DIFFPTR: DIFFUSION POINTER NETWORKS FOR MULTI-AGENT COMBINATORIAL OPTIMIZATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Combinatorial Optimization problems in Multi-Agent systems are particularly challenging due to the combination of NP-hardness, agent coordination, and exponentially large joint action space. Existing Constructive (or AutoRegressive) methods such as PARCO typically rely on *pointers* to deterministically model the joint action distribution, from which multiple trajectories are sampled to explore the solution space. Although these methods have achieved significant progress, they still suffer from issues such as limited generation diversity and poor generalization capability. In this work, we propose the Diffusion Pointer Networks (**DiffPtr**), which incorporates diffusion models into the pointer generation process, to capture the diverse multimodal distributions over multi-agent joint actions and trajectories. DiffPtr introduces an independence assumption between arbitrary agent–action pairs and defines a discrete diffusion process based on binary action variables, consisting of a forward noising process and a state-conditioned reverse denoising process. By iteratively generating diffusion pointers under different noise levels, DiffPtr progressively refines the joint actions, thereby enhancing inter-agent coordination, promoting solution diversity, and improving robustness under out-of-distribution generalization. Experimental results show that DiffPtr outperforms existing methods in both solution quality and generalization, highlighting the potential of diffusion-based policies for complex Multi-agent Combinatorial Optimization problems. Our code, datasets, and checkpoints are publicly available at the anonymous link `https://anonymous.4open.science/r/DiffPtr`.

1 INTRODUCTION

Combinatorial Optimization problems in multi-agent systems (*multi-agent* CO) (Berto et al., 2025; Park et al., 2023; Liu et al., 2024) frequently arise in real-world scenarios such as logistics, transportation (Liberatore et al., 2014) and multi-robot collaboration (Chen et al., 2026), where multiple agents are required to accomplish cooperative tasks under complex constraints. However, the inherent computational complexity of these problems, i.e., NP-hardess (Peres & Castelli, 2021), poses significant challenges to obtaining optimal solutions efficiently. Furthermore, *multi-agent* CO typically involve additional constraints and specialized intra-swarm optimization objectives, such as minimizing task completion time or the maximum individual cost (Liu et al., 2024; Guo et al., 2024), thereby further increasing the solving complexity and difficulty, and conventional exact or heuristic solvers often struggle to obtain high-quality solutions within limited computational time.

In recent years, learning-based neural solvers (Kool et al., 2019; Bi et al., 2026; Kim et al., 2022; Bello et al., 2016; Zeng et al., 2024) have been widely adopted to tackle CO problems. Unlike traditional approaches that rely on manually designed heuristics, these methods leverage Supervised Learning (SL) or Reinforcement Learning (RL) to extract generalizable patterns from data, achieving comparable solution quality and faster inference speed. Among these neural solvers, particularly existing Constructive (or AutoRegressive) approaches (Kool et al., 2019; Bello et al., 2016; Berto et al., 2025), the solving process is formulated as a Markov Decision Process (MDP), where solutions are incrementally constructed from scratch. For *multi-agent* CO, in order to handle the input-dependent joint action spaces and the varying number of agents, some special *pointer* mechanisms (e.g., 2D-Ptr (Liu et al., 2024), multiple pointers(Berto et al., 2025)) are then employed to model the joint action distributions. These methods have been applied to solve *multi-agent* CO problems with various complex constraints, including the Heterogeneous Capacitated Vehicle Routing Problems (HCVRPs)(Li et al., 2022), and

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

Open Multi-Depot Capacitated Pickup and Delivery Problem (OMDCPDPs) (Zong et al., 2022; Berto et al., 2025), demonstrating their potential for addressing a wide range of real-world problems.

Although existing approaches have achieved substantial progress, they still suffer from limitations in both solution generation and generalization capability. Due to the *softmax bottleneck* (Yang et al., 2019; Chang & McCallum, 2022), the deterministic pointers generated by existing methods lack sufficient expressive power to capture the intrinsic multimodal nature of the joint action space in multi-agent CO. As a result, the diversity of action sampling is reduced, thereby constraining the generative capability of the model. Moreover, during out-of-distribution generalization to large-scale problems, the rapidly expanding action space induces significant shifts in the action distribution of the pointer, leading to degraded sampling quality. A representative phenomenon is that multiple sampling attempts can even perform worse than a single greedy decoding step, which is discussed in the experiments in Section 4.

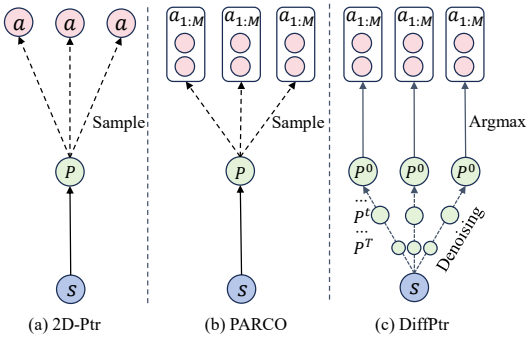

Figure 1: Three pointer-based action sampling methods. 2D-Ptr and PARCO use deterministic pointers for sampling single-agent and multi-agent actions respectively, while DiffPtr generates stochastic diffusion pointers, where each pointer corresponds to a multi-agent action set via Argmax.

Inspired by recent advances in diffusion policies(Kang et al., 2023; Chi et al., 2025), we propose to incorporate diffusion models into the pointer generation process for *multi-agent* CO, and introduce a novel framework termed the Diffusion Pointer Network (DiffPtr). As illustrated in Fig.1, unlike existing approaches that generate deterministic pointers, DiffPtr transfers the stochasticity of action selection to the denoising process of pointers, thereby modeling multimodal distributions through enhanced pointer diversity. Specifically, we define a multi-agent action variable under an *independence assumption*, denoted as $A^t \in \{0,1\}^{\mathcal{A}}$, where $\mathcal{A}$ is the joint action space, along with a corresponding discrete noising process. A diffusion pointer network is then trained to approximate the conditional denoising process described by the probability distribution $P^t : p_\theta(A^0 \mid A^{t+1}, s)$. We refer to P^tas a *diffusion pointer*, as it retains the classical pointer mechanism's ability to adapt to variable action spaces. Furthermore, a hierarchical diffusion decoder based on attention mechanisms is carefully designed to facilitate inter-agent information exchange and extract noisy action information. For training, we train DiffPtr through Imitation learning (IL) (Zare et al., 2024) on a trajectory dataset collected from an existing RL teacher model, such as PARCO(Berto et al., 2025), since the optimal solution to multi-agent CO is difficult to obtain. Experimental results demonstrate that the trained DiffPtr surpasses its teacher in generative capability while significantly improving generalization performance.

Notably, although existing diffusion-based neural solvers (Sun & Yang, 2023; Li et al., 2023; 2024b) have demonstrated promising performance on relatively simple CO problems such as the Traveling Salesman Problem (TSP) and Maximal Independent Set (MIS), they typically adopt a non-autoregressive paradigm that generates complete solutions in a single process, making it difficult to capture the rich intermediate states arising in multi-agent systems. In contrast, our method is capable of effectively modeling the complex decision-making dynamics in multi-agent CO.

Our contributions can be summarized as follows: 1) We propose a novel neural solver, DiffPtr, which formulates the solving process of *multi-agent* CO as an outer Markov decision process coupled with an inner multi-step denoising process, leveraging diffusion models to generate the joint actions of multiple agents. 2) We design a diffusion pointer network equipped with a carefully crafted hierarchical diffusion decoder to generate diffusion pointers, and train the model via Imitation Learning. 3) We conduct extensive experiments on two complex *multi-agent* CO problems, namely the HCVRP and OMDCPDP, where DiffPtr achieves superior generation capability and generalization performance compared with existing state-of-the-art learning-based methods.

2 Related Works

Neural Solvers for (*multi-agent*) CO. Neural combinatorial optimization solvers, which have attracted widespread attention in recent years, can be broadly categorized into three classes: autoregressive constructive methods, non-autoregressive constructive methods, and improvement-based methods. Autoregressive constructive approaches (Kool et al., 2018; Kwon et al., 2020; Kim et al., 2022; Liu et al., 2024; Berto et al., 2025) typically employ carefully designed neural networks to incrementally build complete solutions from scratch. These networks may adopt various architectures such as Recurrent Neural Networks (RNNs) (Vinyals et al., 2015), Transformers (Jin et al., 2023; Vaswani et al., 2017), but generally share similar pointer-based output head. In contrast, non-autoregressive constructive methods (Fu et al., 2021; Sun & Yang, 2023) generate a heatmap in a single forward pass to guide the solution construction process, which remains fixed throughout decoding. Improvement-based methods(Wu et al., 2019; Li et al., 2024a), differ fundamentally from the aforementioned approaches, and they start from an initial solution and iteratively refine it by selecting neural network-guided local search operators, such as 2-opt, to obtain improved solutions.

Neural solvers specifically designed for multi-agent CO have only recently emerged (Zong et al., 2022; Son et al., 2024; Li et al., 2022; 2021). Due to the presence of complex constraints and the need for coordination among heterogeneous agents, multi-agent CO is significantly more challenging than some CO problems with a single agent like TSP. Among pioneering works, 2D-Ptr (Liu et al., 2024) focus on HCVRP and introduce a two-dimensional pointer mechanism that simultaneously determines vehicle selection and customer assignment, thereby addressing the generalization challenge across heterogeneous fleets. PARCO (Berto et al., 2025) proposed a multi-pointer mechanism to parallelize the route construction process for multiple agents. Additional extensions and related studies include USPR(Hua et al., 2026), which addresses the Profiled Vehicle Routing Problem (PVRP), and RANS(Liu et al., 2025), which focuses on enhancing generalization performance. Despite their effectiveness, these methods typically adopt a deterministic pointer, and actions are subsequently sampled according to the probability distribution defined by the pointer. Such deterministic pointer generation mechanisms inherently limit the expressive and generative capabilities of neural solvers.

Diffusion-based Neural Solvers. Recently, diffusion-based solvers(Li et al., 2025; Sun & Yang, 2023; Li et al., 2024b) have emerged as a promising direction for combinatorial optimization, particularly for enhancing the generative capability of non-autoregressive methods. DIFUSCO (Sun & Yang, 2023) models the solution space as graph edges and nodes and employs a graph-based discrete diffusion process for solution generation, achieving competitive performance. Building on this framework, T2T (Li et al., 2023) introduces energy-guided denoising during inference to further improve solution quality. Fast-T2T (Li et al., 2024b) additionally incorporates consistency models to reduce denoising steps and inference time while maintaining high-quality solutions. Although these methods demonstrate the potential of diffusion models on relatively simple CO tasks such as the Traveling Salesman Problem (TSP) and Maximum Independent Set (MIS), they still struggle with complex multi-agent CO problems. This limitation stems from the inability of non-autoregressive diffusion frameworks to adapt heuristics dynamically during decoding as system states evolve. In contrast, the proposed method employs a state-guided diffusion policy to steer the denoising process, enabling higher-quality solutions under complex constraints.

3 Methodology

3.1 Cooperative Multi-Agent MDP Formulation

To efficiently solve the *multi-agent* CO, we model it as a cooperative multi-agent Markov Decision Process (MDP), in which multiple agents collaboratively select the next actions to execute. The MDP with M agents can be represented by a quadruple $(\mathcal{S}, \mathcal{A}, \mathcal{T}, R)$, where $\mathcal{S}$ denotes the state space, $\mathcal{A} = \mathcal{A}_1 \times \cdots \times \mathcal{A}_M$ represents the joint action space of M agents, $\mathcal{T}$ is the state transition function, and R is the shared reward function. At each decision step, the global state $s \in \mathcal{S}$ encodes the problem's configuration (e.g., the coordinates and demands of customers in routing problems), and the static or dynamic status of all agents (e.g., speed or used capacity). Each agent $i \in \{1, \cdots, M\}$ simultaneously selects an action $a_i \in \mathcal{A}_i$, forming a joint action $\mathbf{a} = (a_1, \cdots, a_M)$ according to a parameterized policy π_θ:

$$\mathbf{a} \sim \pi_\theta(\cdot \mid s), \quad \pi_\theta : \mathcal{S} \to \mathcal{A}_1 \times \cdots \times \mathcal{A}_M. \tag{1}$$

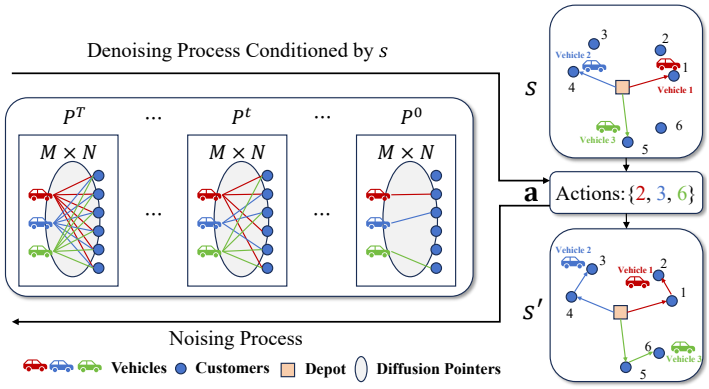

Figure 2: The Discrete Noising Process and the State-Conditioned Denoising Process over the multi-agent joint action space.

Here, the action a_i typically carries a problem-specific meaning in different tasks, e.g., representing the node that the vehicle i will select in the next step. The environment then transitions to the next state s' via the transition function τ.

It is worth noting that, unlike previous works that deterministically predict the action distribution (i.e., pointers) for action sampling, we instead train a diffusion policy that leverages the generative capability of multi-step denoising to generate actions.

3.2 Diffusion Modeling over the Multi-Agent Joint Action Space

Diffusion models (Ho et al., 2020; Austin et al., 2021) are typically defined by a forward noising process that gradually corrupts data by injecting noise step by step, along with a reverse denoising process that learns to progressively remove this noise to reconstruct samples from a simple *Uniform* or *Gaussian* distribution. To apply diffusion models in the joint action space, a straightforward approach is to consider a discrete joint action $\mathbf{a} \in \mathcal{A}$. Without loss of generality, we use the HCVRP with M vehicles (agents) and N customers as an illustrative example in the following discussion, and the action can be represented as $\mathbf{a} \in \{1, \cdots, N\}^M$. However, directly performing diffusion on this variable is challenging, as both the forward noising and reverse denoising processes inherently depend on the action space, i.e., the problem scale. To address this issue, we introduce an *Independence Assumption* in the joint action space and reformulate the action as a binary action matrix $A \in \{0,1\}^{M \times N}$, where $A_{i,j} = 1$ indicates that agent (vehicle) i selects action (node) j, and $A_{i,j} = 0$ otherwise. We then define the forward noising process and the denoising process over A, as shown in the Fig.2.

Discrete Noising Process Let $\bar{A} \in \{0,1\}^{M \times N \times 2}$ denote the one-hot representation of A, and let A^0 denote the initial high-quality joint action provided by experts. The forward noising process is defined as a conditional Markov chain that progressively corrupts the data, i.e., $q(A^t \mid A^{t-1}) = \mathrm{Cat}\big(A^t;\, p = \bar{A}^{t-1} Q^t\big)$, where $\mathrm{Cat}(\cdot)$ denotes the categorical distribution and $Q^t = \begin{bmatrix} 1-\beta^t & \beta^t \\ \beta^t & 1-\beta^t \end{bmatrix}$ is the transition probability matrix. Here, β^t represents the corruption ratio at diffusion step t. The forward process is designed such that $\prod_{t=1}^{T}(1-\beta^t) \approx 0$, ensuring that the terminal distribution approaches a uniform prior, i.e., $A^T \sim \mathrm{Uniform}(\cdot)$, independent of the initial A^0. Then, the t-step marginal distribution admits a closed-form expression: $q(A^t \mid A^0) = \mathrm{Cat}\big(A^t;\, p = \bar{A}^0 \bar{Q}^t\big)$, where $\bar{Q}^t = \prod_{i=1}^{t} Q^i$ denotes the cumulative transition matrix.

State-Conditioned Denoising Process In contrast to the noising process, the denoising process starts from A^T and progressively reconstructs the noising trajectory $A^{0:T}$ under the guidance of the current state s, which can be formulated as $p_\theta(A^{0:T}|s) = \prod_{t=1}^{T} p_\theta(A^{t-1}|A^t, s)$. To approximate this process, we employ a neural network to generate a special pointer $P^t : p_\theta(\hat{A}^0 \mid A^t, s)$ to predict A^0. We term $P^t \in [0,1]^{M \times N}$ *Diffusion Pointer*, which is actually defined as a two-dimensional distribution over the joint action space under an independence assumption, associating noisy joint action A^t with its corresponding clean action A^0. Accordingly, we have:

$$p_\theta(A^{t-1} \mid A^t, s) = \sum_{\hat{A}^0} q(A^{t-1} \mid A^t, \hat{A}^0)\, p_\theta(\hat{A}^0 \mid A^t, s), \tag{2}$$

where the posterior distribution can be obtained by Bayes' theorem:

$$q(A^{t-1} \mid A^t, A^0) = \frac{q(A^t \mid A^{t-1})\, q(A^{t-1} \mid A^0)}{q(A^t \mid A^0)} = \mathrm{Cat}\left(A^{t-1};\, p = \frac{\bar{A}^t [Q^t]^\top \odot \bar{A}^0 \bar{Q}^{t-1}}{\bar{A}^0 \bar{Q}^t [\bar{A}^t]^\top}\right). \quad (3)$$

Efficient Inference via Reduced Denoising Steps. At each decision step, the diffusion policy is required to generate actions, rendering the full T-step denoising process computationally expensive and often impractical. Therefore, reducing the number of inference steps is essential. In this work, we adopt tow optional classical approaches for accelerating the denoising process include Denoising Diffusion Implicit Models (DDIM) (Song et al., 2021) and Consistency Models (Song et al., 2023). Although derived from different assumptions and motivations, both methods practically rely on the Diffusion Pointer P^t when applied to *multi-agent* CO. Through these two methods, we reduce the full set of latent variables involved in the diffusion process, $A^{1:T}$, to a subset $[A^{\tau_1}, \cdots, A^{\tau_K}]$, where $[\tau_1, \cdots, \tau_K]$ is a subsequence of $[1, \cdots, T]$ of length K, satisfying $\tau_1 = 1$ and $\tau_K = T$. Due to space limitations, details of these two methods are provided in the Appendix A. Unless otherwise specified, we use Consistency Models by default.

Action Decoding. Given the diffusion pointer P^0 generated at the last step of the denoising process, a joint action **a** can be derived deterministically. Specifically, we first obtain candidate actions for each agent via a greedy strategy, defined as $\hat{a}_i = \arg\max_j (P^0_{i,j})$. Due to the overlap in the action spaces of multiple agents, conflicts may arise when different agents select the same action, resulting in infeasible solutions. Therefore, conflict resolution is necessary. Consistent with PARCO (Berto et al., 2025), we adopt a priority-based conflict handling mechanism. Agents involved in conflicts are ranked according to their priorities (i.e., action probabilities), where the agent with the highest priority is allowed to execute the action, while the remaining agents are prevented from acting.

3.3 Diffusion Pointer Networks for *multi-agent* CO

As discussed in Sections 3.1 and 3.2, solving *multi-agent* CO problems with DiffPtr involves two nested iterative procedures: an outer-loop Markov Decision Process for state transition and action execution, and an inner multi-step denoising process for action generation conditioned on the current state. Hence, network efficiency is crucial. Therefore, the proposed DiffPtr follows an encoder-decoder architecture. The encoder is executed only once and encodes static state features that remain invariant across diffusion and decision steps, producing the initial action embeddings $U \in \mathbb{R}^{N \times d}$ and agent embeddings $V \in \mathbb{R}^{M \times d}$, i.e., $f(\bar{s}) \mapsto (V, U)$, where $\bar{s}$ denotes the initial state. The decoder performs denoising conditioned on the dynamic state features, the diffusion step, and the noisy joint action, and outputs the Diffusion Pointer $P^t \in [0,1]^{M \times N}$, i.e., $g(s, V, U, A^t, t) \mapsto P^t$. In the following, we describe the implementation details of the encoder and decoder.

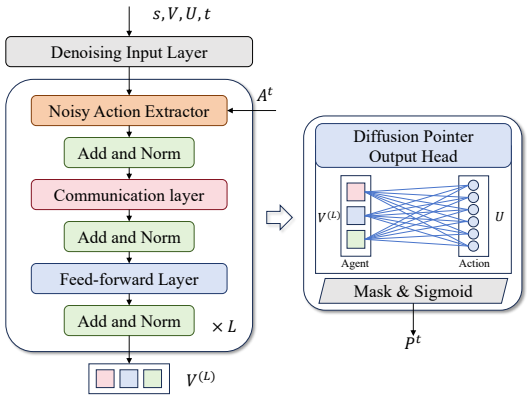

Figure 3: Architecture of the Diffusion Decoder.

3.3.1 Encoder

The encoder is executed only once throughout the entire decision-making procedure. Specifically, it utilizes the static agent features (e.g., speeds and maximum capacities of the vehicles) together with the action features (e.g., candidate customer nodes in routing problems) derived from the initial state $\bar{s}$ to construct reusable embeddings for agents and their actions. These embeddings remain invariant across both decision steps and diffusion steps. To map heterogeneous inputs into a unified representation space, two independent input projection layers are employed to transform the agent and action features into a shared d-dimensional latent space. Subsequently, a stack of L_e Transformer layers (Vaswani et al., 2017) is applied to capture complex interactions between agents and their actions (nodes) in a hierarchical manner. The initial agent and action embeddings are

first concatenated as $h^{(0)} = [h_v^{(0)}, h_u^{(0)}]$, and information exchange is performed via the multi-head attention mechanism within each Transformer layer, i.e., $\mathrm{MHA}(h^{(l)}, h^{(l)}, h^{(l)})$. The final agent embeddings $V \in \mathbb{R}^{M\times d}$ and action embeddings $U \in \mathbb{R}^{N\times d}$ can be extracted from $h^{(L_e)}$, which encode both the structural information of the problem and its underlying constraints.

3.3.2 Diffusion Decoder

Unlike the encoder, the Diffusion Decoder is executed at every decision step and diffusion step. It serves as the core component for capturing dynamic state features, performing denoising under different noise levels, and generating diffusion pointers. As illustrated in Fig.3, the proposed decoder consists of a denoising input layer, L stacked diffusion decoder layers, and a diffusion pointer output head.

Denoising Input Layer. For each agent i, the denoising input layer first extracts its dynamic features δ_i (e.g., the used capacities of vehicles) from the current state s, and subsequently retrieves static features by reusing its agent embedding V_i generated by the encoder. In addition, contextual information c (e.g., task completion status) is incorporated to help agents better assess the progress of the task. Furthermore, a noise-level embedding $e_t \in \mathbb{R}^d$ is injected into the encoding, which is defined as a sinusoidal function (Vaswani et al., 2017) of the diffusion step t, as shown in Eq. (5). The resulting representation for agent i is given by

$$V_i^{(0)} = \mathrm{Concat}(V_i, \delta_i, c)W_v + e_t W_e, \tag{4}$$

where W_v and W_e are learnable parameters. The noise-level embedding is defined as

$$e_t = \left[\sin\frac{t}{T^{\frac{0}{d}}}, \cos\frac{t}{T^{\frac{0}{d}}}, \sin\frac{t}{T^{\frac{2}{d}}}, \cos\frac{t}{T^{\frac{2}{d}}}, \cdots, \sin\frac{t}{T^{\frac{d}{d}}}, \cos\frac{t}{T^{\frac{d}{d}}}\right]. \tag{5}$$

Diffusion Decoder Layer We design a stack of L Diffusion Decoder Layers to process the initial agent embeddings $V^{(0)} \in \mathbb{R}^{M\times d}$ produced by the input layer, perform deep denoising, and prepare for the generation of diffusion pointers. The computation process of the l-th layer is formulated as follows:

$$\bar{V}^{(l)} = \mathrm{Norm}\left(V^{(l)} + \mathrm{NAE}\left(V^{(l)}, U, U, A^t\right)\right), \tag{6}$$

$$\bar{\bar{V}}^{(l)} = \mathrm{Norm}\left(\bar{V}^{(l)} + \mathrm{Communication}(\bar{V}^{(l)})\right), \tag{7}$$

$$V^{(l+1)} = \mathrm{Norm}\left(\bar{\bar{V}}^{(l)} + \mathrm{FFN}\left(\bar{\bar{V}}^{(l)}\right)\right). \tag{8}$$

As shown in Eq.6, we first inject noisy action information into the agent embeddings $V^{(l)}$ through a Noisy Action Extractor, i.e., $\mathrm{NAE}(\cdot)$, which is implemented via a simple multi-head attention mechanism with bias. Specifically, based on the noisy joint action A^t , we construct an attention bias matrix $Z \in \{-1, 0, 1\}^{M\times N}$ to modulate the attention scores α, analogous to the additional scores introduced by edge features in some Graph Neural Networks (Wu et al., 2020), as formulated below:

$$\alpha_{i,j} = \frac{(V_i^{(l)} W_q [U_j W_k]^T)}{\sqrt{d_k}} + Z_{i,j}, \text{where } Z_{i,j} = \begin{cases} -1, & \mathrm{Illegal}(i,j), \\ A_{i,j}^t, & \text{otherwise.} \end{cases} \tag{9}$$

Here, the function $\mathrm{Illegal}(i, j)$ indicates that action j is infeasible for agent i, e.g., violating vehicle capacity constraints or precedence constraints of Pickup-and-Delivery problems. Unlike prior works (Berto et al., 2025) that completely eliminate the influence of infeasible actions through masking operations, we instead softly suppress their attention weights via a negative bias (e.g., -1). This design is motivated by the observation that such actions may become feasible again in future decision steps and could therefore provide useful information for long-term planning. For feasible actions, a binary bias (1 or 0) is further employed to softly amplify or attenuate the weights of selected and unselected actions in A^t, respectively, since the action selection states may still be reversed during the subsequent denoising steps.

Subsequently, a Communication Layer (i.e., $\mathrm{Communication}(\cdot)$ in Eq.7) based a simple self-attention, is employed to facilitate information exchange among agents, followed by a two-layer Feed-forward Layer (i.e., $\mathrm{FFN}(\cdot)$ in Eq.8) with ReLU activation for deep feature modeling. In addition, Transformer-style inter-layer architectures, namely residual connections and normalization operations, are further introduced to improve gradient stability and accelerate convergence during training.

Diffusion Pointer Output Head. Given the agent embeddings $V_i^{(L)}$ and the action embeddings $U \in \mathbb{R}^{N \times d}$ from the encoder, The diffusion pointer is then computed as follows:

$$P_{i,j}^t = \begin{cases} 0, & \text{Illegal}(i,j), \\ \sigma\left(\frac{V_i^{(L)} W U_j^\top}{\sqrt{d}}\right), & \text{otherwise.} \end{cases} \tag{10}$$

where $W \in \mathbb{R}^{d \times d}$ is a learnable parameter matrix, $\sigma(\cdot)$ denotes the sigmoid function, and the final probabilities of invalid actions are reset to zero.

3.4 Training

We train DiffPtr via Imitation Learning (IL) (Zare et al., 2024) using an expert trajectory dataset $\mathcal{D}$. Specifically, we randomly sample state–action pairs $(s, \mathbf{a})$ from $\mathcal{D}$ and uniformly select a diffusion timestep $t \in \{1 \cdots T\}$. The ground-truth action $\mathbf{a}$ is first converted into its binary-valued action representation A^0, which is then perturbed according to the forward diffusion process to obtain A^t. Conditioned on the state s and A^t, DiffPtr predicts a denoised diffusion pointer $P^t : p_\theta(\hat{A}^0 | A^t, s)$. The model parameters are optimized by minimizing the cross-entropy loss between the predicted distribution P^t and the original clean A^0, using stochastic gradient descent until convergence. Due to the high computational complexity of the *multi-agent* CO, constructing high-quality expert data is non-trivial. We therefore use the state-of-the-art PARCO solver to generate randomized expert trajectories, while our framework can also leverage data from heuristics or other neural solvers.

4 Experiments

To evaluate the effectiveness of the proposed DiffPtr, we conduct extensive experiments on Multi-agent CO problems: the *Heterogeneous Capacitated Vehicle Routing Problem* (HCVRP) (Liu et al., 2024) and the *Open Multi-Depot Capacitated Pickup and Delivery Problem* (OMDCPDP) (Berto et al., 2025). These benchmarks involve cooperative decision-making among multiple agents under complex operational constraints and are widely adopted for assessing multi-agent combinatorial optimization methods. Detailed descriptions of all experimental settings are provided in the Appendix C.

HCVRP. This problem aims to determine optimal routes for a fleet of heterogeneous vehicles (agents) with varying speeds and capacities. These vehicles are required to serve a set of customers with diverse demands at different locations while minimizing the overall service completion time under a min–max objective. ***Baselines***. We consider three classical heuristic solvers: 1) SISR (Christiaens & Vanden Berghe, 2020), 2) Genetic Algorithm (GA) (Karakatič & Podgorelec, 2015), and 3) Simulated Annealing (SA) (İlhan, 2021). And for learning-based neural solvers, we include 4) Attention Model(AM) (Kool et al., 2019), 5) Equity Transformer (ET) (Son et al., 2024), 6) Decoupling Partition and Navigation (DPN)(Zheng et al., 2024), 7) DRL$_{Li}$ (Li et al., 2022), and 8) 2D-Ptr (Liu et al., 2024), which employs a two-dimensional pointer mechanism to jointly select agents and actions, as well as 9) the state-of-the-art PARCO (Berto et al., 2025), an autoregressive framework that adopts a multi-pointer mechanism to support parallel decision-making across agents.

OMDCPDP. The OMDCPDP is a collaborative pickup-and-delivery problem where multiple agents start from different depots to serve transportation requests. Unlike standard closed-route settings, agents are not required to return to their depots after completing deliveries. During execution, vehicles must satisfy dynamic capacity constraints as pickup and delivery operations change their loads. The objective is to minimize total delivery delay, measured by the sum of arrival times of all completed deliveries. ***Baselines***. We adopt 1) OR-Tools as the primary classical optimization baseline. In addition, We also compare our approach with representative neural methods for pickup-and-delivery problem, including the 2) HAM (Heterogeneous Attention Model) (Li et al., 2021), 3) MAPDP (Zong et al., 2022) and 4) the state-of-the-art PARCO (Berto et al., 2025).

4.1 Main Results

Tab.1 and Tab.2 present the main results for the HCVRP and OMDCPDP problems, respectively, including the average objective value (Obj.), the gap to the best-known solution, and the average

Table 1: Main results on HCVRP for in-distribution and out-of-distribution ($10\times$ larger than those seen during training) generalization. For all metrics, the lower the better (↓).

In-distribution																		
	N=60									N=100								
	M=3			M=5			M=7			M=3			M=5			M=7		
Metric	Obj.	Gap	Time	Obj.	Gap	Time	Obj.	Gap	Time	Obj.	Gap	Time	Obj.	Gap	Time	Obj.	Gap	Time
SISRs	6.57	0.00%	271s	4.00	0.00%	274s	2.91	0.00%	276s	10.29	0.00%	615s	6.17	0.00%	623s	4.45	0.00%	625s
GA	9.21	40.18%	233s	6.89	72.25%	320s	5.98	105.50%	405s	15.33	48.98%	479s	10.93	77.15%	623s	9.10	104.49%	772s
SA	7.04	7.15%	130s	4.39	9.75%	289s	3.30	13.40%	362s	11.13	8.16%	434s	6.80	10.21%	557s	5.01	12.58%	678s
AM	7.62	15.98%	0.14s	4.82	20.50%	0.13s	3.63	24.74%	0.14s	11.82	14.87%	0.29s	7.45	20.75%	0.28s	5.58	25.39%	0.28s
ET	7.14	8.68%	0.21s	4.46	11.50%	0.22s	3.33	14.43%	0.22s	11.20	8.84%	0.41s	6.85	11.02%	0.38s	4.98	11.91%	0.40s
DPN	7.08	7.76%	0.25s	4.35	8.75%	0.28s	3.20	9.97%	0.38s	11.04	7.29%	0.48s	6.66	7.94%	0.52s	4.79	7.64%	0.78s
DRL$_{Li}$	6.97	6.09%	0.30s	4.34	8.50%	0.36s	3.25	11.68%	0.43s	10.90	5.93%	0.60s	6.65	7.78%	0.76s	4.98	11.91%	0.92s
2D-Ptr	6.82	3.81%	0.13s	4.20	5.00%	0.13s	3.09	6.19%	0.14s	10.71	4.08%	0.22s	6.46	4.70%	0.23s	4.68	5.17%	0.24s
PARCO	6.82	3.81%	0.05s	4.17	4.25%	0.05s	3.06	5.15%	0.07s	10.61	3.11%	0.08s	6.36	3.08%	0.08s	4.58	2.92%	0.09s
DiffPtr	**6.77**	**3.08**%	0.24s	**4.13**	**3.35**%	0.35s	**3.02**	**3.82**%	0.48s	**10.59**	**2.88**%	0.49s	**6.34**	**2.74**%	0.70s	**4.57**	**2.66**%	0.91s
Out-of-distribution																		
	N=500									N=1000								
	M=30			M=50			M=70			M=30			M=50			M=70		
Metric	Obj.	Gap	Time	Obj.	Gap	Time	Obj.	Gap	Time	Obj.	Gap	Time	Obj.	Gap	Time	Obj.	Gap	Time
AM	6.14	26.47%	0.13s	4.39	35.96%	0.14s	3.64	41.78%	0.15s	13.95	52.46%	0.53s	7.14	27.19%	0.51s	5.39	29.66%	0.52s
2D-Ptr	5.05	3.92%	0.64s	3.44	6.74%	0.88s	2.89	12.83%	1.11s	10.40	13.67%	2.40s	5.96	6.19%	3.10s	4.37	4.97%	3.88s
PARCO	5.16	6.32%	0.22s	3.59	11.13%	0.33s	3.09	20.41%	0.41s	10.49	14.59%	0.67s	6.84	21.82%	0.90s	5.51	32.37%	1.11s
DiffPtr	**4.85**	**0.00%**	1.71s	**3.23**	**0.00%**	2.38s	**2.57**	**0.00%**	2.81s	**9.15**	**0.00%**	5.40s	**5.61**	**0.00%**	8.14s	**4.16**	**0.00%**	10.41s

Table 2: Main results on OMDCPDP for in-distribution and out-of-distribution generalization.

In-distribution																		
	N=50									N=100								
	M=5			M=7			M=10			M=10			M=15			M=20		
Metric	Obj.	Gap	Time	Obj.	Gap	Time	Obj.	Gap	Time	Obj.	Gap	Time	Obj.	Gap	Time	Obj.	Gap	Time
OR-Tools	37.61	5.99%	30s	30.18	4.64%	30s	24.48	2.89%	30s	66.78	5.74%	60s	51.49	3.79%	60s	43.90	2.20%	60s
HAM	36.26	2.18%	1.18s	29.54	2.42%	1.31s	27.77	16.72%	1.37s	66.91	5.94%	2.33s	51.60	4.01%	2.53s	52.24	21.61%	2.72s
MAPDP	35.64	0.44%	0.02s	29.03	0.65%	0.02s	23.97	0.75%	0.01s	63.64	0.77%	0.03s	50.07	0.93%	0.03s	43.57	1.43%	0.02s
PARCO	35.63	0.42%	0.03s	28.97	0.43%	0.03s	23.89	0.39%	0.03s	63.20	0.07%	0.05s	49.69	0.17%	0.04s	43.08	0.29%	0.04s
DiffPtr	**35.48**	**0.00**%	0.07s	**28.84**	**0.00**%	0.08s	**23.79**	**0.00**%	0.08s	**63.16**	**0.00**%	0.17s	**49.61**	**0.00**%	0.21s	**42.96**	**0.00**%	0.22s
Out-of-distribution																		
	N=500									N=1000								
	M=50			M=75			M=100			M=100			M=150			M=200		
Metric	Obj.	Gap	Time	Obj.	Gap	Time	Obj.	Gap	Time	Obj.	Gap	Time	Obj.	Gap	Time	Obj.	Gap	Time
OR-Tools	290.79	10.28%	300s	223.72	7.78%	300s	192.59	4.61%	300s	780.89	54.35%	600s	642.54	61.72%	600s	584.27	64.33%	600s
HAM	409.67	55.37%	1.19s	305.16	47.01%	1.31s	203.50	10.54%	1.39s	708.55	40.05%	2.94s	552.76	39.13%	3.22s	384.10	8.03%	3.46s
PARCO	265.63	0.74%	0.29s	208.54	0.47%	0.42s	185.07	0.53%	0.53s	518.16	2.42%	0.85s	404.84	1.90%	0.97s	360.69	1.45%	1.09s
DiffPtr	**263.68**	**0.00**%	1.10s	**207.57**	**0.00**%	1.30s	**184.10**	**0.00**%	1.38s	**505.91**	**0.00**%	3.97s	**397.31**	**0.00**%	4.85s	**355.54**	**0.00**%	5.17s

solving time under both in-distribution and out-of-distribution settings, across different problem sizes N and agent numbers M. For neural solvers, we report the sampling performance based on 64 sampled solutions for out-of-distribution setting of HCVRP and 1280 for others settings.

In the in-distribution setting, DiffPtr achieves the lowest Gap in most configurations. For example, under the $N = 60$ and $M = 7$ setting for HCVRP, DiffPtr obtains a Gap of 3.82%, outperforming PARCO (5.15%) and 2D-Ptr (6.19%). Meanwhile, its objective values are consistently better than existing learning-based neural solvers, indicating that the diffusion model can effectively capture collaborative behaviors among multiple agents and generate higher-quality joint action sequences. The advantage of DiffPtr becomes more significant in the out-of-distribution setting. Although many neural solvers struggle to generalize in a zero-shot manner to problem instances that are $10\times$ larger than those seen during training, DiffPtr still demonstrates stable generation quality and strong generalization capability.

4.2 Analysis

Diffusion vs. Non-Diffusion under IL. Since DiffPtr is trained under the imitation learning (IL) paradigm, it is necessary to determine whether the performance improvement mainly comes from the diffusion or the IL training scheme. Therefore, we design an ablation study, in which we keep the expert dataset $\mathcal{D}$ and all training hyperparameters unchanged, train a non-diffusion model under the same IL setting, and then compare its performance with DiffPtr. The results shown in Fig.4

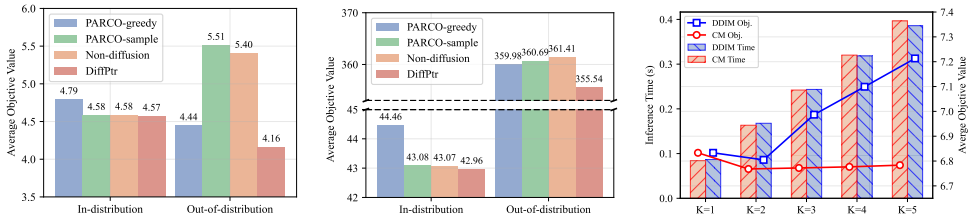

Figure 4: Ablation results. The *left* and *middle* panels compare the performance of DiffPtr, non-diffusion models, and their teacher model PARCO on HCVRP and OMDCPDP, respectively. The *right* panel shows the impact of diffusion steps and acceleration methods on performance and speed.

reveals an interesting phenomenon: under the out-of-distribution setting, the teacher (PARCO) suffers a clear degradation in sampling ability, where multiple samples perform substantially worse than a single greedy generation. Non-diffusion models trained with IL inherit this behavior, whereas our diffusion-based DiffPtr breaks this limitation and achieves the best performance, highlights the advantages of diffusion-based pointers compared with deterministic pointers.

Effect of Different Acceleration Methods. We evaluate two inference acceleration methods introduced in Section 3.2: DDIM and the Consistency Model (CM). Considering the trade-off between efficiency and solution quality, We recommend using a small number of inference steps, e.g. $K \leq 5$. As shown in Fig.4, DDIM and CM exhibit comparable inference time that increases linearly with K, while CM significantly outperforms DDIM in performance. A possible reason is that the consistency constraint enables the model to better preserve the denoising trajectory, making it more suitable for low-step generation scenarios. Additionally, The performance initially improves and then deteriorates as K increases. This phenomenon may be caused by exposure bias accumulated during the iterative denoising process of the diffusion model.

Table 3: Ablation results about Diffusion Decoder on OMDCPDP.

	$N = 100, M = 20$			$N = 1000, M = 200$		
Metric	Obj.	Gap	Time	Obj.	Gap	Time
w/o Comm.	43.12	0.37%	0.20s	358.74	0.90%	4.77s
w/o NAE	43.19	0.54%	0.18s	359.93	1.24%	5.93s
w/o FFN	42.98	0.05%	0.19s	360.54	1.41%	9.96s
DiffPtr	**42.96**	**0.00%**	0.22s	**355.54**	**0.00%**	5.17s

Effect of Diffusion Decoder. To investigate the role of each component in the proposed diffusion decoder, we construct several variants by removing specific modules, including 1) w/o Comm.(removing the communication layer), 2) w/o FFN (removing the FFN layer), and 3) w/o NAE (removing the Noisy Action Extractor, and using mean-pooling to incorporate denoised action information). The experimental results are shown in Tab.3. All variants exhibit performance degradation to different extents, demonstrating the effectiveness of each component in the proposed architecture.

More Experiments Results. Due to space limitations, additional experiments and discussions are provided in the Appendix D.

5 Conclusion

We propose Diffusion Pointer Network (DiffPtr), a diffusion-based policy designed for solving multi-agent combinatorial optimization (CO) problems by modeling the multimodal distribution over the joint action space. By introducing a diffusion-pointer-based denoising process together with a hierarchical diffusion decoder, DiffPtr is capable of capturing the multimodal characteristics of the joint action space while improving out-of-distribution generation stability. Extensive experiments on both the HCVRP and OMDCPDP benchmarks demonstrate the superior solution quality and generalization capability of DiffPtr, highlighting the potential of diffusion policies for addressing complex multi-agent CO problems. **Limitations & Future Work.** Although DiffPtr achieves substantially higher efficiency than heuristic algorithms, the multi-step denoising iterations result in slower inference speed compared with non-diffusion neural solvers. Moreover, the exposure bias introduced by iterative denoising also negatively affects solution quality, and the dependence on expert data further constrains the performance of DiffPtr. In future work, we aim to further improve the inference efficiency of DiffPtr via one-shot multi-step action generation, while also exploring approaches to alleviate exposure bias and the Reinforcement Learning training framework.

AI USE STATEMENT

We used large language models (e.g., ChatGPT) as general-purpose assistants for (i) language polishing of the manuscript, including grammar, wording, and clarity, and (ii) suggesting occasional non-substantive code snippets. LLMs were not used for problem formulation, algorithm or model design, experimental design or analysis, data generation, or drawing conclusions. All technical content and code were developed and verified by the authors, and all LLM-assisted text and code suggestions were reviewed and edited by the authors. We take full responsibility for the final content of this work.

REFERENCES

Jacob Austin, Daniel D Johnson, Jonathan Ho, Daniel Tarlow, and Rianne Van Den Berg. Structured denoising diffusion models in discrete state-spaces. *Advances in neural information processing systems*, 34:17981–17993, 2021.

Irwan Bello, Hieu Pham, Quoc V Le, Mohammad Norouzi, and Samy Bengio. Neural combinatorial optimization with reinforcement learning. *arXiv preprint arXiv:1611.09940*, 2016.

Federico Berto, Chuanbo Hua, Laurin Luttmann, Jiwoo Son, Junyoung Park, Kyuree Ahn, Changhyun Kwon, Lin Xie, and Jinkyoo Park. PARCO: Parallel AutoRegressive Models for Multi-Agent Combinatorial Optimization. In *Advances in Neural Information Processing Systems*, 2025. URL https://github.com/ai4co/parco.

Jieyi Bi, Zhiguang Cao, Jianan Zhou, Wen Song, Yaoxin Wu, Jie Zhang, Yining Ma, and Cathy Wu. Towards efficient constraint handling in neural solvers for routing problems. *arXiv preprint arXiv:2602.16012*, 2026.

Haw-Shiuan Chang and Andrew McCallum. Softmax bottleneck makes language models unable to represent multi-mode word distributions. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 8048–8073, 2022.

Peng Chen, Jing Liang, Kang-Jia Qiao, Hui Song, Cai-Tong Yue, Tian-Lei Ma, Kun-Jie Yu, Ponnuthurai Nagaratnam Suganthan, and Witold Pedrycz. An adaptive multi-objective task splitting algorithm for agricultural multi-robot task allocation. *IEEE Transactions on Evolutionary Computation*, 2026.

Cheng Chi, Zhenjia Xu, Siyuan Feng, Eric Cousineau, Yilun Du, Benjamin Burchfiel, Russ Tedrake, and Shuran Song. Diffusion policy: Visuomotor policy learning via action diffusion. *The International Journal of Robotics Research*, 44(10-11):1684–1704, 2025.

Jan Christiaens and Greet Vanden Berghe. Slack induction by string removals for vehicle routing problems. *Transportation Science*, pp. 417–433, Mar 2020. doi: 10.1287/trsc.2019.0914. URL http://dx.doi.org/10.1287/trsc.2019.0914.

Zhang-Hua Fu, Kai-Bin Qiu, and Hongyuan Zha. Generalize a small pre-trained model to arbitrarily large tsp instances. In *Proceedings of the AAAI conference on artificial intelligence*, volume 35, pp. 7474–7482, 2021.

Yifan Guo, Zhongqiang Ren, and Chen Wang. imtsp: Solving min-max multiple traveling salesman problem with imperative learning. In *2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 10245–10252. IEEE, 2024.

Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. *Advances in neural information processing systems*, 33:6840–6851, 2020.

Chuanbo Hua, Federico Berto, Zhikai Zhao, Jiwoo Son, Changhyun Kwon, and Jinkyoo Park. Uspr: Learning a unified solver for profiled routing. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pp. 36973–36981, 2026.

İLHAN İlhan. An improved simulated annealing algorithm with crossover operator for capacitated vehicle routing problem. *Swarm and Evolutionary Computation*, 64:100911, 2021.

Yan Jin, Yuandong Ding, Xuanhao Pan, Kun He, Li Zhao, Tao Qin, Lei Song, and Jiang Bian. Pointerformer: Deep reinforced multi-pointer transformer for the traveling salesman problem. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, pp. 8132–8140, 2023.

Bingyi Kang, Xiao Ma, Chao Du, Tianyu Pang, and Shuicheng Yan. Efficient diffusion policies for offline reinforcement learning. *Advances in Neural Information Processing Systems*, 36: 67195–67212, 2023.

Sašo Karakatič and Vili Podgorelec. A survey of genetic algorithms for solving multi depot vehicle routing problem. *Applied Soft Computing*, 27:519–532, 2015.

Minsu Kim, Junyoung Park, and Jinkyoo Park. Sym-nco: Leveraging symmetricity for neural combinatorial optimization. *Advances in Neural Information Processing Systems*, 35:1936–1949, 2022.

W Kool, HV Hoof, and M Welling. Attention, learn to solve routing problems! iclr, 2019.

Wouter Kool, Herkevan Hoof, and Max Welling. Attention, learn to solve routing problems! *arXiv: Machine Learning,arXiv: Machine Learning*, Mar 2018.

Yeong-Dae Kwon, Jinho Choo, Byoungjip Kim, Iljoo Yoon, Youngjune Gwon, and Seung-Jai Min. Pomo: Policy optimization with multiple optima for reinforcement learning. *Neural Information Processing Systems,Neural Information Processing Systems*, Jan 2020.

Jingwen Li, Liang Xin, Zhiguang Cao, Andrew Lim, Wen Song, and Jie Zhang. Heterogeneous attentions for solving pickup and delivery problem via deep reinforcement learning. *IEEE Transactions on Intelligent Transportation Systems*, 23(3):2306–2315, 2021.

Jingwen Li, Yining Ma, Ruize Gao, Zhiguang Cao, Andrew Lim, Wen Song, and Jie Zhang. Deep reinforcement learning for solving the heterogeneous capacitated vehicle routing problem. *IEEE Transactions on Cybernetics*, pp. 13572–13585, Dec 2022. doi: 10.1109/tcyb.2021.3111082. URL http://dx.doi.org/10.1109/tcyb.2021.3111082.

Jingwen Li, Yining Ma, Zhiguang Cao, Yaoxin Wu, Wen Song, Jie Zhang, and Yeow Meng Chee. Learning feature embedding refiner for solving vehicle routing problems. *IEEE Transactions on Neural Networks and Learning Systems*, 35(11):15279–15291, 2024a. doi: 10.1109/TNNLS.2023.3285077.

Yang Li, Jinpei Guo, Runzhong Wang, and Junchi Yan. T2t: From distribution learning in training to gradient search in testing for combinatorial optimization. *Advances in Neural Information Processing Systems*, 36:50020–50040, 2023.

Yang Li, Jinpei Guo, Runzhong Wang, Hongyuan Zha, and Junchi Yan. Fast t2t: Optimization consistency speeds up diffusion-based training-to-testing solving for combinatorial optimization. *Advances in Neural Information Processing Systems*, 37:30179–30206, 2024b.

Yang Li, Lvda Chen, Haonan Wang, Runzhong Wang, and Junchi Yan. Generation as search operator for test-time scaling of diffusion-based combinatorial optimization. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025.

Federico Liberatore, M Teresa Ortuño, Gregorio Tirado, Begoña Vitoriano, and Maria Paola Scaparra. A hierarchical compromise model for the joint optimization of recovery operations and distribution of emergency goods in humanitarian logistics. *Computers & Operations Research*, 42:3–13, 2014.

Qidong Liu, Chaoyue Liu, Shaoyao Niu, Cheng Long, Jie Zhang, and Mingliang Xu. 2d-ptr: 2d array pointer network for solving the heterogeneous capacitated vehicle routing problem. In *Proceedings of the 23rd International Conference on Autonomous Agents and Multiagent Systems*, pp. 1238–1246, 2024.

Qidong Liu, Jiurui Lian, Chaoyue Liu, and Zhiguang Cao. Enhancing generalization in large-scale hcvrp: A rank-augmented neural solver. In *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining V. 2*, pp. 1845–1856, 2025.

Alexander Quinn Nichol and Prafulla Dhariwal. Improved denoising diffusion probabilistic models. In *International conference on machine learning*, pp. 8162–8171. PMLR, 2021.

Junyoung Park, Changhyun Kwon, and Jinkyoo Park. Learn to solve the min-max multiple traveling salesmen problem with reinforcement learning. In *AAMAS*, volume 22, pp. 878–886, 2023.

Fernando Peres and Mauro Castelli. Combinatorial optimization problems and metaheuristics: Review, challenges, design, and development. *Applied sciences*, 11(14):6449, 2021.

Jiwoo Son, Minsu Kim, Sanghyeok Choi, Hyeonah Kim, and Jinkyoo Park. Solving np-hard min-max routing problems as sequential generation with equity context. In *ICML 2023 Workshop: Sampling and Optimization in Discrete Space*, 2023.

Jiwoo Son, Minsu Kim, Sanghyeok Choi, Hyeonah Kim, and Jinkyoo Park. Equity-transformer: Solving np-hard min-max routing problems as sequential generation with equity context. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pp. 20265–20273, 2024.

Jiaming Song, Chenlin Meng, and Stefano Ermon. Denoising diffusion implicit models. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021. URL https://openreview.net/forum?id=St1giarCHLP.

Yang Song, Prafulla Dhariwal, Mark Chen, and Ilya Sutskever. Consistency models. In Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett (eds.), *International Conference on Machine Learning, ICML 2023, 23-29 July 2023, Honolulu, Hawaii, USA*, Proceedings of Machine Learning Research, pp. 32211–32252. PMLR, 2023. URL https://proceedings.mlr.press/v202/song23a.html.

Zhiqing Sun and Yiming Yang. Difusco: Graph-based diffusion solvers for combinatorial optimization. *Advances in neural information processing systems*, 36:3706–3731, 2023.

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

Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. Pointer networks. *Advances in neural information processing systems*, 28, 2015.

Yaoxin Wu, Wen Song, Zhiguang Cao, Jie Zhang, and Andrew Lim. Learning improvement heuristics for solving routing problems. *IEEE Transactions on Neural Networks,IEEE Transactions on Neural Networks*, Dec 2019.

Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S Yu. A comprehensive survey on graph neural networks. *IEEE transactions on neural networks and learning systems*, 32(1):4–24, 2020.

Zhilin Yang, Thang Luong, Russ R Salakhutdinov, and Quoc V Le. Mixtape: Breaking the softmax bottleneck efficiently. *Advances in Neural Information Processing Systems*, 32, 2019.

Maryam Zare, Parham M Kebria, Abbas Khosravi, and Saeid Nahavandi. A survey of imitation learning: Algorithms, recent developments, and challenges. *IEEE Transactions on Cybernetics*, 54 (12):7173–7186, 2024.

Hao Zeng, Jiaqi Wang, Avirup Das, Junying He, Kunpeng Han, Haoyuan Hu, and Mingfei Sun. Effective generation of feasible solutions for integer programming via guided diffusion. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pp. 4107–4118, 2024.

Zhi Zheng, Shunyu Yao, Zhenkun Wang, Xialiang Tong, Mingxuan Yuan, and Ke Tang. Dpn: Decoupling partition and navigation for neural solvers of min-max vehicle routing problems. *arXiv preprint arXiv:2405.17272*, 2024.

Zefang Zong, Meng Zheng, Yong Li, and Depeng Jin. Mapdp: Cooperative multi-agent reinforcement learning to solve pickup and delivery problems. In *Proceedings of the AAAI conference on artificial intelligence*, volume 36, pp. 9980–9988, 2022.

A Fast Inference Procedure

A.1 DDIM

Denoising Diffusion Implicit Models (DDIM) (Song et al., 2021) is an efficient generative method based on diffusion models, proposed as an improvement over the sampling process of traditional diffusion probabilistic models. Compared with DDPM, which typically requires a large number of iterative denoising steps, DDIM employs a deterministic or semi-deterministic non-Markovian sampling strategy, enabling high-quality sample generation with significantly fewer sampling steps. This greatly improves inference efficiency while maintaining strong generation quality.

Formally, DDIM accelerates the diffusion sampling procedure by replacing the original full diffusion trajectory $A^{1:T}$ with a sparse subsequence of latent states $[A^{\tau_1}, \dots, A^{\tau_K}]$, where $[\tau_1, \dots, \tau_K] \subseteq [1, \dots, T]$, $K \ll T$, and the boundary conditions satisfy $\tau_1 = 1$ and $\tau_K = T$. Instead of performing denoising at every timestep, DDIM only evaluates several selected timesteps, thereby substantially reducing the computational cost during inference while preserving the overall generation quality.

Under this formulation, the marginal distribution at timestep τ_i is defined as

$$q(A^{\tau_i} \mid A^0) = \mathrm{Cat}\big(A^{\tau_i};\, p = \bar{A}^0 \bar{Q}^{\tau_i}\big), \tag{11}$$

where $\bar{Q}^{\tau_i}$ denotes the accumulated transition matrix up to timestep τ_i. This formulation indicates that the noisy latent variable A^{τ_i} can be directly sampled from the clean state A^0 without explicitly traversing all intermediate diffusion steps.

Similar to Eq.(3), the reverse denoising transition can be expressed as

$$p_\theta(A^{\tau_{i-1}} \mid A^{\tau_i}, s) = \sum_{\hat{A}^0} q(A^{\tau_{i-1}} \mid A^{\tau_i}, \hat{A}^0)\, p_\theta(\hat{A}^0 \mid A^{\tau_i}, s), \tag{12}$$

where the model first predicts the clean latent state $\hat{A}^0$ conditioned on the current noisy state and condition s, and then reconstructs the previous latent variable $A^{\tau_{i-1}}$. Compared with the standard DDPM formulation, this sparse reverse process enables more efficient inference by skipping redundant denoising steps.

Furthermore, the conditional transition distribution between two adjacent selected timesteps is given by

$$q(A^{\tau_{i-1}} \mid A^{\tau_i}, A^0) = \mathrm{Cat}\left(A^{\tau_{i-1}};\, p = \frac{\bar{A}^{\tau_i} (Q^{\tau_{i-1}, \tau_i})^\top \odot \bar{A}^0 \bar{Q}^{\tau_{i-1}}}{\bar{A}^0 \bar{Q}^{\tau_i} (\bar{A}^{\tau_i})^\top}\right), \tag{13}$$

where $\bar{Q}^{\tau_{i-1}, \tau_i} = \prod_{t=\tau_{i-1}}^{\tau_i} Q^t$.

This expression characterizes the probabilistic relationship between two nonconsecutive diffusion states and allows DDIM to perform implicit deterministic-style sampling across larger timestep intervals. Consequently, the model achieves significantly faster sampling while maintaining consistency with the original diffusion objective.

A.2 Consistency Models

Consistency Models (Song et al., 2023) are a class of generative models designed to directly map noisy samples to clean data distributions through a consistency transformation, i.e., 1) Boundary condition: $f(x_\varepsilon, \varepsilon) = x_\varepsilon$ and 2) Self-consistency property: $f_\theta(x_t, t) = f_\theta(x_{t'}, t'), \forall\, t, t' \in [\varepsilon, T]$. Unlike traditional DDPM or DDIM that require multiple iterative denoising steps during inference, consistency models learn a time-consistent mapping such that samples at different noise levels are projected toward the same clean target. As a result, they can generate high-quality samples in one or only a few inference steps, significantly improving generation efficiency.

In this paper, we consider Optimizing Consistency (Li et al., 2024b), namely enforcing:

$$p_\theta(\hat{A}^0 \mid A^t, s) = p_\theta(\hat{A}^0 \mid A^{t'}, s) = \delta(\hat{A}^0 - A^0), \tag{14}$$

where $\delta(\cdot)$ denotes the Dirac distribution. Although different timesteps t and t' are not explicitly sampled within a single training step, the expert dataset $\mathcal{D}$ consists of independently and identically

Algorithm 1 Multistep Consistency Sampling

Require: Consistency model p_θ, current state s, sequence of timesteps $[\tau_1 > \tau_2 > \cdots > \tau_K]$
1: Sample A^T from uniform distribution $\mathcal{U}$
2: $A^0 \sim p_\theta(\hat{A}^0 \mid A^T, s)$
3: **for** $i = 1$ to $K - 1$ **do**
4: Sample $A^{\tau_i} \sim \mathrm{Cat}\big(A^{\tau_i};\, p = A^0 \bar{Q}^{\tau_i}\big)$
5: $A^0 \sim p_\theta(\hat{A}^0 \mid A^{\tau_i}, s)$
6: **end for**
7: **Output:** Final A_0

distributed trajectories generated by an expert policy. Consequently, across multi-step and repeated sampling during training, the model is optimized toward the same objective, thereby implicitly achieving alignment across different timesteps.

During inference, the consistency model can further enhance generation quality through *Multistep Consistency Sampling*. As shown in Algorithm 1, the denoising process relies on a subset of the full timesteps, $[\tau_1, \ldots, \tau_K] \subseteq [1, \ldots, T], \quad K \ll T$. At each step, a prediction of A^0 is first sampled from $p_\theta(\hat{A}^0 \mid A^{\tau_i}, s)$, and is subsequently re-noised to obtain $A^{\tau_{i-1}}$. This procedure is iteratively repeated until the inference process terminates.

B Problem Definitions

B.1 HCVRP

The Heterogeneous Capacitated Vehicle Routing Problem (HCVRP) considers a fleet of heterogeneous vehicles dispatched from a central depot to serve geographically distributed customers. Let an instance be defined as a tuple $(\mathcal{V}, \mathcal{N})$, where $\mathcal{V} = \{1, 2, \cdots, M\}$ denotes a set of M vehicles, and $\mathcal{N} = \{0, 1, \cdots, N\}$ represents the set of nodes consisting of a depot and N customers. Each vehicle i is characterized by a capacity ρ_i and a travel speed χ_i. Each customer node j is associated with coordinates (x_j, y_j) and a demand q_j, where the depot has zero demand, i.e., $q_0 = 0$. Each customer must be visited exactly once, and the total demand served by a vehicle in a single trip cannot exceed its capacity. Vehicles are allowed to return to the depot for replenishment, and each route starts and ends at the depot.

Let $\mathcal{R} = \{R_i\}_{i=1}^{M}$ denote a feasible solution, where R_i represents the route assigned to vehicle i. The objective is to minimize the maximum travel time among all vehicles, known as the min–max objective:

$$\mathcal{R}^* = \arg\min_{\mathcal{R}} \max_{R_i \in \mathcal{R}} \left(\sum_{k=2}^{|R_i|} \frac{dist(R_i^{k-1}, R_i^k)}{\chi_i} \right), \tag{15}$$

where $dist(\cdot, \cdot)$ denotes the distance between two nodes, $|R_i|$ is the length of route R_i, and R_i^k represents the k-th visited node.

B.2 OMDCPDP

The Open Multi-Depot Capacitated Pickup and Delivery Problem (OMDCPDP) considers a fleet of heterogeneous vehicles dispatched from multiple depots to fulfill paired pickup and delivery requests. Let an instance be defined as a tuple $(\mathcal{V}, \mathcal{N}, \mathcal{P}, \mathcal{D})$, where $\mathcal{V} = \{1, 2, \cdots, M\}$ denotes a set of M vehicles, $\mathcal{P} = \{1, \cdots, N\}$ denotes the set of pickup nodes, and $\mathcal{D} = \{N + 1, \cdots, 2N\}$ denotes the set of corresponding delivery nodes. The complete node set is $\mathcal{N} = \mathcal{P} \cup \mathcal{D}$. Each vehicle i is associated with a depot location o_i and a carrying capacity (stacking limit) Q_i. Each pickup node $j \in \mathcal{P}$ is paired with a unique delivery node $j + N \in \mathcal{D}$. Vehicles start from their designated depots and operate in an open-routing manner, meaning they are not required to return to the depot after completing service.

A feasible solution must satisfy the following conditions: each pickup and delivery node is visited exactly once; each pickup must be served before its corresponding delivery; and the number of

active orders carried by a vehicle at any time cannot exceed its capacity. Let $\mathcal{R} = \{R_i\}_{i=1}^{M}$ denote a feasible routing solution, where R_i represents the ordered sequence of nodes visited by vehicle i. The objective of OMDCPDP is to minimize the cumulative arrival times at delivery nodes, which corresponds to minimizing the total delivery lateness:

$$\mathcal{R}^* = \arg\min_{\mathcal{R}} \sum_{j=N+1}^{2N} time(\mathcal{R}, j), \tag{16}$$

where $time(\mathcal{R}, j)$ denotes the arrival time at delivery node j under routing solution $\mathcal{R}$.

C Experimental Details

C.1 HCVRP

C.1.1 Baselines

Our HCVRP experiments are conducted following the setup of Liu et al. (Liu et al., 2024), with baseline-specific hyperparameters configured as reported in their original publications. Below we summarize the competing methods.

SISR Slack Induction by String Removals (SISR) (Christiaens & Vanden Berghe, 2020) is a heuristic solver for VRPs that alternates between decomposing existing routes and rebuilding them from scratch, while also attempting to reduce fleet size. It applies to a broad range of VRP variants, including those with pickup-and-deliver constraints. In our study, SISR is run with the following configuration: $\bar{c} = 10$, $L_{\max} = 10$, $\alpha = 10^{-3}$, $\beta = 10^{-2}$, $T_0 = 100$, $T_f = 1$, and iteration budget iter $= 3 \times 10^5 \times N$.

GA The Genetic Algorithm baseline (Karakatič & Podgorelec, 2015) mimics natural-selection procedures to produce feasible VRP solutions within moderate computational budgets. We adopt the tuned hyperparameters from Liu et al. (Liu et al., 2024): population size $n = 200$, iter $= 40 \times N$, mutation probability $P_m = 0.8$, and crossover probability $P_c = 1$.

SA Simulated Annealing (İlhan, 2021) tackles the capacitated VRP via a population-based scheme enhanced with crossover operators, local search, and an improved 2-opt local refinement routine. We use the hyperparameters from Liu et al. (Liu et al., 2024): initial temperature $T_0 = 100$, final temperature $T_f = 10^{-7}$, Markov chain length $L = 20 \times N$, and cooling rate $\alpha = 0.98$.

AM The Attention Model (Kool et al., 2018) employs multi-head attention layers to solve routing problems and is trained with REINFORCE augmented by deterministic rollouts. In our implementation, the model is adapted under the DRL$_{L_i}$ framework: vehicles are chosen sequentially, after which the next customer node is selected for the active vehicle. Vehicle-specific features are injected into the context embedding to distinguish heterogeneous fleet members.

ET Equity-Transformer (Son et al., 2023) targets min-max routing through a sequential planning paradigm based on the Transformer architecture, with an explicit focus on balancing workload across agents. To accommodate HCVRP, we alter the decoder mask of ET so that only feasible actions are emitted, and incorporate vehicle characteristics into both the input and context encoders, following the modifications described by Liu et al. (Liu et al., 2024).

DPN Decoupling Partition and Navigation (DPN) (Zheng et al., 2024) is a state-of-the-art sequential AR baseline for min-max VRPs. It disentangles customer-to-vehicle assignment from per-vehicle route construction through a dedicated Partition-and-Navigation encoder. Additionally, DPN exploits routing symmetries via an Agent-Permutation-Symmetric loss and employs Rotation-Based Positional Encoding to improve generalization across depot positions. We configure DPN with the same adaptations used for ET.

DRL$_{L_i}$ Li et al.(Li et al., 2022) propose a transformer-based policy for HCVRP that selects a vehicle and a customer in two separate decoding stages, thereby consuming two actions per construction step. We run their original architecture with the added context of heterogeneous vehicle speeds. Note that, as in the original work, each model checkpoint is trained on a single problem distribution with fixed fleet size M and identical vehicle attributes.

2D-Ptr The 2D Array Pointer network (Liu et al., 2024) addresses HCVRP with a dual-encoder design that separately embeds vehicles and customer nodes. Its decoder uses a 2D pointer array to prioritize actions over agents, enabling dynamic decision-making. The architecture is purposely built to accommodate varying numbers of vehicles and customers, yielding robust performance across diverse scenarios.

PARCO PARCO (Berto et al., 2025) is a state-of-the-art Parallel AutoRegressive Combinatorial Optimization framework for multi-agent problems. It constructs solutions for multiple agents simultaneously to reduce inference latency and improve coordination efficiency. The method incorporates transformer-based communication layers and a multiple-pointer decoding mechanism to enable collaborative decision-making while maintaining solution feasibility through conflict handling. Due to its strong performance and generalization ability on large-scale routing and scheduling tasks, PARCO is adopted as a representative autoregressive baseline in our experiments. In our experiments, we use the publicly available code and pretrained checkpoint released by PARCO for evaluation.

C.1.2 DATASETS

Train instance generation Most neural baselines are typically trained and evaluated on instances with the same number of nodes and vehicles, except for PARCO. For DiffPtr, we follow the training strategy of PARCO by adopting a consistent mixed-distribution training scheme and ensuring the same instance data distribution. Specifically, at every training step we sample $N \sim \mathcal{U}(60, 100)$ and $i \sim \mathcal{U}(3, 7)$. Customer coordinates (x_j, y_j) for $j = 1, \ldots, N$, as well as the depot location, are drawn i.i.d. from $\mathcal{U}(0.0, 1.0)$ in the unit square. Customer demands q_j are sampled from $\mathcal{U}(1, 10)$, with $q_0 = 0$ for the depot. Each vehicle $i = 1, \ldots, M$ receives a capacity $\rho_i \sim \mathcal{U}(20, 41)$ and a speed $\chi_i \sim \mathcal{U}(0.5, 1.0)$.

Train trajectory generation DiffPtr is trained via imitation learning and therefore requires expert trajectory datasets. However, obtaining optimal solutions for HCVRP is highly challenging, especially under the min-max objective, making it impractical to use either exact solvers or even the strongest SISR heuristic solver for large-scale trajectory generation. Instead, we employ the state-of-the-art neural solver PARCO to generate training trajectories online. Specifically, for each instance in a batch, PARCO randomly samples 8 trajectories for training without any additional selection or filtering. Therefore, the resulting dataset $\mathcal{D}$ does not exceed the average solution quality of PARCO itself. Nevertheless, as shown in Tab.1, DiffPtr is still able to learn effective solving patterns from these non-optimal expert trajectories and achieves superior performance over both heuristic and neural solver baselines.

Testing Datasets For in-distribution setting, we uses the 1280 instances per problem scale (N, M) provided by Liu et al. (Liu et al., 2024). In Tab.1, we evaluate the generation performance by returning best one of sampled 1280 solutions for each HCVRP instance. As for the out-of-distribution setting, the large-scale HCVRP datasets, whose problem sizes are up to 10× larger than the training instances, are constructed following the same procedure as Liu et al.(Liu et al., 2024). Each scale contains 128 instances. Models trained with mixed-distribution training, including PARCO and DiffPtr, are directly evaluated on these datasets in a zero-shot generalization setting after in-distribution training. For the remaining models without mixed-distribution training, we evaluate the checkpoints trained on the closest in-distribution scale (N=100,M=7). The sampling size for evaluation is set to 64.

C.1.3 DIFFPTR HYPERPARAMETERS

Network The encoder follows a Transformer architecture consisting of L_e encoder layers. Each layer contains a multi-head self-attention module for information interaction and a two-layer feed-forward network, together with residual connections and normalization. In our implementation, L_e is set to 3, the number of attention heads is 8, the feature dimension is 128, and the hidden dimension

of the feed-forward network is 512. RMSNorm is adopted for normalization. Similarly, the Diffusion Decoder also uses $L = 3$ layers with a feature dimension of 128 and a feed-forward hidden dimension of 512. Both the Noisy Action Extractor (NAE) and the Communication layers employ 8 attention heads. LayerNorm is used for normalization in the decoder. In addition, KV caching is applied in the NAE to further improve training and inference efficiency.

Diffusion In training, DiffPtr employs a cosine noise schedule in (Nichol & Dhariwal, 2021), with T=1000. We provide two accelerated inference strategies, namely DDIM(Song et al., 2021) and Consistency Models(Song et al., 2023), both of which adopt a cosine-based inference scheduler to define $[\tau_1, \cdots, \tau_K]$, i.e., $\tau_i = \left\lfloor \cos\left(\frac{(1-ci)\pi}{2}\right) \cdot T \right\rfloor$ from some c. In Tab.1, for both in-distribution and out-of-distribution settings, the number of inference steps K is consistently set to 3.

Training In contrast to most baseline methods, DiffPtr adopts the mixed-training strategy in PARCO, enabling a single model to generalize across varying numbers of nodes and agents, including both in-distribution and out-of-distribution HCVRP settings. DiffPtr is trained via imitation learning for 100 epochs using the Adam optimizer with a learning rate initialized to 10^{-4} and decayed by a factor of 0.1 at the 80th and 95th epochs. For each epoch, 10^5 instances are generated, the batch size is set to 64, and each instance contains 8 expert trajectories. Training is conducted on a single GPU and requires approximately 21 hours.

C.2 OMDCPDP

C.2.1 BASELINES

The OMDCPDP setting considered here is more general and realistic than the one in Zong et al. (Zong et al., 2022), mainly because of the inclusion of multiple depots and the global lateness objective, which is considerably harder to optimize than a plain min-sum criterion.

OR-Tools Google OR-Tools is an open-source toolkit that supplies solvers for linear programming, mixed-integer programming, constraint programming, and routing/scheduling tasks. For the OMDCPDP, OR-Tools can incorporate side constraints to improve solution fidelity. We keep parameters uniform across all problem scales: the global-span cost coefficient is set to 10,000, the initial solution heuristic is `PATH_CHEAPEST_ARC`, and local improvement is performed via `GUIDED_LOCAL_SEARCH`. Time limits are $\{30, 60, 300, 600\}$ seconds for $N = \{50, 100, 500, 1000\}$, respectively.

HAM The Heterogeneous Attention Model (Li et al., 2021) is a neural policy that distinguishes node types through a heterogeneous attention mechanism and enforces precedence constraints between paired pickup and delivery nodes. This design allows the underlying RL agent to make context-aware routing decisions. We extend the original HAM to the multi-agent OMDCPDP by adopting the sequential planning protocol of Son et al. (Son et al., 2024).

MAPDP The Multi-Agent Reinforcement Learning framework for cooperative Pickup and Delivery Problems (Zong et al., 2022) trains a centralized (MA)RL policy that produces coordinated decisions among vehicle agents. A paired context embedding captures inter-dependencies between heterogeneous nodes. To ensure a fair comparison, we equip MAPDP with the same encoder as PARCO while retaining its original random conflict handler and all other hyperparameters reported in the source paper.

PARCO PARCO (Berto et al., 2025) is a state-of-the-art Parallel AutoRegressive Combinatorial Optimization framework for multi-agent problems. It constructs solutions for multiple agents simultaneously to reduce inference latency and improve coordination efficiency. The method incorporates transformer-based communication layers and a multiple-pointer decoding mechanism to enable collaborative decision-making while maintaining solution feasibility through conflict handling. Due to its strong performance and generalization ability on large-scale routing and scheduling tasks, PARCO is adopted as a representative autoregressive baseline in our experiments. In our experiments, we use the publicly available code and pretrained checkpoint released by PARCO for evaluation.

C.2.2 DATASETS

Train data generation For OMDCPDP, each step samples $N \sim \mathcal{U}(50, 100)$ and $i \sim \mathcal{U}(5, 20)$, following the same setting as PARCO (Berto et al., 2025). Tab.2 shows that one DiffPtr model generalizes well enough to outperform other baselines. Node coordinates (x_j, y_j) for $j = 1, \ldots, N$ and vehicle starting locations are both sampled from $\mathcal{U}(0.0, 1.0)$. Every customer has unit demand $q_j = 1$, and every vehicle possesses a stacking limit (capacity) of 3. This configuration mirrors practical parcel-delivery scenarios in which each customer contributes a single package.

Train trajectory generation For OMDCPDP, we also employ the state-of-the-art neural solver PARCO to generate training trajectories online. Specifically, for each instance in a batch, PARCO randomly samples 8 trajectories for training without any additional selection or filtering. Therefore, the resulting dataset $\mathcal{D}$ does not exceed the average solution quality of PARCO itself. Nevertheless, as shown in Tab.2, DiffPtr is still able to learn effective solving patterns from these non-optimal expert trajectories and achieves superior performance over both heuristic and neural solver baselines.

Testing For in-distribution setting, we uses the 1000 instances per problem scale (N, M) provided by PARCO (Berto et al., 2025). In Tab.2, we evaluate the generation performance by returning best one of sampled 1280 solutions for each OMDCPDP instance. As for the out-of-distribution setting, the large-scale OMDCPDP datasets, whose problem sizes are up to 10× larger than the training instances, are also provide by PARCO (Berto et al., 2025). Each scale contains 100 instances.

C.2.3 DIFFPTR HYPERPARAMETERS

The network architecture, diffusion process, and training hyperparameters on OMDCPDP are largely consistent with those used for HCVRP. During training, the batch size is set to 128. For inference, the number of diffusion steps K is set to 2 under the in-distribution setting and 1 under the out-of-distribution setting. The model is trained on a single GPU, with a total training time of approximately 6 hours.

C.3 HARDWARE AND SOFTWARE

Hardware. We experiment on a workstation equipped with Intel Core i9-14900K CPUs and one NVIDIA RTX A6000 GPU with 48 GB of VRAM.

Software. We used Python 3.12, PyTorch 2.8 coupled with PyTorch Lightning, and the operating system is Ubuntu 24.04 LTS.

D ADDITIONAL EXPERIMENTAL RESULTS

We provide additional experiments to further investigate the effectiveness, efficiency, and robustness of DiffPtr. Specifically, we study its performance under the same inference-time budget, the impact of the quantity and quality of training trajectories, and the training cost compared with PARCO.

D.1 PERFORMANCE UNDER THE SAME INFERENCE-TIME BUDGET

To provide a fair comparison of solution generation capability, we further evaluate DiffPtr and PARCO under a comparable inference-time budget. For PARCO, we increase the number of parallel sampled solutions by $10\times$–$20\times$ relative to its default setting, while DiffPtr uses the same sampling configuration as in the main experiments.

The results are reported in the Tab.4, where "s." denotes the number of sampled solutions. DiffPtr achieves comparable or slightly better objective values than PARCO while requiring substantially less inference time. For example, on $N = 60, M = 7$, DiffPtr obtains an objective value of 3.02 in 0.48 seconds, compared with 3.04 in 0.78 seconds for PARCO.

For the out-of-distribution instances with $N = 1000$ and substantially larger numbers of vehicles, DiffPtr also achieves lower objective values and smaller gaps while using less inference time. The performance difference becomes more pronounced as the number of vehicles increases, indicating that the iterative denoising process remains effective when the problem scale changes substantially.

Table 4: Performance under a comparable inference-time budget.

In-distribution	$N = 60, M = 3$			$N = 60, M = 5$			$N = 60, M = 7$		
Metric	Obj.	Gap	Time	Obj.	Gap	Time	Obj.	Gap	Time
PARCO ($s.1280 \times 10$)	6.78	3.24%	0.57s	4.15	3.71%	0.68s	3.04	4.30%	0.78s
DiffPtr ($s.1280$)	**6.77**	**3.08%**	**0.24s**	**4.13**	**3.35%**	**0.35s**	**3.02**	**3.82%**	**0.48s**
Out-of-distribution	$N = 1000, M = 30$			$N = 1000, M = 50$			$N = 1000, M = 70$		
Metric	Obj.	Gap	Time	Obj.	Gap	Time	Obj.	Gap	Time
PARCO ($s.64 \times 20$)	10.28	12.35%	8.60s	6.66	18.60%	10.97s	5.29	27.23%	12.89s
DiffPtr ($s.64$)	**9.15**	**0.00%**	**5.40s**	**5.61**	**0.00%**	**8.14s**	**4.16**	**0.00%**	**10.41s**

D.2 Impact of Training Trajectory Quantity and Quality

We further investigate how the quantity and quality of expert trajectories affect DiffPtr. We consider three additional training settings. First, DiffPtr-half is trained using only half of the original training trajectories, where the trajectories are generated by the original PARCO model without additional screening. Second, DiffPtr(Greedy PARCO) uses trajectories generated by greedy PARCO. Third, DiffPtr(2D-Ptr) uses trajectories generated by the weaker 2D-Ptr teacher. The results are summarized in Tab.5.

Table 5: Impact of training trajectory quantity and quality on HCVRP.

	In-distribution						***Out-of-distribution***					
	$N = 100, M = 3$		$N = 100, M = 5$		$N = 100, M = 7$		$N = 1000, M = 30$		$N = 1000, M = 50$		$N = 1000, M = 70$	
Metric	Obj.	Gap	Obj.	Gap	Obj.	Gap	Obj.	Gap	Obj.	Gap	Obj.	Gap
2D-Ptr	10.71	4.08%	6.46	4.70%	4.68	5.17%	10.40	13.67%	5.96	6.19%	4.37	4.97%
DiffPtr(2D-Ptr)	10.72	4.15%	6.46	4.71%	4.69	5.35%	9.85	7.62%	5.71	1.64%	4.25	2.28%
PARCO	10.61	3.11%	6.36	3.08%	4.58	2.92%	10.49	14.59%	6.84	21.82%	5.51	32.37%
DiffPtr(Greedy PARCO)	10.58	2.84%	6.34	2.69%	4.57	2.62%	9.01	-1.53%	5.52	-1.73%	4.07	-2.04%
DiffPtr-half	10.59	2.92%	6.34	2.79%	4.57	2.78%	9.11	-0.38%	5.58	-0.65%	4.13	-0.74%
DiffPtr	10.59	2.88%	6.34	2.74%	4.57	2.66%	9.15	0.00%	5.61	0.00%	4.16	0.00%

Reducing the number of training trajectories has only a minor effect on in-distribution performance. Interestingly, DiffPtr-half slightly improves the out-of-distribution (OOD) results, suggesting that using fewer demonstrations does not necessarily impair generalization and may reduce the extent to which the model fits scale-specific characteristics of the training distribution.

The quality of the training trajectories also affects OOD performance. Training with greedy PARCO trajectories leads to improved OOD results, while DiffPtr trained with trajectories from the weaker 2D-Ptr teacher still achieves better OOD performance than 2D-Ptr itself. These results suggest that DiffPtr is not restricted to a particular teacher quality and can benefit from both higher-quality demonstrations and relatively weak supervision.

D.3 Ablation study of Diffusion Decoder on HCVRP

To further examine the contribution of the diffusion decoder components, we conduct additional ablations on HCVRP. Table 6 compares different DiffPtr variants that remove individual decoder components, including the communication module (Comm.), Noisy Action Extractor (NAE), and feed-forward network (FNN).

The results show that communication and NAE substantially degrade performance, while the complete DiffPtr model achieves the best result. The results also indicate that the contribution of individual components is not uniform across scales, motivating the use of the complete architecture. These findings further highlight the importance of information exchange among multiple agents, while demonstrating that NAE can more effectively recover useful information from noisy actions.

Table 6: Ablation results about Diffusion Decoder on HCVRP.

	$N = 100, M = 7$		$N = 1000, M = 70$	
Metric	Obj.	Gap	Obj.	Gap
w/o Comm.	4.79	7.65%	5.48	31.72%
w/o NAE	4.70	5.62%	4.44	6.77%
w/o FNN	4.57	2.76%	4.20	0.86%
DiffPtr	4.57	2.66%	4.16	0.00%

Table 7: Training cost of PARCO and DiffPtr on an NVIDIA A6000 GPU.

	HCVRP		OMDCPDP	
	Training Time	GPU Memory	Training Time	GPU Memory
PARCO	40.8h	24.37GB	4.2h	17.78GB
DiffPtr	21.2h	12.59GB	5.4h	9.72GB

D.4 Training Cost

We compare the training efficiency of DiffPtr and PARCO on the same NVIDIA A6000 GPU. Table 7 reports the training time and peak GPU memory consumption on HCVRP and OMDCPDP.

DiffPtr requires substantially less GPU memory than PARCO on both tasks. Its training time is also lower on HCVRP, while remaining comparable on OMDCPDP. Overall, these results show that the diffusion-based decoder can be trained within a practical computational budget without introducing a substantial memory overhead.