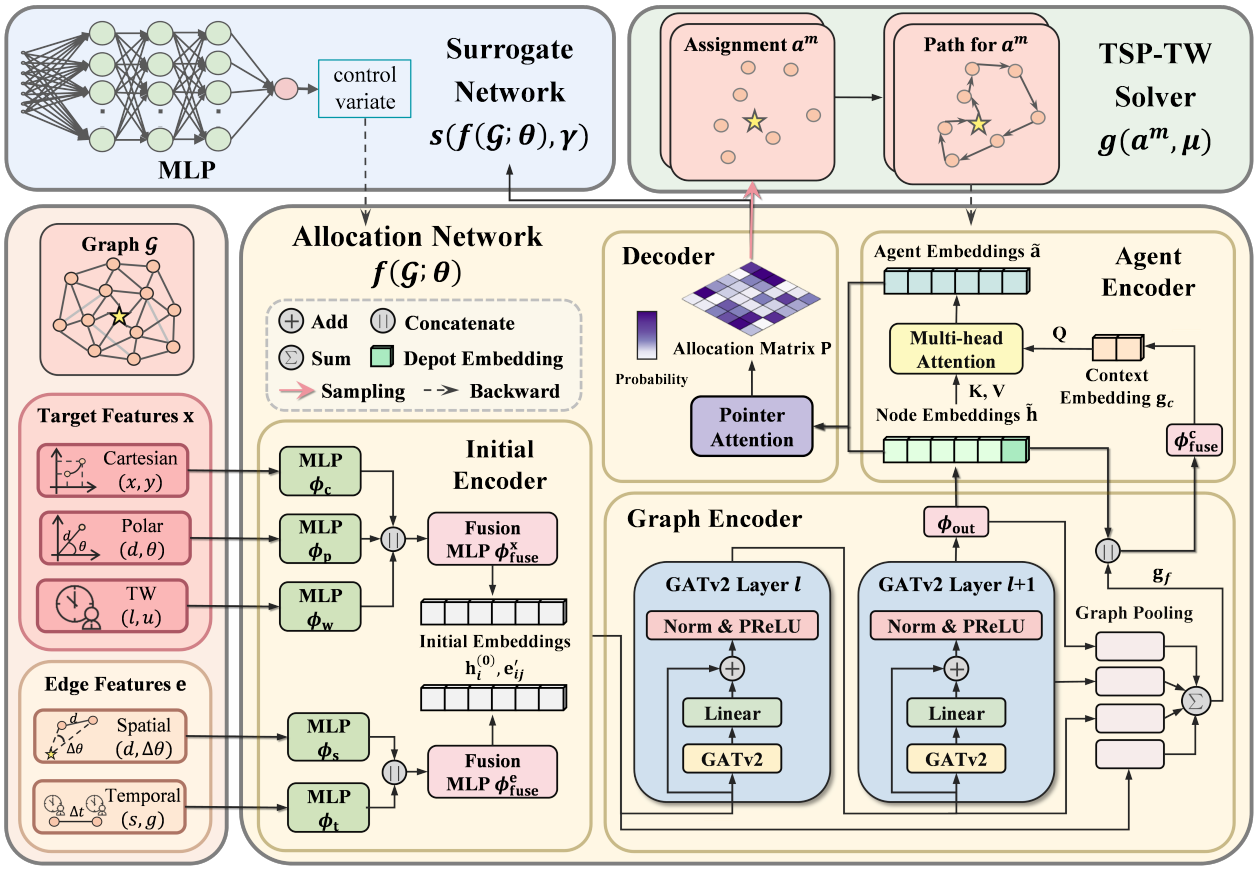

Fig. 2. Our Method. An allocation network takes as input the features of the graph and decomposes RTW into multiple TSP-TW subproblems by assigning targets to each agent. A TSP-TW solver receives the assignment and feeds back computation results, forming an end-to-end self-supervised framework. A surrogate network is leveraged here to produce the control variate for reducing training variance.

approaches further integrate RL with exact algorithms [22] or classic heuristics [23]–[26], which employ a learned model to iteratively select an improvement operator to refine solutions [22]–[24], or use RL to generate high-quality initial solutions for subsequent refinement using heuristics [25], [26]. However, these methods can still struggle to find high-quality solutions for large-scale problems and to generalize to unseen instances.

C. Neuro-Symbolic (NeSy) Approaches

Recent research seeks to combine the strengths of symbolic and learning methods in other areas such as robotics [30], vision and language tasks [45], [46], general artificial intelligence [47]. In general, a NeSy system is an amalgamation of both data-driven neural methods and symbolic methods that utilize formal logic and symbols for knowledge representation and rule-based reasoning. There are also nascent efforts to bring NeSy ideas into combinatorial optimization, such as the approach in [21], which studies the min-max MTSP, without time window constraints. Our work shares a similar bi-level optimization framework as in [21], yet differs from it not only by handling time window constraints, but also a new design of the network with completely different architecture and input features.

III. Problem Definition

Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ denote a fully connected directed graph. The node set $\mathcal{V} = \{v_0, v_1, \ldots, v_N, v_{N+1}\}$ contains a starting point v_0 (depot), an ending point v_{N+1} (a copy of v_0) and N target points $\{v_1, \ldots, v_N\}$. We use $N_g = N + 1$ to denote the number of nodes supplied to the model, comprising the depot and the N targets; the ending-depot copy v_{N+1} is used only for notation and is not counted in N_g. Each edge $(i, j) \in \mathcal{E}$ is assigned a travel cost d_{ij}, which is not required to be symmetric. In the synthetic problem setting, d_{ij} is the Euclidean travel time under constant unit velocity; in the road-network setting, d_{ij} is obtained from the corresponding asymmetric travel-cost matrix. Each node v_i is associated with a coordinate $\mathbf{c}_i = (x_i, y_i) \in \mathbb{R}^2$ in a bounded Euclidean space and a time window $[l_i, u_i]$. Here, l_i and u_i denote the lower bound and upper bound of the required service time window, respectively. Furthermore, each target v_i, $i \in \{1, \ldots, N\}$, must be visited exactly once, whereas the depot is shared by all agents. The time windows for both v_0 and v_{N+1} are defined as $[0, B]$, where B represents a sufficiently large planning horizon to ensure all agents can depart from the depot and return to the same depot.

Consider a group of agents $\mathcal{M} = \{m_1, m_2, ..., m_M\}$ with M representing the size of the group. Let π^m denote the path (i.e., an ordered sequence of nodes) assigned to agent m, and

let $v_i^m \in \pi^m$ denote the i-th target visited along this path. Let $t(v_i^m)$ denote the time when agent m visits target v_i and $t(\pi^m)$ denote the completion time of a path. By definition, since the path terminates at v_{N+1}, the completion time is given by $t(\pi^m) = t(v_{N+1}^m)$. The goal of the RTW is to find a path for each agent that satisfies the time-window constraints of all targets, while minimizing the maximum completion time among all paths, as shown in (1).

Problem 1 (RTW Problem)**.**

$$\min \max_{\pi^m, m\in\mathcal{M}} t(\pi^m), \tag{1a}$$

$$\textit{s.t.}\ \ l_i \le t(v_i^m) \le u_i, \quad \forall\, m \in \mathcal{M},\ v_i^m \in \pi^m \tag{1b}$$

$$\bigcup_{m\in\mathcal{M}} \pi^m = \mathcal{V} \tag{1c}$$

$$\pi^{m_1} \cap \pi^{m_2} = \{v_0, v_{N+1}\}, \forall m_1, m_2 \in \mathcal{M}, m_1 \neq m_2 \tag{1d}$$

Here, the last two constraints ensure that all targets are visited exactly once.

IV. Method

A. Reformulation

To decompose the multi-agent problem into multiple simpler single-agent subproblems, we reformulate the problem in (1) as a bi-level optimization problem, as shown in (2). This reformulation is similar to [21] and enables reciprocal learning between the neural network and the symbolic solver in a self supervised manner. Specifically, the upper-level optimization trains an allocation network to assign each target to an agent, while the lower-level optimization minimizes the routing cost for the given assignment of each agent.

Problem 2 (Bi-Level Optimization)**.**

$$\min_{\theta} U_\theta\left(H(\mu^*), f(\theta); \gamma\right); \quad \min_{\gamma} U_\gamma\left(H(\mu^*), s(\gamma); \theta\right) \tag{2a}$$

$$\textit{s.t.}\quad \mu^* = \arg\min_{\mu}\ H(\mu), \tag{2b}$$

$$H \doteq \max_{m\in\mathcal{M}}\ h(g(a^m;\mu)), \tag{2c}$$

$$a^m \sim f(\theta). \tag{2d}$$

Here, f denotes the allocation network with learnable parameters θ, and U_θ serves as the upper-level objective function corresponding to the allocation network. To reduce the high variance in gradient estimation (detailed in Section IV-E), we introduce a surrogate network $s(\cdot;\gamma)$ with learnable parameters γ, optimized via the cost U_γ. The term a^m represents the subset of targets assigned to agent m by f. The TSP-TW solver $g(\cdot;\mu)$, parameterized by μ, receives a^m and returns the optimized path cost together with feasibility and time-window-violation information to the allocation network. Finally, h represents a hybrid cost function that incorporates a soft penalty for time-window violations, providing a continuous violation signal for training while preserving explicit feasibility feedback from the lower-level solver. To quantify constraint violations, the solver computes the cumulative lateness along path π^m, denoted as $p^m = \sum_{v_i^m\in\pi^m} \max(0, \delta_i^m)$, where $\delta_i^m = t(v_i^m) - u_i$ represents the delay beyond the upper bound of the required service time window u_i. Note that early arrivals (i.e., $t(v_i^m) < l_i$) incur waiting time that is explicitly integrated into the path completion time $t(\pi^m)$. Consequently, we formulate the hybrid cost by combining $t(\pi^m)$ and p^m, as shown in (3).

$$h(\pi^m) = t(\pi^m) + \beta \times \frac{p^m}{t(\pi^m)}, \tag{3}$$

where $\beta \in \mathbb{R}^+$ is a hyperparameter for the penalty coefficient.

In summary, the upper-level objective optimizes the allocation and surrogate networks, while the lower-level objective minimizes the hybrid cost defined in (3). The maximum cost derived from the lower-level solver in (2c) acts as feedback, guiding the allocation network toward the min-max objective.

B. Augmented Feature Design

In RTW, each target has two key attributes: its spatial location and the temporal constraints imposed by its service time window. This differs from the min-max MTSP, which mainly focuses on balancing tasks based on location. RTW requires considering both space and time together. For example, MTSP typically visits geographically clustered targets consecutively to reduce travel distance. However, in RTW, these nearby targets might have different time windows. Prioritizing this spatial proximity alone can lead to excessive waiting times or infeasible solutions. Therefore, the allocation in RTW must shift from simple spatial clustering to capturing the complex spatio-temporal dependencies among targets.

To help the network learn such dependencies, which are difficult to infer from raw coordinates alone, we design explicit redundant features to augment the target and edge representations. For each target v_i, we combine its spatial and temporal attributes. The spatial attributes include both Cartesian coordinates $\mathbf{x}_i^{\mathrm{c}} = (x_i, y_i)$ and polar coordinates $\mathbf{x}_i^{\mathrm{p}} = (d_{0i}, \sin\theta_i, \cos\theta_i)$. The inclusion of polar coordinates helps the network learn the angular relationships of the targets relative to the depot [48]. The temporal attributes are defined by the service time window $\mathbf{x}_i^{\mathrm{w}} = (l_i, u_i)$. Thus, the augmented target feature $\mathbf{x}_i$ is defined in (4).

$$\mathbf{x}_i = (\mathbf{x}_i^{\mathrm{c}}, \mathbf{x}_i^{\mathrm{p}}, \mathbf{x}_i^{\mathrm{w}}). \tag{4}$$

Similarly, edge features $\mathbf{e}_{ij}$ comprise redundant spatial components $\mathbf{e}_{ij}^{\mathrm{s}} = (d_{ij}, \cos\Delta\theta_{ij})$ and temporal components $\mathbf{e}_{ij}^{\mathrm{t}} = (s_{ij}, g_{ij})$. Here, $\Delta\theta_{ij}$ denotes the angular difference between v_i and v_j. The temporal terms $s_{ij} = u_j - l_i$ and $g_{ij} = l_j - u_i$ quantify the maximum and minimum temporal gaps between the targets, respectively.

$$\mathbf{e}_{ij} = (\mathbf{e}_{ij}^{\mathrm{s}}, \mathbf{e}_{ij}^{\mathrm{t}}). \tag{5}$$

C. Masking Mechanism

To complement the feature augmentation, we introduce a masking mechanism to explicitly prune infeasible transitions. Specifically, we compute a binary adjacency mask $\mathcal{A}_{ij}$ that acts as a strict necessary condition for mathematical feasibility. The mask is defined as:

$$\mathcal{A}_{ij} = \begin{cases} 1, & \text{if } \max(l_i, d_{0i}) + d_{ij} \le u_j \\ 0, & \text{otherwise} \end{cases} \tag{6}$$

Here, the term $\max(l_i, d_{0i})$ defines the absolute earliest possible departure time from node i, representing a theoretical best-case scenario with zero prior accumulated waiting times. If adding the travel time d_{ij} to this earliest departure time exceeds the latest allowable arrival time u_j, reaching node j on time becomes unconditionally impossible, regardless of the broader routing context or previous node sequences. By masking these infeasible edges, constraints are explicitly integrated into the network.

D. Allocation Network

The allocation network is designed to properly assign targets to agents. Given that RTW is formulated as a graph, we construct the network using a Graph Neural Network backbone. For clarity, we refer to targets as "nodes" throughout this section to align with standard graph terminology.

As illustrated in Fig. 2, the network follows an encoder-decoder architecture, operating in a sequential manner comprising four key stages: 1) *Initial Encoder*, which first projects raw features in Section IV-B into a unified latent space, producing initial embeddings; 2) *Graph Encoder*, which further captures spatio-temporal dependencies among nodes, producing node and graph embeddings; 3) *Agent Encoder*, which derives specific embeddings for each agent based on the global graph context; and 4) *Assignment Decoder*, which computes the compatibility between nodes and agents to generate the final assignment. The details of each stage are presented below.

1) Initial Encoder: The first stage aims to prepare the raw features for the subsequent high-level graph feature extraction. Empirically, we observed that directly projecting the concatenated raw features in (4) and (5) is inefficient for feature extraction and often hinders the optimization process. A possible reason is that the spatial and temporal components have distinct physical meanings and distributions. To address this, we employ a multi-stream projection strategy that processes each component independently before mapping them into a unified latent space. Let D denote the dimension of the hidden embedding. We utilize three parallel multilayer perceptrons (MLPs), denoted as ϕ_{c}, ϕ_{p}, and ϕ_{w}, to process the Cartesian, polar, and temporal components in (4), respectively. Each MLP projects its corresponding input to an intermediate dimension of $D/2$. These intermediate representations are then concatenated and fused by a subsequent MLP $\phi^{\text{x}}_{\text{fuse}}$ to generate the initial embedding for node i, denoted by $\mathbf{h}_i^0 \in \mathbb{R}^D$:

$$\mathbf{h}_i^{(0)} = \phi^{\text{x}}_{\text{fuse}}\left(\left[\phi_{\text{c}}(\mathbf{x}_i^{\text{c}}) \,\middle\|\, \phi_{\text{p}}(\mathbf{x}_i^{\text{p}}) \,\middle\|\, \phi_{\text{w}}(\mathbf{x}_i^{\text{w}})\right]\right), \tag{7}$$

where $\|$ represents the concatenation operation.

Similarly, we employ two MLPs, ϕ_{s} and ϕ_{t}, to separately project the spatial and temporal edge components in (5) into $\mathbb{R}^{D/2}$. A fusion MLP $\psi^{\text{e}}_{\text{fuse}}$ then integrates these intermediate representations to form the edge embedding $\mathbf{e}'_{ij} \in \mathbb{R}^D$:

$$\mathbf{e}'_{ij} = \psi^{\text{e}}_{\text{fuse}}\left(\left[\phi_{\text{s}}(\mathbf{e}^{\text{s}}_{ij}) \,\middle\|\, \phi_{\text{t}}(\mathbf{e}^{\text{t}}_{ij})\right]\right). \tag{8}$$

2) Graph Encoder: With the initial embeddings established, the next phase focuses on further capturing the complex dependencies between nodes. Therefore, we adopt GATv2 [49], an extension of Graph Attention Networks [50] with dynamic attention, as the backbone.

Let $\mathbf{h}_i^{(l)} \in \mathbb{R}^D$ denote the input embedding corresponding to node i at the l-th GATv2 layer. In this work, we adopt the multi-head formulation of GATv2 with K heads. For the k-th head, the unnormalized attention score $b_{ij}^{(k)}$ between node i and its neighbor j is computed in (9).

$$\begin{aligned} b_{ij}^{(k)} = \mathbf{a}^{(k)\top}\text{LeakyReLU}\Big(&\mathbf{W}^{(k)}_{\text{src}}\,\mathbf{h}_i^{(l-1)} + \mathbf{W}^{(k)}_{\text{dst}}\,\mathbf{h}_j^{(l-1)} \\ &+ \mathbf{W}^{(k)}_{\text{edge}}\,\mathbf{e}'_{ij}\Big), \end{aligned} \tag{9}$$

where $\mathbf{W}^{(k)}_{\text{src}}, \mathbf{W}^{(k)}_{\text{dst}}, \mathbf{W}^{(k)}_{\text{edge}} \in \mathbb{R}^{D\times D}$ represent the learnable projection matrices for source nodes, destination nodes, and edges, respectively, and $\mathbf{a}^{(k)} \in \mathbb{R}^D$ denotes the attention weight vector. Then the normalized attention coefficient $\alpha_{ij}^{(k)}$ is derived by applying the softmax function:

$$\alpha_{ij}^{(k)} = \frac{\exp(b_{ij}^{(k)})}{\sum_{n\in\mathcal{N}(i)}\exp(b_{in}^{(k)})}, \tag{10}$$

where $\mathcal{N}(i) = \{j \mid \mathcal{A}_{ij} = 1\}$ denotes the set of neighbors retained by the masking mechanism for node i.

Subsequently, the feature aggregation for head k is calculated as $\mathbf{m}_i^{(k)} = \sum_{j\in\mathcal{N}(i)} \alpha_{ij}^{(k)}\mathbf{W}_{\text{v}}^{(k)}\mathbf{h}_j^{(l-1)}$, where $\mathbf{W}_v^{(k)} \in \mathbb{R}^{D\times D}$ is a linear projection matrix. To obtain the output of the l-th layer, the results from all K heads are concatenated, projected back to the original dimension D, and integrated via a residual connection, followed by Layer Normalization (LN) [51] and PReLU activation [52]:

$$\mathbf{h}_i^{(l)} = \text{PReLU}\left(\text{LN}\left(\mathbf{h}_i^{(l-1)} + \mathbf{W}_{\text{map}}\Big\|_{k=1}^{K}\mathbf{m}_i^{(k)}\right)\right), \tag{11}$$

where $\mathbf{W}_{\text{map}} \in \mathbb{R}^{D\times DK}$ is a linear projection that compresses the concatenated multi-head output. After stacking L layers, the final node embedding for node i is generated via an output MLP: $\widetilde{\mathbf{h}}_i = \phi_{\text{out}}(\mathbf{h}_i^{(L)})$, $\widetilde{\mathbf{h}}_i \in \mathbb{R}^D$.

At the end of this stage, we also derive a graph embedding $\mathbf{g}_f \in \mathbb{R}^D$ to summarize the global structural information of the input graph. This is achieved by summing the processed outputs from all GATv2 layers through a pooling block:

$$\mathbf{g}_f \;=\; \sum_{l=0}^{L+1}\text{D}_{\text{drop}}\big(\mathbf{W}^{(l)}_{\text{pool}}\,\text{Pool}\big(\mathbf{h}^{(l)}\big)\big), \tag{12}$$

where $\text{Pool}(\cdot)$ denotes the global mean pool operator, $\mathbf{W}^{(l)}_{\text{pool}} \in \mathbb{R}^{D\times D}$ denotes a linear projection, and $\text{D}_{\text{drop}}(\cdot)$ denotes the dropout function. Here, $\mathbf{h}^{(L+1)}$ denotes the final node embeddings $\widetilde{\mathbf{h}}$ produced by ϕ_{out}.

3) Agent Encoder: To generate agent embeddings, we first construct a context embedding $\mathbf{g}_c \in \mathbb{R}^{2D}$ by augmenting the global graph embedding $\mathbf{g}_f$ with the depot embedding $\widetilde{\mathbf{h}}_0$:

$$\mathbf{g}_{\text{c}} = \phi^{\text{c}}_{\text{fuse}}\Big(\mathbf{g}_f \,\Big\|\, \widetilde{\mathbf{h}}_0\Big), \tag{13}$$

where ϕ^{c}_{fuse} denotes the fusion MLP.

Next, we employ a Multi-Head Attention (MHA) mechanism [53] to derive a unique embedding for each agent. The scaled dot-product attention is defined in (14).

$$\text{Attn}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V, \tag{14}$$

where d_k is the dimension of the key and query vectors. MHA aggregates information from all S heads:

$$\text{MHA}(Q, K, V) = \Big(\Big\Vert_{i=1}^{S} head_i \Big) \mathbf{W}^{\text{O}}, \tag{15}$$

where $head_i = \text{Attn}(Q\mathbf{W}_i^{\text{Q}}, K\mathbf{W}_i^{\text{K}}, V\mathbf{W}_i^{\text{V}})$. $\mathbf{W}^{\text{O}} \in \mathbb{R}^{SD_v \times 2D}$, $\mathbf{W}_i^{\text{Q}} \in \mathbb{R}^{2D \times D_k}$, $\mathbf{W}_i^{\text{K}} \in \mathbb{R}^{2D \times D_k}$, and $\mathbf{W}_i^{\text{V}} \in \mathbb{R}^{2D \times D_v}$ are the projection matrices for head i. $D_v = D_k = \lfloor 2D/S \rfloor$. The context embedding $\mathbf{g}_c$ serves as the query Q, while the node embeddings $\widetilde{\mathbf{h}}$ serve as both keys K and values V. Consequently, the embedding for agent m is computed as:

$$\widetilde{\mathbf{a}}_m = \text{MHA}_m(\mathbf{g}_c, \widetilde{\mathbf{h}}, \widetilde{\mathbf{h}}). \tag{16}$$

Here, each agent m uses an independent set of learnable parameters, which breaks symmetry across agents, allowing each MHA module to attend to different aspects of the shared context and produce a unique embedding $\widetilde{\mathbf{a}}_m$.

4) Assignment Decoder: In the final stage, the network determines the assignment policy based on the learned representations of agents ($\widetilde{\mathbf{a}} \in \mathbb{R}^{M \times 2D}$) and nodes ($\widetilde{\mathbf{h}} \in \mathbb{R}^{N_g \times D}$). Unlike the previous stages that focus on feature fusion, this phase aims to establish a direct probabilistic mapping. To this end, we adopt a mechanism in Pointer Networks [44]. Rather than fusing vectors to generate new embeddings, this approach explicitly calculates the compatibility scores between target nodes and agents, utilizing these scores as pointers to determine assignments [16]. The mechanism is mathematically formalized as follows:

$$K' = \widetilde{\mathbf{a}}\mathbf{W}_{\text{a}}, \quad Q' = \widetilde{\mathbf{h}}_{\text{d}}\mathbf{W}_{\text{n}}, \tag{17a}$$

$$P = \text{softmax}_{m \in \mathcal{M}} \left(\eta \cdot \tanh \left(\frac{Q' K'^{\text{T}}}{\sqrt{D}} \right) \right), \tag{17b}$$

where $\widetilde{\mathbf{h}}_{\text{d}} \in \mathbb{R}^{(N_g - 1) \times D} = \mathbb{R}^{N \times D}$ denotes the node embeddings excluding the depot. $\mathbf{W}_{\text{a}} \in \mathbb{R}^{2D \times D}$ and $\mathbf{W}_{\text{n}} \in \mathbb{R}^{D \times D}$ are projection matrices. η is a scaling factor that controls the range of the logits. The output is a probability matrix $P \in \mathbb{R}^{N \times M}$, which dictates the allocation of targets to agents. Specifically, each element p_{ij} represents the probability that target i is assigned to agent j. The final allocation is generated by sampling from the distribution defined by each row of P.

Upon determining the allocation, the original RTW is decomposed into multiple independent TSP-TW subproblems. Subsequently, while any solver for TSP-TW is applicable, we invoke PyVRP [34], a state-of-the-art solver based on a hybrid genetic search algorithm, to solve these subproblems in parallel. Finally, the maximum hybrid cost in (3) derived from these solutions is returned to the upper-level network to guide the optimization process.

E. Surrogate Network

To mitigate high gradient variance in the allocation network during training, we incorporate a surrogate network to act as a control variate [32]. Control variates are a widely established technique in machine learning for variance reduction; for instance, they serve as the theoretical foundation for baseline-based policy gradients and actor-critic methods [54]. The fundamental principle is formalized in the following lemma. To make the paper self-contained, we also provide a proof, which follows similar ideas in [21].

Lemma 1. *Let C be a random variable that is correlated with the random variable X. Further assume C has an expectation $\mathbb{E}[C]$, and let $b \in \mathbb{R}$ be a constant. Define a new random variable Y by:*

$$Y = X + b\big(C - \mathbb{E}[C]\big). \tag{18}$$

When the constant b is properly chosen and C is correlated with X, we have $\mathbb{E}[Y] = \mathbb{E}[X]$ and $\text{Var}(Y) \leq \text{Var}(X)$.

Proof. We first prove random variable X and Y have the same expectation, namely $\mathbb{E}[X] = \mathbb{E}[Y]$. As $\mathbb{E}[Y] = \mathbb{E}[X + b\big(C - \mathbb{E}[C]\big)]$, we can obtain that:

$$\mathbb{E}[Y] = \mathbb{E}[X] + b\big(\mathbb{E}[C] - \mathbb{E}[C]\big) = \mathbb{E}[X]. \tag{19}$$

We next prove that the new variable Y has no larger variance as X, namely $\text{Var}(Y) \leq \text{Var}(X)$. The new variance of Y can be written as:

$$\sigma_Y^2 = \sigma_X^2 + b^2\sigma_C^2 + 2b\,\sigma_{X,C}, \tag{20}$$

where we denote $\sigma_Y^2 = \text{Var}(Y)$, $\sigma_X^2 = \text{Var}(X)$, $\sigma_C^2 = \text{Var}(C)$, and $\sigma_{X,C} = \text{Cov}(X, C)$. Treat (20) as a function of b and obtain its minimum by setting its derivative to zero, which yields the optimal coefficient b^*:

$$b^* = -\frac{\sigma_{X,C}}{\sigma_C^2}. \tag{21}$$

Then we have:

$$\text{Var}(Y)\big|_{b=b^*} = \sigma_X^2\big(1 - \rho_{X,C}^2\big), \tag{22}$$

where $\rho_{X,C} = \dfrac{\sigma_{X,C}}{\sigma_X \sigma_C}$ is the correlation coefficient between X and C. Since $\rho_{X,C}^2 \in [0, 1]$, the minimal variance $\text{Var}(Y)\big|_{b=b^*}$ is always no greater than $\text{Var}(X)$. Thus, when b is chosen optimally, the constructed estimator has reduced variance. □

The surrogate network accepts the probability matrix P as input and operates in parallel with the TSP-TW solver. It predicts a baseline value $H' = s(P(\theta); \gamma)$ from the allocation probability representation and uses this value as the control variate for the objective defined in (2c).

The primary goal of the surrogate network is to minimize the gradient variance of the allocation network. However, a naive optimization strategy that merely imitates the lower-level solver's input-output mapping is ineffective, as minimizing prediction error does not guarantee the capture of useful gradient information. Consequently, we define the surrogate's

objective function U_γ to explicitly minimize the variance of the allocation network's gradient:

$$U_\gamma = \mathrm{Var}\left(\frac{\partial U}{\partial \theta}\right). \tag{23}$$

As $\mathrm{Var}\left(\frac{\partial U}{\partial \theta}\right)$ can be written in expectation form, $\mathrm{Var}\left(\frac{\partial U}{\partial \theta}\right) = \mathbb{E}[\left(\frac{\partial U}{\partial \theta}\right)^2] - \mathbb{E}[\left(\frac{\partial U}{\partial \theta}\right)]^2$, its derivative can be written as:

$$\frac{\partial U_\gamma}{\partial \gamma} = \frac{\partial \mathbb{E}\left[\left(\frac{\partial U}{\partial \theta}\right)^2\right]}{\partial \gamma} - \frac{\partial \mathbb{E}\left[\left(\frac{\partial U}{\partial \theta}\right)\right]^2}{\partial \gamma} = \frac{\partial \mathbb{E}\left[\left(\frac{\partial U}{\partial \theta}\right)^2\right]}{\partial \gamma}. \tag{24}$$

This formulation ensures that the surrogate minimizes variance without introducing bias, thereby facilitating stable and efficient learning.

F. Optimization

The upper-level objective U_θ is to minimize the maximum cost H defined in (2). However, direct gradient descent is still infeasible because the overall bi-level objective depends on discrete target assignments, the non-differentiable max operator, and the lower-level solver. To address this, we employ the REINFORCE algorithm [31] to estimate the gradients. The resulting single-sample gradient estimator is given by:

$$\nabla_\theta H = H \nabla_\theta \log P(\theta). \tag{25}$$

Consistent with Section IV-E, we utilize the control variate to stabilize the gradient estimates, addressing the variance introduced by REINFORCE and the soft penalty formulation. Specifically, we map the terms from (18) to our context: the variable X corresponds to the raw gradient estimator in (25), and the control variable C is defined as the surrogate-based estimator $H' \nabla_\theta \log P(\theta)$. Applying the log-derivative trick, the expectation of the control variable simplifies to $\mathbb{E}[C] = \nabla_\theta \mathbb{E}[H']$. b is set to be -1 following [21]. Consequently, the final variance-reduced gradient estimator for the allocation network is formulated as:

$$\nabla_\theta H_{\text{new}} = [H(\mu^*) - H'] \nabla_\theta \log P(\theta) + \nabla_\theta s(P(\theta); \gamma). \tag{26}$$

V. Experimental Results

We evaluate the proposed NS-RTW on synthetic datasets with varying scales and distributions, comparing it against the RL approach and classical solvers. We analyze performance metrics including success rate, solution quality, and computational efficiency, with a specific focus on robustness under variable time windows. We present ablation studies to validate the specific contributions of our neuro-symbolic design and several key components within our framework. Finally, we evaluate all methods on the widely used Solomon benchmark [2] and real-world cases derived from MAMUT [35].

A. Settings

1) Data Generation: We generate synthetic datasets across various problem scales, target distributions, and time window lengths. All target coordinates are normalized to the unit square $[0, 1]^2$, with the depot fixed at the center $(0.5, 0.5)$.

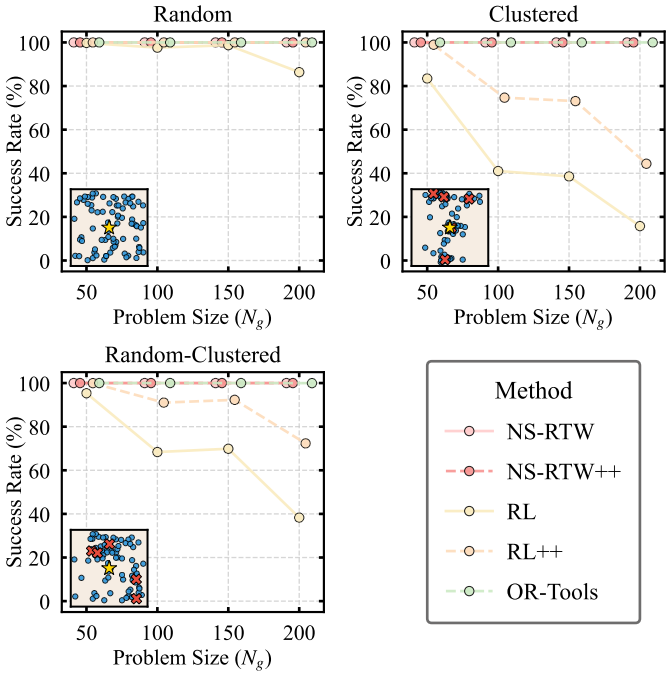

Fig. 3. Success rate comparison across varying distributions and scales. Our NS-RTW achieves robust performance comparable to OR-Tools, maintaining a 100% success rate across all synthetic scenarios. In contrast, the RL baseline fails to generalize to clustered and random-clustered scenarios and exhibits significant degradation as problem size N_g increases.

Regarding time windows, we follow the protocol established in [1]: $l_i \sim \mathcal{U}(0, 3)$ for $i = 1, \ldots, N$, and $u_i = l_i + \Delta l$, where Δl represents the time window length and $\mathcal{U}(\cdot)$ denotes the uniform distribution.

We generate target locations using three target distributions commonly used in VRP benchmarks [2], [55], [56]: (i) Random, (ii) Clustered, and (iii) Random-Clustered. Specifically, the Clustered and Random-Clustered variants follow the convention in [55]. We include those clustered variants as they closely resemble real-world topologies like urban agglomerations [56]. We also include the Solomon benchmark [2] and real world cases derived from MAMUT [35] to evaluate the transfer performance of our approach.

2) Experimental setup: During training, we generate instances on-the-fly with problem size $N_g = 50$ (one depot and 49 targets), using a random target distribution and 5 agents. The time window length Δl is fixed as 3. To comprehensively evaluate scalability and generalization, we construct test sets with varying problem sizes ($N_g \in \{50, 100, 150, 200\}$) across all three distribution types defined in Section V-A1. Each test configuration consists of 3,000 instances. Furthermore, to verify robustness, we conduct additional experiments on datasets with variable time window lengths. Note that during testing, time windows are enforced as hard constraints. Any violation renders the solution infeasible.

All experiments were conducted on a server equipped with an Intel Xeon Gold 6448Y CPU and a single NVIDIA A100 (80GB) GPU. The model was implemented in Python 3.9.12 using the PyTorch 2.1.2 framework. Detailed hyperparameter settings are provided in Appendix A2.

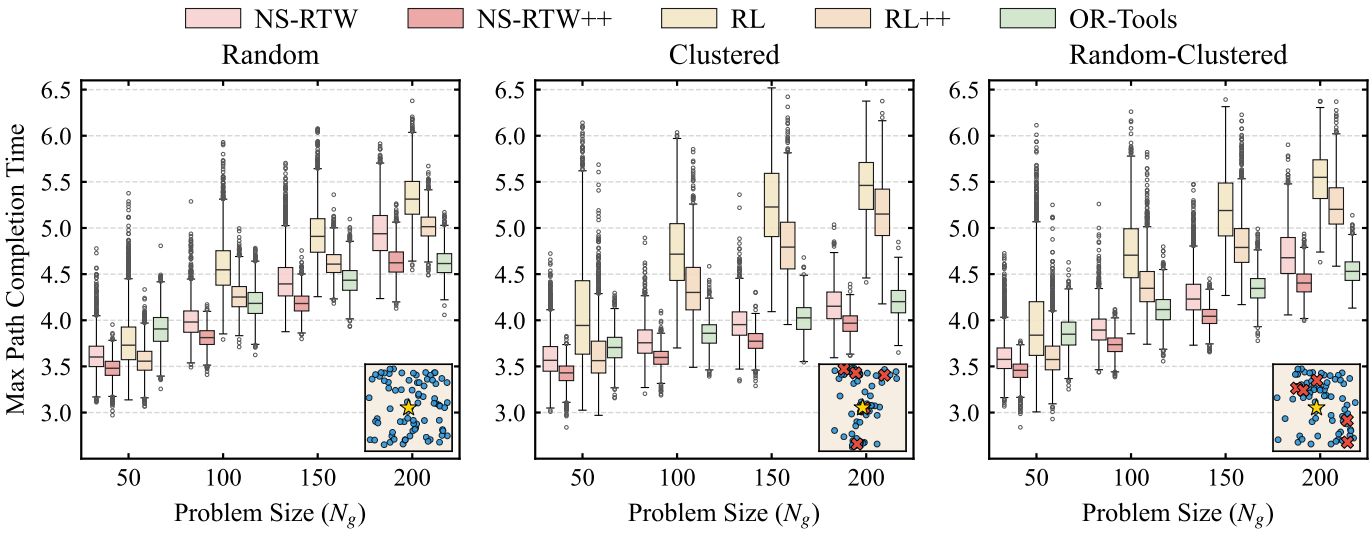

Fig. 4. Statistical comparison of solution quality in terms of maximum path completion time across Random, Clustered, and Random-Clustered distributions. Results are reported for problem sizes $N_g \in \{50, 100, 150, 200\}$. Statistics are computed on the subset of instances successfully solved by all compared methods. Our NS-RTW++ achieves the lowest median value in nearly all evaluated settings and remains comparable to the best-performing baseline in the remaining setting, while maintaining tight interquartile ranges.

3) Baselines: To evaluate the effectiveness and efficiency of our proposed framework, we select three representative state-of-the-art baselines for comparison. These benchmarks encompass both a recent data-driven reinforcement learning approach and classical solvers, providing a comprehensive assessment of our method's performance across different paradigms.

- **RL** [1]: This approach employs a dual-learning framework that similarly decomposes the RTW into multiple subproblems. It involves a two-stage process: first, pre-training a "worker agent" to solve single-vehicle routing tasks, allowing it to reject infeasible targets with a penalty, and subsequently training a "manager agent" to assign targets based on worker feedback. We select this baseline to demonstrate that our framework achieves superior performance without the complexity of multi-stage pre-training. Due to the unavailability of a pre-trained model matching our exact problem size ($N_g = 50$), we utilize their official model trained on the nearest available configuration, $N_g = 100$ with 5 agents, which is reported to have better scalability than models trained with $N_g = 50$.
- **OR-Tools** [33]: Developed by Google, OR-Tools is a widely adopted open-source suite for combinatorial optimization that provides optimal or near-optimal solutions for various VRP variants. We select OR-Tools as the representative of classical search-based approaches to verify that our method can generate competitive solutions while offering significantly higher computational efficiency.
- **PyVRP** [34]: As an emerging state-of-the-art VRP library, PyVRP achieves outstanding performance by combining genetic algorithms with local search heuristics. While we utilize PyVRP as the embedded lower-level solver in our framework, we also include it here as a standalone baseline to conduct ablation studies (see Section V-C), allowing us to quantify the specific contribution of our neural allocation mechanism.

TABLE I
EIGHT COORDINATE TRANSFORMATIONS USED FOR INFERENCE-TIME INSTANCE AUGMENTATION.

$f(x, y)$	
(x, y)	(y, x)
$(x, 1-y)$	$(y, 1-x)$
$(1-x, y)$	$(1-y, x)$
$(1-x, 1-y)$	$(1-y, 1-x)$

4) Instance Augmentation: To further enhance solution quality, we implement an inference-time instance augmentation strategy for neural solvers called the $\times 8$ augmentation proposed in [36]. This approach leverages the geometric symmetries inherent in routing problems, where the optimal objective value remains invariant under transformations such as rotation and reflection. Specifically, for each test instance, we generate eight variations via coordinate transformations (detailed in Table I). The final solution is derived by selecting the minimum cost among these eight results. To ensure a fair comparison, we apply this augmentation to both our proposed model and the RL baseline. We refer to our method with augmentation and the RL baseline with augmentation as NS-RTW++ and RL++, respectively, in the rest of this paper.

B. Evaluation Results

1) Success Rate: As shown in Fig. 3, both NS-RTW and OR-Tools achieve a 100% success rate across all tested instances. In contrast, the performance of the RL baseline drops as the problem scale increases. The RL baseline is also very sensitive to changes in the target distribution. Its success rate falls in clustered and random-clustered scenarios, even when we add augmentation. On the other hand, NS-RTW solves every instance, regardless of changes in scale or distribution.

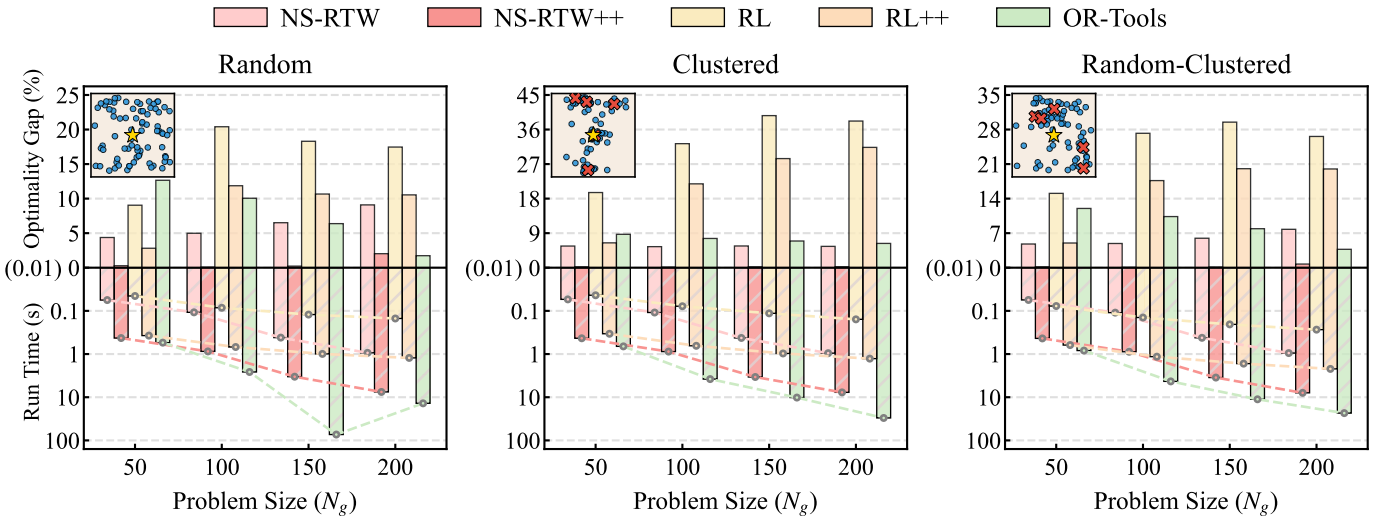

Fig. 5. Trade-off analysis between solution quality and computational efficiency across random, clustered, and random-clustered distributions. The upper section displays the optimality gap (%), which is defined as the percentage deviation of a method's solution from the best-found solution among all comparative methods. Smaller values indicate higher solution quality. The lower section shows the runtime (seconds) on a logarithmic scale. The runtime axis is inverted for the combined layout; lower runtime values remain better. Results are reported for problem sizes $N_g \in \{50, 100, 150, 200\}$. Overall, our NS-RTW demonstrates a superior trade-off, achieving high solution quality while maintaining computational efficiency.

Notably, NS-RTW outperforms the RL baseline by up to 84.24 percentage points in clustered scenarios and 61.70 percentage points in random-clustered scenarios. These results confirm that NS-RTW scales well and generalizes better to both seen and unseen distributions.

2) Solution Quality and Computational Efficiency: As the RL baseline cannot solve all instances, we focus on the subset of instances that are successfully solved by all baseline methods to ensure a fair comparison. Fig. 4 shows the maximum path completion times across different problem scales and distributions. Our NS-RTW++ consistently achieves the lowest median value in nearly all evaluated settings, while maintaining tight interquartile ranges, indicating superior stability and performance in all scenarios. Even without augmentation, NS-RTW can still achieve the lowest median value for $N_g < 150$. When compared to the RL baseline, NS-RTW reduces the median completion time by 8.49%, 19.57%, and 14.68% in random, clustered, and random-clustered scenarios, respectively. With augmentation, NS-RTW++ reduces these times by 7.50%, 16.18%, and 12.25% compared to RL++. Finally, compared to OR-Tools, NS-RTW++ improves performance by 6.44%, 6.62%, and 7.49% in the same test settings.

To further quantify the trade-off between computational efficiency and solution quality, we define the optimality gap G as follows:

$$G = \frac{P_m - P_b}{P_b} \times 100\%, \tag{27}$$

where P_m denotes the maximum path completion time obtained by the tested method, and P_b represents the best-found solution among all comparative methods for the given instance.

Fig. 5 visualizes the results. The top half of the chart displays the optimality gap, while the bottom half shows the runtime on a log scale. In terms of optimality, NS-RTW++ maintains the lowest average gaps of 0.54%, 0.13%, and 0.18% for random, clustered, and random-clustered distributions, respectively. Furthermore, it finds the best-quality solution in

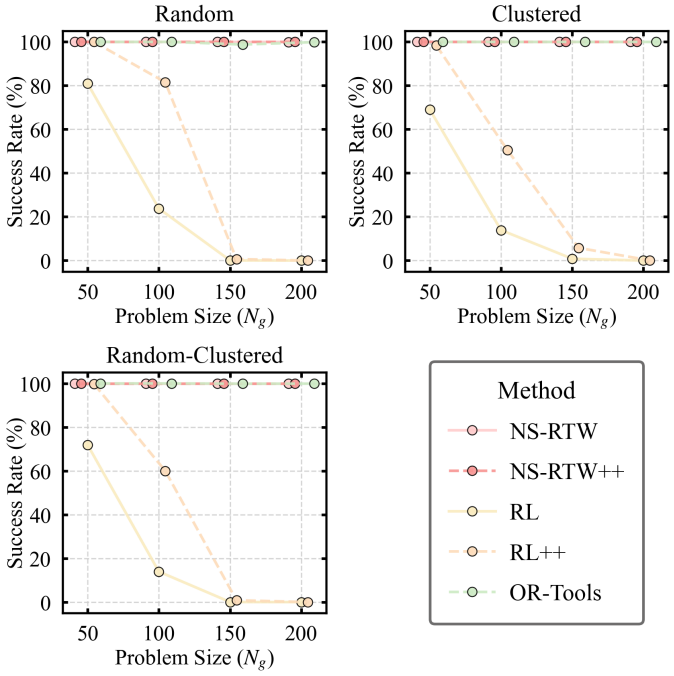

Fig. 6. Success rate comparison across varying distributions and scales with variable time window lengths. Our NS-RTW consistently achieves a 100% success rate across all scenarios. In contrast, the RL baselines struggle to satisfy the constraints as time windows become tighter.

80.1%, 91.8%, and 89.6% of instances for each distribution, peaking at 99.4% in specific cases(i.e., the random-clustered distribution at $N_g = 100$). Overall, NS-RTW++ achieves an average best-solution rate of 87.1% across the three synthetic target distributions on commonly solved instances. By comparison, the RL and RL++ baselines struggle, showing much larger gaps that range from roughly 9% to 32%. Notably, while NS-RTW achieves smaller optimality gaps on clustered

TABLE II
ABLATION RESULTS FOR NS-RTW. "BASELINE" USES THE BASELINE METHOD IN SECTION IV-E, "NOMASK" REMOVES THE MASKING MECHANISM IN SECTION IV-C, AND "NOAUG" REMOVES THE AUGMENTED FEATURE DESIGN IN SECTION IV-B. ALL VARIANTS USE THE TRAINING SETTINGS IN APPENDIX A2; SCENARIOS FOLLOW SECTION V-A1 WITH FIXED TIME WINDOWS AND A CENTERED DEPOT. INFEASIBLE SOLUTIONS RECEIVE A COST OF 20. GAP IS DEFINED IN (27) RELATIVE TO THE BEST VALUE AMONG ALL METHODS, AND N_g DENOTES THE NUMBER OF INPUT NODES, INCLUDING THE DEPOT. THE LOWEST GAP AND PENALIZED AVERAGE COST ARE HIGHLIGHTED IN BOLD.

Distribution	Method	$N_g = 50$		$N_g = 100$		$N_g = 150$		$N_g = 200$	
		Cost	Gap	Cost	Gap	Cost	Gap	Cost	Gap
Random	NS-RTW	**3.623**	**0.00%**	**4.000**	**0.00%**	**4.449**	**0.00%**	**4.962**	**0.00%**
	Baseline	3.684	1.70%	4.171	4.28%	4.666	4.87%	5.132	3.43%
	NoMask	3.820	5.45%	4.190	4.76%	4.492	0.96%	5.170	4.18%
	NoAug	3.663	1.11%	4.048	1.22%	4.493	0.99%	4.986	0.48%
Clustered	NS-RTW	**3.624**	**0.00%**	**3.841**	**0.00%**	**4.038**	**0.00%**	**4.233**	**0.00%**
	Baseline	3.699	2.08%	3.958	3.04%	4.230	4.75%	4.605	8.80%
	NoMask	3.675	1.42%	3.929	2.30%	4.143	2.58%	4.390	3.71%
	NoAug	3.666	1.16%	3.889	1.27%	4.111	1.79%	4.315	1.95%
Random-Clustered	NS-RTW	**3.611**	**0.00%**	**3.912**	**0.00%**	**4.278**	**0.00%**	**4.705**	**0.00%**
	Baseline	3.691	2.23%	4.060	3.77%	4.521	5.68%	4.930	4.79%
	NoMask	3.778	4.62%	4.105	4.93%	4.385	2.50%	4.865	3.40%
	NoAug	3.665	1.49%	3.973	1.56%	4.334	1.31%	4.753	1.01%

instances than on random ones, the RL baselines exhibit the opposite trend. This further demonstrates that the RL baseline overfits to the uniform data patterns seen during its training, struggling with unseen clustered topologies. In contrast, our NS-RTW utilizes an end-to-end NeSy framework. Clustered topologies naturally form spatial task groups, which may simplify the high-level target allocation to some extent. By embedding a symbolic solver that performs rigorous rule-based reasoning, our method not only generalizes well to unseen distributions but explicitly exploits the data structure to find better routing plans Finally, OR-Tools performs better than the RL methods but still falls short of our approach, with an average gap of around 7–8%.

Regarding computational efficiency, the RL baseline is the fastest, while OR-Tools is the slowest. Our methods, NS-RTW and NS-RTW++, fall in between. Specifically, NS-RTW requires 1.2–5.9 times the runtime of the RL baseline. However, since the runtime of the RL baseline is extremely short (less than 1s), this slight increase is acceptable for practical use. Compared with OR-Tools, NS-RTW is 9.7–38.7 times faster and NS-RTW++ remains 1.5–4.8 times faster.

In summary, although RL offers high inference speed, it suffers from poor solution quality and even fails to generate feasible solutions when target distribution shifts. Conversely, OR-Tools provides high-quality solutions but lacks efficiency. Our approach effectively balances this trade-off. This validates our expectation for the NeSy framework: it successfully combines the fast inference of learning-based methods with the high solution quality of classical search methods, compensating for the limitations of each.

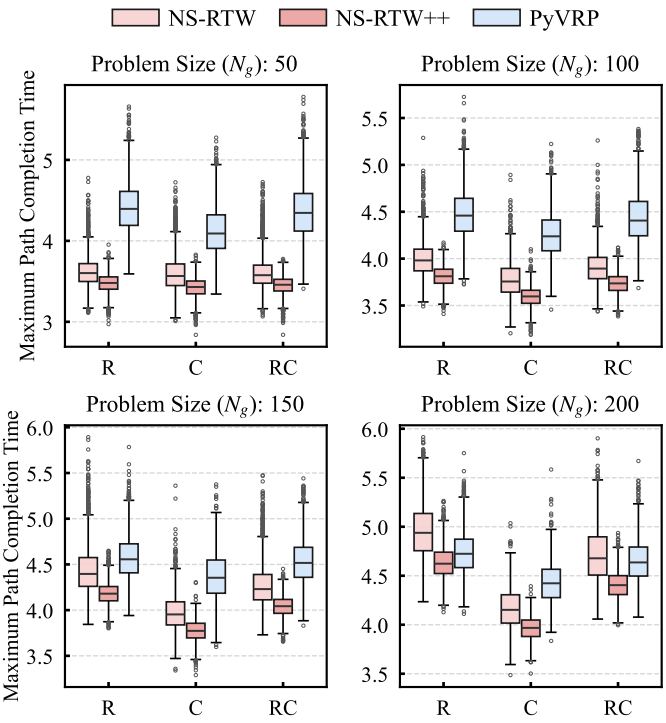

Fig. 7. Ablation study comparing our NS-RTW against the standalone lower-level solver PyVRP, where random, clustered, random-clustered are abbreviated as R, C and RC, respectively. Across all problem sizes and scenarios, NS-RTW++ consistently achieves lower median time with reduced variance, demonstrating the effectiveness of our proposed framework in optimizing the search process for the low-level solver.

3) Robustness to Variable Time Windows: In real-world scenarios, the length of each target's time window is heterogeneous due to factors such as the variations in delivery difficulty across regions, or the differences in targets' tolerance for waiting time. Therefore, we conduct extended experiments to evaluate our model's robustness to variable time windows. Specifically, we change Δl from fixed 3 to any value that is uniformly sampled within $[2, 4]$. Fig. 6 shows the results of success rate. RL shows great sensitivity to changes in time-window lengths, resulting in an obvious drop in success rate when time windows vary. Even with augmentation, RL++ achieves almost zero success rate at $N_g = 200$. In contrast, our NS-RTW can still solve all instances with feasible solutions across all scenarios, which further demonstrates our model's

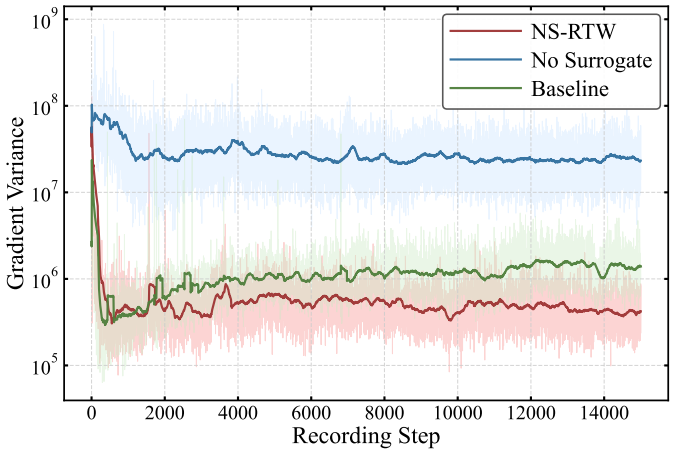

Fig. 8. Gradient variance of the allocation network during training for NS-RTW with the surrogate network, NS-RTW without the surrogate network (denoted as No Surrogate), and NS-RTW with the baseline method (denoted as Baseline). The gradient variance is reported in a logarithmic scale. Light-colored traces show the raw recorded values, and the bold curves show the moving-average smoothed trends. Lower values indicate more stable gradient estimates during training. Our NS-RTW maintains low variance throughout the training process, particularly during the mid-to-late stages.

superior generalization ability.

C. Ablation Study

In this section, we conduct an ablation study to evaluate the effectiveness of both our NeSy framework and several components within our framework, including the augmented feature design (Section IV-B), the masking mechanism (Section IV-C) and the surrogate network design (Section IV-E).

1) Ablation Study for NeSy Framework: Since NS-RTW builds upon PyVRP as the low-level solver, we compare it against a standalone PyVRP as another baseline. This allows us to isolate and validate the impact of our proposed neural guidance mechanism and NeSy framework. As illustrated in Fig. 7, NS-RTW outperforms the standalone PyVRP in all scenarios with $N_g \leq 150$, while NS-RTW++ dominates across the entire test set. Specifically, NS-RTW reduces the median path completion time by 7.30%, 8.79%, and 9.04% across random, clustered, and random-clustered distributions, respectively. NS-RTW++ achieves greater reductions of 13.15%, 11.75%, and 13.25% across the same distributions. These results confirm that the high-level neural planner plays a critical role in optimizing the search space for the low-level solver. Furthermore, PyVRP lacks native support for the min-max objective. Our framework bridges this gap, demonstrating a paradigm for leveraging standard min-sum solvers to address complex min-max routing problems.

2) Ablation Study for Surrogate Network: To demonstrate the effectiveness of the surrogate network, we conduct comparison experiments among our NS-RTW, NS-RTW without the surrogate network (denoted as No Surrogate), and NS-RTW with the traditional baseline method utilized in [1] (denoted as Baseline). Fig. 8 compares the total gradient variance of the allocation network under these three settings. Results show that the surrogate network effectively and consistently suppresses gradient noise, maintaining low variance throughout the training process, particularly during the mid-to-late stages.

TABLE III
SUCCESS RATE COMPARISON BETWEEN NS-RTW AND NS-RTW WITHOUT MASKING MECHANISM (NOMASK) UNDER VARIABLE TIME-WINDOW LENGTHS. HIGHER SUCCESS RATE IN EACH SETTING IS HIGHLIGHTED IN BOLD.

Distribution	Method	$N_g = 100$	$N_g = 150$	$N_g = 200$
		Success Rate	Success Rate	Success Rate
Random	NS-RTW	**100.00%**	**100.00%**	**100.00%**
	NoMask	99.97%	99.40%	86.97%
Clustered	NS-RTW	**100.00%**	**100.00%**	**100.00%**
	NoMask	**100.00%**	99.93%	98.87%
Random-Clustered	NS-RTW	**100.00%**	**100.00%**	**100.00%**
	NoMask	99.97%	99.40%	92.57%

Regarding solution quality, NS-RTW with the surrogate network consistently achieves a lower penalized cost across all settings compared to the Baseline. As shown in Table II, the average performance gap is 4.12%. This gap becomes more pronounced as the problem size increases, rising from 2.00% at $N_g = 50$ to 5.67% at $N_g = 200$, with a peak gap of 8.80% observed under the clustered distribution at $N_g = 200$.

These results associate the lower gradient variance obtained by the surrogate network with more stable policy updates and better final solutions.

3) Ablation Study for Masking: Table II demonstrates that removing the masking mechanism consistently degrades performance, introducing an average optimality gap of 3.40% and up to 5.45% for random distribution at $N_g = 50$. This mechanism proves particularly effective under variable time windows (Section V-B3). As shown in Table III, NS-RTW with masking maintains a 100% success rate across all evaluated settings. In contrast, operating without masking leads to a drop in success rate, reaching a maximum decrease of 13.03% at $N_g = 200$ in random distribution. The mask removes unconditionally infeasible transitions and improves constraint awareness, while global path feasibility is determined by the lower-level solver, thereby improving both feasibility and solution quality.

4) Ablation Study for Augmented Feature Design: The augmented feature design provides a smaller but still consistent contribution. Across all settings, NS-RTW without augmented feature design is uniformly worse than the full NS-RTW, with an average gap of 1.28%. The degradation stays within a relatively narrow range, from 0.48% to 1.95%, and becomes most visible under the clustered distribution. This result shows that the augmented features are not the dominant source of improvement, but they offer stable complementary gains and further refine the routing decisions learned by NS-RTW.

D. Experiments on the Solomon Benchmark

We also conduct experiments on the widely used VRP benchmark Solomon [2], which consists of 100 customers and one depot (101 nodes in total). The original Solomon benchmark is commonly evaluated lexicographically by the number of vehicles and total distance. Instances are categorized by customer distribution into random (R), clustered (C),

and random-clustered (RC), aligning with our configuration in Section V-A1. Each class features two scheduling-horizon variants: short (denoted by 1) and long (denoted by 2). Accordingly, the benchmark comprises six scenario types: R1 (12 instances), R2 (11), C1 (9), C2 (8), RC1 (8), and RC2 (8). To adapt it to our RTW setting, we removed the capacity constraints and fixed the fleet size to 5 agents. The original service times are set to zero. Because the original benchmark allows fleet sizes that may differ from our fixed five-agent setting, this adaptation does not guarantee that every converted instance is feasible. To evaluate our method and the RL baselines on the Solomon benchmark, each instance is converted to match our input representation. Specifically, customer coordinates and time windows are normalized by 100 to align with the training scale, and final costs are rescaled back to the original units. The objective is minimizing the maximum path completion time among all agents. Travel cost is computed using Euclidean distance, and travel time is assumed equal to travel distance. To ensure fairness, any infeasible solution is assigned a large penalty cost of 10,000. The time limit is set to 3s for the lower-level PyVRP within our framework and 15s for the standalone PyVRP to ensure the same total CPU computing time budget.

Table IV reports the success rate (Succ. Rate), runtime, and penalized average cost (Avg. Cost). We fine-tune our model using curriculum learning on data generated following [57], with 300 epochs of standard training followed by 200 epochs under tightened time windows. The RL baseline adopts a dual-learning framework that requires a two-stage training procedure, where a "worker" agent is first pre-trained and then frozen. Since the code for this initial training stage is unavailable, we cannot apply an identical fine-tuning process to the RL baseline. Therefore, to ensure a fair comparison while also demonstrating the zero-shot capability of our method, we report the results of our fine-tuned model alongside the non-fine-tuned version, shown in parentheses.

Since infeasible solutions incur a large penalty cost, higher success rates generally lead to lower average costs. Under this metric, the RL++ generalizes poorly, yielding a 0% success rate across all scenarios despite its fast inference time. In contrast, NS-RTW++ achieves substantially higher success rates than RL++ and the lowest penalized cost in 4 out of 6 scenarios. Meanwhile, standalone PyVRP maintains the highest overall success rate and obtains the lowest penalized cost in the R1 and RC1 scenarios. OR-Tools exhibits relatively stable performance but generally remains inferior to PyVRP. Without fine-tuning, NS-RTW++ still maintains a 100% success rate in the C2, R2, and RC2 scenarios and achieves the second-lowest penalized average cost in each of these three scenarios, surpassed only by the fine-tuned NS-RTW++. Overall, in the evaluated Solomon scenarios, the proposed NS-RTW framework generalizes better than RL++ and remains competitive with traditional routing solvers.

TABLE IV
DETAILED RESULTS ON THE SOLOMON BENCHMARK. FOR EACH SCENARIO, THE BEST SUCCESS RATE AND THE LOWEST RUNTIME AND PENALIZED AVERAGE COST ARE HIGHLIGHTED IN BOLD. FOR NS-RTW++, VALUES OUTSIDE PARENTHESES ARE OBTAINED BY THE FINE-TUNED MODEL, WHEREAS VALUES IN PARENTHESES CORRESPOND TO THE NON-FINE-TUNED (ZERO-SHOT) MODEL.

Scenario	Metric	NS-RTW++	RL++	OR-Tools	PyVRP
C1	Succ. Rate	**100.0** (77.8)	0.0	88.9	**100.0**
	Runtime(s)	7.273	**1.079**	2.586	15.044
	Avg. Cost	**966.7** (2952.2)	10000.0	1974.2	993.6
C2	Succ. Rate	**100.0** (100.0)	0.0	50.0	**100.0**
	Runtime(s)	7.288	**1.079**	3.665	15.027
	Avg. Cost	**3020.9** (3023.1)	10000.0	6492.4	3036.8
R1	Succ. Rate	8.3 (0.0)	0.0	8.3	**25.0**
	Runtime(s)	7.277	**1.086**	2.789	15.040
	Avg. Cost	9184.3 (10000.0)	10000.0	9182.6	**7552.2**
R2	Succ. Rate	**100.0** (100.0)	0.0	**100.0**	**100.0**
	Runtime(s)	7.336	**1.064**	4.140	15.044
	Avg. Cost	**781.1** (787.2)	10000.0	787.4	875.1
RC1	Succ. Rate	0.0 (0.0)	0.0	0.0	**25.0**
	Runtime(s)	7.346	**1.082**	2.703	15.057
	Avg. Cost	10000.0 (10000.0)	10000.0	10000.0	**7555.2**
RC2	Succ. Rate	**100.0** (100.0)	0.0	**100.0**	**100.0**
	Runtime(s)	7.356	**1.070**	3.299	15.028
	Avg. Cost	**770.1** (773.1)	10000.0	773.4	822.1

E. Real-World Case Study

1) Dataset and Experimental Design: To further evaluate the practical applicability of the proposed method, we conduct a real-world case study using MAMUT [35], an open-source urban-logistics routing benchmark. Its road-network data are derived from OpenStreetMap [58], and travel costs are computed on the actual road network rather than by point-to-point Euclidean distance. The public request layer contains customer locations and service time windows, thereby reflecting key operational characteristics of real urban delivery orders in a reproducible form.

We select 18 representative cities across 6 continents and generate 90 instances in total. Each instance contains 101 nodes, including one depot and 100 targets, with at most 5 agents that can be used. Detailed parameter configurations for generating the MAMUT instances are provided in Appendix B for reproducibility.

To match our RTW formulation, we retain customer locations, asymmetric road-network travel costs, and service time windows, while ignoring demand and vehicle-capacity constraints and setting service durations to zero.

For model input, coordinates and temporal quantities are normalized to match the synthetic training distribution, and reported objective values are converted back to minutes.

We evaluate NS-RTW++, OR-Tools, and PyVRP under the target min-max objective. Standalone PyVRP retains its native min-sum routing objective; in the min-max experiments, its returned solution is evaluated using the maximum path completion time. In addition to the target min-max setting, we also introduce a min-sum variant as an extended robustness test under a different objective. For the min-sum variant, we replace the target objective with $H_{\text{sum}} \doteq \sum_{m\in\mathcal{M}} h(g(a^m;\mu^*))$ while keeping the synthetic training data and all other training settings unchanged. No MAMUT-specific fine-tuning is performed. The RL baseline is not included in this case study because it is designed for direct point-to-point Euclidean costs

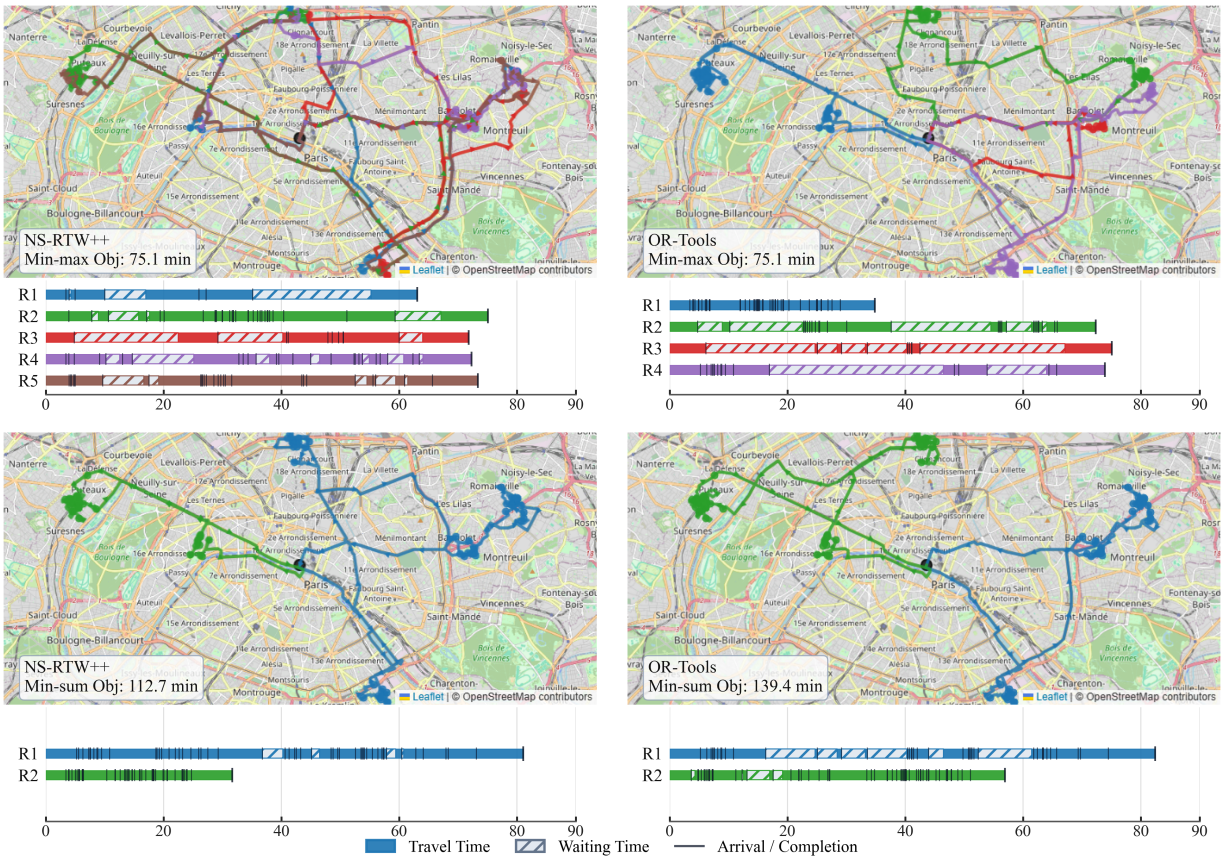

Fig. 9. Visualization of planning results for a Paris urban logistics instance derived from MAMUT [35]. The proposed NS-RTW++ ties OR-Tools for the lowest min-max cost and achieves the lowest min-sum cost among the displayed solutions.

TABLE V
RESULTS ON THE MAMUT REAL-WORLD ROAD-NETWORK BENCHMARK. OBJECTIVE COST IS THE MEAN OVER THE INSTANCES SOLVED BY ALL METHODS, REPORTED IN MINUTES. SUCCESS RATE IS COMPUTED OVER ALL 90 INSTANCES.

Objective	Metric	NS-RTW++	OR-Tools	PyVRP
Min-sum	Succ. Rate	82.22%	98.89%	**100.00%**
	Avg. Cost	**237.86**	265.46	317.04
	Runtime(s)	**3.58**	5.00	5.03
Min-max	Succ. Rate	97.78%	98.89%	**100.00%**
	Avg. Cost	152.47	**152.38**	154.06
	Runtime(s)	**3.38**	5.00	5.03

and requires re-engineering to operate on road-network travel-cost matrices. For OR-Tools, parameter settings are adjusted for the two objectives, as detailed in Appendix A3.

2) Quantitative Results: As shown in Table V, under the target min-max objective, NS-RTW++ achieves a 97.78% success rate and an average objective cost of 152.47 minutes, nearly matching the best OR-Tools result while improving over PyVRP in objective cost. Meanwhile, NS-RTW++ requires the shortest runtime, reducing the runtime by about 32% compared with OR-Tools and PyVRP. These results show that the proposed framework transfers effectively to real road-network travel costs while maintaining competitive solution quality and higher computational efficiency.

Under the min-sum objective, which is included as an extended robustness test rather than the main target task, NS-RTW++ obtains the lowest average cost and the shortest runtime on the commonly solved instances. Its success rate is lower than those of the search-based solvers, which is reasonable because the framework is designed primarily for min-max RTW and the min-sum variant is obtained without MAMUT-specific fine-tuning. Nevertheless, the results indicate that the current framework can also be adapted to min-sum routing on real road-network instances.

3) Qualitative Case Study: We further select one representative instances Paris as qualitative case studies. The visualization results show that NS-RTW++ generally incurs less waiting time than OR-Tools, suggesting that the learned allocation module can better capture global spatio-temporal relations among requests. The displayed paths show that NS-RTW++ not only selects geographically close targets but also aligns targets with compatible time windows, resulting in fewer long waiting intervals between consecutive visits.

VI. CONCLUSION AND FUTURE WORK

This paper investigates the agent routing problem with time windows (RTW) by proposing NS-RTW, a novel self-supervised neuro-symbolic approach. By leveraging a neural network to decompose the complex global problem into manageable single-agent TSP-TW subproblems and solving

them with classic solvers, NS-RTW effectively combines the computational efficiency of learning-based approaches with the solution quality of search-based methods. Experimental results demonstrate that our framework consistently yields superior solutions with high efficiency across varying scenarios, validating its potential for large-scale routing tasks.

In future work, we plan to extend NS-RTW to bridge the gap between theoretical routing and real-world logistics. First, we aim to incorporate vehicle capacity into our framework. Second, we will explore the adaptability of our model in dynamic environments, specifically focusing on scenarios with dynamic obstacles or uncertain travel times, to achieve robust real-time path planning.

APPENDIX

In this section, we detail the experimental configurations for both our NS-RTW and the comparative baselines. For the RL baseline, we strictly adhere to the default hyperparameter configurations reported in [1].

A. Detailed Settings for Each Solver

TABLE VI
HYPERPARAMETERS AND ARCHITECTURE OF THE ALLOCATION NETWORK IN NS-RTW

Name	Component	Notation	Configuration / Value
Global Parameters	Hidden Dimension	D	128
	Batch Size	–	128
	Learning Rate	–	10^{-4}
	Total Iterations	–	15000
	Penalty Coefficient	β	50
Initial Encoder	Node Proj. MLP	$\phi_{\mathrm{c}}, \phi_{\mathrm{p}}, \phi_{\mathrm{w}}$	Linear $\rightarrow$ BN [59] $\rightarrow$ PReLU
	Node Fusion MLP	$\phi^{\mathrm{x}}_{\mathrm{fuse}}$	Linear $\rightarrow$ PReLU
	Edge Proj. MLP	$\phi_{\mathrm{s}}, \phi_{\mathrm{t}}$	Linear $\rightarrow$ BN $\rightarrow$ SiLU
	Edge Fusion MLP	$\psi^{\mathrm{e}}_{\mathrm{fuse}}$	Linear $\rightarrow$ SiLU [60]
Graph Encoder	Number of GATv2 Blocks	L	2
	Attention Heads	K	16
	Dropout in GATv2	–	first layer: 0, second layer: 0.5
	Output MLP	ϕ_{out}	Linear $\rightarrow$ PReLU $\rightarrow$ Linear
	Dropout in pooling block	$\mathrm{D}_{\mathrm{drop}}(\cdot)$	0.5
Agent Encoder	Fusion MLP	$\phi^{\mathrm{c}}_{\mathrm{fuse}}$	Linear $\rightarrow$ PReLU
	Attention Heads	S	4
Assignment Decoder	Clipping Factor	η	10

1) Settings of PyVRP: The implementation of PyVRP follows official guidelines provided in the PyVRP documentation for the VRP with time windows. We employ a runtime-based stopping criterion, where the time budget scales with the problem size. The time limit is set to 0.1s during training. During inference, it is set to $\{0.05\text{s}, 0.1\text{s}, 0.4\text{s}, 0.9\text{s}\}$ for problem sizes $N_g \in \{50, 100, 150, 200\}$, respectively. The standalone PyVRP baseline and each lower-level PyVRP solver within NS-RTW use one dedicated thread. To ensure a fair comparison in the ablation study (Section V-C), the standalone PyVRP baseline is granted a time budget five times that of the lower-level solver used in our proposed model to ensure both methods consume the exact same total computational CPU time. For the MAMUT experiments, each lower-level PyVRP solver within NS-RTW is given a 1s solve-time budget while standalone PyVRP is given 5s, five times the per-solver budget, to account for the five decomposed single-agent subproblems. The adopted version of PyVRP is $0.9.1$.

2) Settings of Network: The hyperparameters and detailed network architectures utilized in our experiments are summarized in Table VI for the allocation network and Table VII for the surrogate network.

TABLE VII
HYPERPARAMETER SETTINGS AND ARCHITECTURE DETAILS FOR SURROGATE NETWORK IN OUR NS-RTW

Name	Component	Notation	Configuration / Value
Global Parameters	Learning Rate	–	10^{-3}
	Hidden Dimension	–	64
Architecture	MLP	–	Network Depth: 3, Activation: Tanh

TABLE VIII
CONFIGURATION AND PARAMETER SETTINGS FOR OR-TOOLS ROUTING MODEL

Parameter	Configuration/Value	Explanation
Version	v9.12	-
First Solution Heuristic	`PATH_CHEAPEST_ARC`	Strategy for Finding First Solution
Search Parameters	Default	-
Time Limit	3000 s	Maximum Search Time
Maximum Slack	300	Maximum Allowed Waiting Time
Maximum Horizon	10^6	Maximum Route Duration
Global Span Coeff.	100	Weight for Span Minimization
Penalty for Dropping Visits	10^5	Penalty for Unvisited Nodes

TABLE IX
GENERATION PROTOCOL FOR THE MAMUT REAL-WORLD BENCHMARK.

Parameter	Configuration
Seeds	0, 1, 2, 13, 14
Travel-cost metric	Fastest-path metric
Problem type	VRPTW
Generation method	Hybrid
Number of customers	100
Demand type	1
Average route size	4
Only intersections	True
Depot mode	Center
Customer mode	Clustered
Number of cluster seeds	6
Cluster decay	120 (360 / 1080)
Point-of-interest categories	Restaurant, cafe, fast food, school, university, pharmacy
Hybrid POI share	0.0
Global planning horizon	[0, 5000]

3) Settings of OR-Tools: Our implementation primarily adheres to the official guidelines provided in the Google OR-Tools documentation for the Vehicle Routing Problem with Time Windows (VRPTW). However, our experiments showed that the default heuristics faced significant convergence challenges on larger-scale RTW (e.g., $N_g \geq 150$), often failing to produce feasible solutions within limited time. To

address this, we employed the "Penalties and Dropping Visits" mechanism in OR-Tools. This makes visits optional in the final optimization model, with a substantial penalty (10^5) assigned to each unvisited node. This ensures the solver prioritizes complete coverage in the final solution while maintaining search flexibility. In evaluation, any final output containing dropped nodes is classified as an infeasible solution. Therefore, all reported successful instances for OR-Tools strictly contain zero dropped nodes.

Furthermore, to align with the min-max objective, OR-Tools uses the travel-cost evaluator together with a *Global Span Cost Coefficient* of 100, which encourages a smaller maximum completion time. The standalone OR-Tools baseline processes the full problem using a single thread by default. Detailed settings are summarized in Table VIII. For MAMUT experiments, the *Global Span Coeff.* is disabled for the min-sum objective and the time limit is 5s.

B. MAMUT Benchmark Generation Details

Table IX shows the generation protocol. Cluster decay is adjusted if 120 is not suitable for generating the instance.

REFERENCES

[1] R. Zhang, C. Zhang, Z. Cao, W. Song, P. S. Tan, J. Zhang, B. Wen, and J. Dauwels, "Learning to solve multiple-tsp with time window and rejections via deep reinforcement learning," *IEEE Transactions on Intelligent Transportation Systems*, vol. 24, no. 1, pp. 1325–1336, 2023.

[2] M. M. Solomon, "Algorithms for the vehicle routing and scheduling problems with time window constraints," *Operations research*, vol. 35, no. 2, pp. 254–265, 1987.

[3] J. Desrosiers, F. Soumis, M. Desrochers, and M. SauvéGerad, "Methods for routing with time windows," *European Journal of Operational Research*, vol. 23, no. 2, pp. 236–245, 1986.

[4] T. Phiboonbanakit, T. Horanont, V.-N. Huynh, and T. Supnithi, "A hybrid reinforcement learning-based model for the vehicle routing problem in transportation logistics," *IEEE Access*, vol. 9, pp. 163 325–163 347, 2021.

[5] Y. Wang, Y. Wang, and J. Leng, "A study on the vehicle routing planning method for fresh food distribution," *Applied Sciences*, vol. 14, no. 22, p. 10499, 2024.

[6] S. Riazi, O. Wigström, K. Bengtsson, and B. Lennartson, "A column generation-based gossip algorithm for home healthcare routing and scheduling problems," *IEEE Transactions on Automation Science and Engineering*, vol. 16, no. 1, pp. 127–137, 2019.

[7] T. Bektas, "The multiple traveling salesman problem: an overview of formulations and solution procedures," *Omega*, vol. 34, no. 3, pp. 209–219, 2006.

[8] L. Costa, C. Contardo, and G. Desaulniers, "Exact branch-price-and-cut algorithms for vehicle routing," *Transportation Science*, vol. 53, no. 4, pp. 946–985, 2019.

[9] D. Lu and F. Gzara, "The robust vehicle routing problem with time windows: Solution by branch and price and cut," *European Journal of Operational Research*, vol. 275, no. 3, pp. 925–938, 2019.

[10] I. Kara and T. Bektas, "Integer linear programming formulations of multiple salesman problems and its variations," *European Journal of Operational Research*, vol. 174, no. 3, pp. 1449–1458, 2006.

[11] R. Baldacci, N. Christofides, and A. Mingozzi, "An exact algorithm for the vehicle routing problem based on the set partitioning formulation with additional cuts," *Mathematical Programming*, vol. 115, no. 2, pp. 351–385, 2008.

[12] B. M. Baker and M. Ayechew, "A genetic algorithm for the vehicle routing problem," *Computers & Operations Research*, vol. 30, no. 5, pp. 787–800, 2003.

[13] M. Gendreau, A. Hertz, and G. Laporte, "A tabu search heuristic for the vehicle routing problem," *Management Science*, vol. 40, no. 10, pp. 1276–1290, 1994.

[14] J. E. Bell and P. R. McMullen, "Ant colony optimization techniques for the vehicle routing problem," *Advanced Engineering Informatics*, vol. 18, no. 1, pp. 41–48, 2004.

[15] Z. Zong, X. Tong, M. Zheng, and Y. Li, "Reinforcement learning for solving multiple vehicle routing problem with time window," *ACM Trans. Intell. Syst. Technol.*, vol. 15, no. 2, p. 19, 2024.

[16] Y. Zhang, J. Wang, and Z. Zhang, "Edge-based formulation with graph attention network for practical vehicle routing problem with time windows," in *2022 International Joint Conference on Neural Networks (IJCNN)*, 2022, pp. 01–08.

[17] Z. Zheng, S. Yao, Z. Wang, T. Xialiang, M. Yuan, and K. Tang, "DPN: Decoupling partition and navigation for neural solvers of min-max vehicle routing problems," in *Proceedings of the 41st International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, R. Salakhutdinov, Z. Kolter, K. Heller, A. Weller, N. Oliver, J. Scarlett, and F. Berkenkamp, Eds., vol. 235. PMLR, 21–27 Jul 2024, pp. 61 559–61 592.

[18] H. Gao, X. Zhou, X. Xu, Y. Lan, and Y. Xiao, "Amarl: An attention-based multiagent reinforcement learning approach to the min-max multiple traveling salesmen problem," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 35, no. 7, pp. 9758–9772, 2024.

[19] J. Park, C. Kwon, and J. Park, "Learn to solve the min-max multiple traveling salesmen problem with reinforcement learning," in *Proceedings of the 2023 International Conference on Autonomous Agents and Multiagent Systems*, ser. AAMAS '23. Richland, SC: International Foundation for Autonomous Agents and Multiagent Systems, 2023, p. 878–886.

[20] A. Buinoschi-Tîrpescu and M. E. Breabăn, "Solving min-max mtsp in a reinforcement learning context," *Procedia Computer Science*, vol. 246, pp. 1001–1010, 2024, 28th International Conference on Knowledge Based and Intelligent information and Engineering Systems (KES 2024).

[21] Y. Guo, Z. Ren, and C. Wang, "imtsp: Solving min-max multiple traveling salesman problem with imperative learning," in *2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2024, pp. 10 245–10 252.

[22] C. Chi, A. Aboussalah, E. Khalil, J. Wang, and Z. Sherkat-Masoumi, "A deep reinforcement learning framework for column generation," *Advances in Neural Information Processing Systems*, vol. 35, pp. 9633–9644, 2022.

[23] K. Xu, Z. Cao, C. Zheng, and L. Liu, "Learning to search for vehicle routing with multiple time windows," *Computers & Industrial Engineering*, p. 111760, 2025.

[24] W. Feijen, G. Schäfer, K. Dekker, and S. Pieterse, "Learning-enhanced neighborhood selection for the vehicle routing problem with time windows," 2024.

[25] J. Zhao, M. Mao, X. Zhao, and J. Zou, "A hybrid of deep reinforcement learning and local search for the vehicle routing problems," *IEEE Transactions on Intelligent Transportation Systems*, vol. 22, no. 11, pp. 7208–7218, 2021.

[26] R. Wu, R. Wang, J. Hao, Q. Wu, P. Wang, and D. Niyato, "Multiobjective vehicle routing optimization with time windows: A hybrid approach using deep reinforcement learning and nsga-ii," *IEEE Transactions on Intelligent Transportation Systems*, vol. 26, no. 3, pp. 4032–4047, 2025.

[27] P. He, J.-K. Hao, and J. Xia, "Learning-guided iterated local search for the minmax multiple traveling salesman problem," *Computers & Operations Research*, vol. 185, p. 107255, 2026.

[28] P. He and J.-K. Hao, "Memetic search for the minmax multiple traveling salesman problem with single and multiple depots," *European Journal of Operational Research*, vol. 307, no. 3, pp. 1055–1070, 2023.

[29] L. Bertazzi, B. Golden, and X. Wang, "Min–max vs. min–sum vehicle routing: A worst-case analysis," *European Journal of Operational Research*, vol. 240, no. 2, pp. 372–381, 2015.

[30] C. Wang, K. Ji, J. Geng, Z. Ren, T. Fu, F. Yang, Y. Guo, H. He, X. Chen, Z. Zhan, Q. Du, S. Su, B. Li, Y. Qiu, Y. Du, Q. Li, Y. Yang, X. Lin, and Z. Zhao, "Imperative learning: A self-supervised neuro-symbolic learning framework for robot autonomy," *The International Journal of Robotics Research*, vol. 0, no. 0, p. 02783649251353181, 2025.

[31] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Machine Learning*, vol. 8, pp. 229–256, 1992.

[32] B. L. Nelson, "Control variate remedies," *Operations Research*, vol. 38, no. 6, pp. 974–992, 1990.

[33] V. Furnon and L. Perron, "OR-Tools Routing Library," Google, 2025, version 9.12. Accessed Jul. 15, 2026. [Online]. Available: https://developers.google.com/optimization/routing/

[34] N. A. Wouda, L. Lan, and W. Kool, "PyVRP: a high-performance VRP solver package," *INFORMS Journal on Computing*, vol. 36, no. 4, pp. 943–955, 2024.

[35] F. Rascoussier and A. Pichon, "MAMUT-routing," IMT Atlantique, INSA Lyon, and Université Bretagne Sud, France, May 2026, research

software, version 0.1.0, archived by Software Heritage (SWHID: swh:1:rev:5dd0e60f69816a5e6afa3fa8c3c95902c5de3245). [Online]. Available: https://github.com/ANR-MAMUT/MAMUT-routing

[36] Y.-D. Kwon, J. Choo, B. Kim, I. Yoon, Y. Gwon, and S. Min, "Pomo: Policy optimization with multiple optima for reinforcement learning," *Advances in Neural Information Processing Systems*, vol. 33, pp. 21 188–21 198, 2020.

[37] A. G. Philip, Z. Ren, S. Rathinam, and H. Choset, "A mixed-integer conic program for the moving-target traveling salesman problem based on a graph of convex sets," in *2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2024, pp. 8847–8853.

[38] S. Zhao, Y. Wu, and Z. Ren, "Bi-objective search for the traveling salesman problem with time windows and vacant penalties," in *Proceedings of the International Symposium on Combinatorial Search*, vol. 18, no. 1, Jul. 2025, pp. 171–179.

[39] X. Shi, Y. Liang, H. Lee, C. Lu, and Q. Wang, "Particle swarm optimization-based algorithms for tsp and generalized tsp," *Information Processing Letters*, vol. 103, no. 5, pp. 169–176, 2007.

[40] J. Kytöjoki, T. Nuortio, O. Bräysy, and M. Gendreau, "An efficient variable neighborhood search heuristic for very large scale vehicle routing problems," *Computers & Operations Research*, vol. 34, no. 9, pp. 2743–2757, 2007.

[41] G. Laporte, "Fifty years of vehicle routing," *Transportation Science*, vol. 43, no. 4, pp. 408–416, 2009.

[42] V. Ilin, D. Simić, S. D. Simić, S. Simić, N. Saulić, and J. L. Calvo-Rolle, "A hybrid genetic algorithm, list-based simulated annealing algorithm, and different heuristic algorithms for the travelling salesman problem," *Logic Journal of the IGPL*, vol. 31, no. 4, pp. 602–617, 2023.

[43] A. Maroof, B. Ayvaz, and K. Naeem, "Logistics optimization using hybrid genetic algorithm (hga): A solution to the vehicle routing problem with time windows (vrptw)," *IEEE Access*, vol. 12, pp. 36 974–36 989, 2024.

[44] O. Vinyals, M. Fortunato, and N. Jaitly, "Pointer networks," in *Advances in Neural Information Processing Systems*, vol. 28, 2015.

[45] N. Díaz-Rodríguez, A. Lamas, J. Sanchez, G. Franchi, I. Donadello, S. Tabik, D. Filliat, P. Cruz, R. Montes, and F. Herrera, "Explainable neural-symbolic learning (x-nesyl) methodology to fuse deep learning representations with expert knowledge graphs: The monumai cultural heritage use case," *Information Fusion*, vol. 79, pp. 58–83, 2022.

[46] R. Manhaeve, S. Dumancic, A. Kimmig, T. Demeester, and L. De Raedt, "Deepproblog: Neural probabilistic logic programming," in *Advances in Neural Information Processing Systems*, vol. 31, 2018.

[47] S. Badreddine, A. d'Avila Garcez, L. Serafini, and M. Spranger, "Logic tensor networks," *Artificial Intelligence*, vol. 303, p. 103649, 2022.

[48] S. Nasehibasharzad, F. Choudhury, E. Tanin, and M. Sarvi, "Deepmdv: Global spatial matching for multi-depot vehicle routing problems," in *Proceedings of the 33rd ACM International Conference on Advances in Geographic Information Systems*, 2025, pp. 77–89.

[49] S. Brody, U. Alon, and E. Yahav, "How attentive are graph attention networks?" 2022.

[50] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio, "Graph attention networks," *arXiv preprint arXiv:1710.10903*, 2017.

[51] J. L. Ba, J. R. Kiros, and G. E. Hinton, "Layer normalization," *arXiv preprint arXiv:1607.06450*, 2016.

[52] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," 2015.

[53] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. u. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in Neural Information Processing Systems*, vol. 30, 2017.

[54] E. Greensmith, P. L. Bartlett, and J. Baxter, "Variance reduction techniques for gradient estimates in reinforcement learning," *J. Mach. Learn. Res.*, vol. 5, p. 1471–1530, Dec. 2004.

[55] E. Queiroga, R. Sadykov, E. Uchoa, and T. Vidal, "10,000 optimal cvrp solutions for testing machine learning based heuristics," *AAAI'22 Workshop on Machine Learning for Operations Research*, 2022.

[56] E. Uchoa, D. Pecin, A. Pessoa, M. Poggi, T. Vidal, and A. Subramanian, "New benchmark instances for the capacitated vehicle routing problem," *European Journal of Operational Research*, vol. 257, no. 3, pp. 845–858, 2017.

[57] J. K. Falkner and L. Schmidt-Thieme, "Learning to solve vehicle routing problems with time windows through joint attention," *arXiv preprint arXiv:2006.09100*, 2020.

[58] OpenStreetMap contributors, "OpenStreetMap database," https://www.openstreetmap.org/copyright, open Data Commons Open Database License (ODbL) 1.0. Accessed Jul. 15, 2026.

[59] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *International conference on machine learning*. pmlr, 2015, pp. 448–456.

[60] S. Elfwing, E. Uchibe, and K. Doya, "Sigmoid-weighted linear units for neural network function approximation in reinforcement learning," *Neural networks*, vol. 107, pp. 3–11, 2018.

Jiaxian Li received the B.E. degree in transportation engineering from Central South University, Changsha, China, in 2025. He is currently pursuing his M.S. degree in Control Science and Engineering with Global College, Shanghai Jiao Tong University, Shanghai, China.

His research interests include neuro-symbolic approach and self-supervised learning for path and motion planning.

Yifan Guo received the B.S. degree in computer science from Jiangsu University, Jiangsu, China, in 2018, and the M.S. degree in electrical and computer engineering from the University of Pittsburgh, Pittsburgh, PA, USA, in 2020. He is currently pursuing the Ph.D. degree with the School of Electrical and Computer Engineering, Purdue University, West Lafayette, IN, USA.

His research interests include machine learning, multi-agent control, and cybersecurity.

Chen Wang (Senior Member, IEEE) received the B.Eng. degree in electrical engineering from Beijing Institute of Technology, Beijing, China, in 2014, and the Ph.D. degree in electrical engineering from Nanyang Technological University, Singapore, in 2019. He is an Assistant Professor with the Department of Computer Science and Engineering, University at Buffalo (UB), where he leads the Spatial AI and Robotics Lab. From 2019 to 2022, he was a Postdoctoral Fellow with the Robotics Institute, Carnegie Mellon University. Dr. Wang's research focuses on Spatial AI and robotics. He was an Associate Editor for IEEE Transactions on Robotics, International Journal of Robotics Research, and IEEE Robotics and Automation Letters, and as an Area Chair for the IEEE/CVF Conference on Computer Vision and Pattern Recognition, IEEE International Conference on Robotics and Automation, and the Conference on Neural Information Processing Systems. He was an Associate Co-chair of the IEEE RAS Technical Committee on Computer and Robot Vision.

Zhongqiang (Richard) Ren (Member, IEEE) received the dual B.E. degree in mechatronics from Tongji University, Shanghai, China, and FH Aachen University of Applied Sciences, Aachen, Germany, in 2015, and the M.S. degree in mechanical engineering and the Ph.D. degree in mechanical engineering from Carnegie Mellon University, Pittsburgh, PA, USA, in 2017 and 2022, respectively.

He is currently an Associate Professor with the Global College, Shanghai Jiao Tong University, Shanghai, China. His research interests include robotics, path and motion planning, and multi-robot systems. He is an Associate Editor for IEEE Transactions on Robotics, Autonomous Robots, ICRA and IROS.