SeqDiCO: Sequence-oriented Diffusion for Scale-Generalizable Neural Combinatorial Optimization

Yu Wang[1,2], Qiaolin Lu[2], Yang Wu[1], Bohao Qu[3], Shirui Pan[4], Chengqi Zhang[2], Yi Chang[1,5,6]

[1]Jilin University [2]Hong Kong Polytechnic University [3]A*STAR [4]Griffith University
[5]International Center of Future Science, Jilin University
[6]Engineering Research Center of Knowledge-Driven Human-Machine Intelligence, MOE, China

Abstract

Diffusion-based neural combinatorial optimization (NCO) has recently shown promise for routing problems by generating solutions through denoising or reconstruction. Most existing diffusion solvers, however, define the generative process over full-instance solution structures. This instance-level formulation ties the learned reconstruction distribution to the problem size observed during training, making it difficult to transfer models trained on small instances to larger ones. In this paper, we propose SeqDiCO, a sequence-oriented diffusion framework for scale-generalizable neural combinatorial optimization. The key idea is to move diffusion from full-instance solutions to fixed-endpoint sequence subproblems, decoupling the learned reconstruction representation from the full instance size. Given two endpoints and a set of intermediate nodes, SeqDiCO learns a local denoising operator that reconstructs the open path connecting the endpoints through progressive denoising of corrupted structural observations. This local formulation allows the same operator trained on small instances to be repeatedly composed to solve larger routing problems. During inference, SeqDiCO uses this operator for both construction, by growing a solution through receding-horizon local reconstruction, and refinement, by replacing selected subpaths in an existing solution. Experiments on TSP and CVRP show that SeqDiCO consistently outperforms representative diffusion-based and autoregressive neural solvers under cross-scale evaluation. Trained only on 100-node instances, SeqDiCO generalizes to 10K-node problems, reducing the optimality gap by up to 51% over SOTA neural solvers, while achieving up to $10\times$ faster inference on representative benchmarks.

1 Introduction

Combinatorial optimization problems (COPs) [1, 2] are fundamental to routing [3, 4], logistics [5, 6], scheduling, and resource allocation [7]. Many of these problems are NP-hard, making exact algorithms computationally prohibitive at large scales [8, 9]. Neural combinatorial optimization (NCO) has therefore emerged as a learning-based alternative that amortizes search by training neural solvers from data [2, 10, 11]. For routing problems such as the Traveling Salesman Problem (TSP) and the Capacitated Vehicle Routing Problem (CVRP), existing neural solvers broadly follow autoregressive or non-autoregressive generation paradigms [11]. Autoregressive methods construct solutions sequentially by selecting one node or edge at a time [12, 13, 14, 15], whereas non-autoregressive methods predict multiple solution components in parallel, typically through edge probabilities, heatmaps, or

* Correspondence to: Yi Chang (yichang@jlu.edu.cn).

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

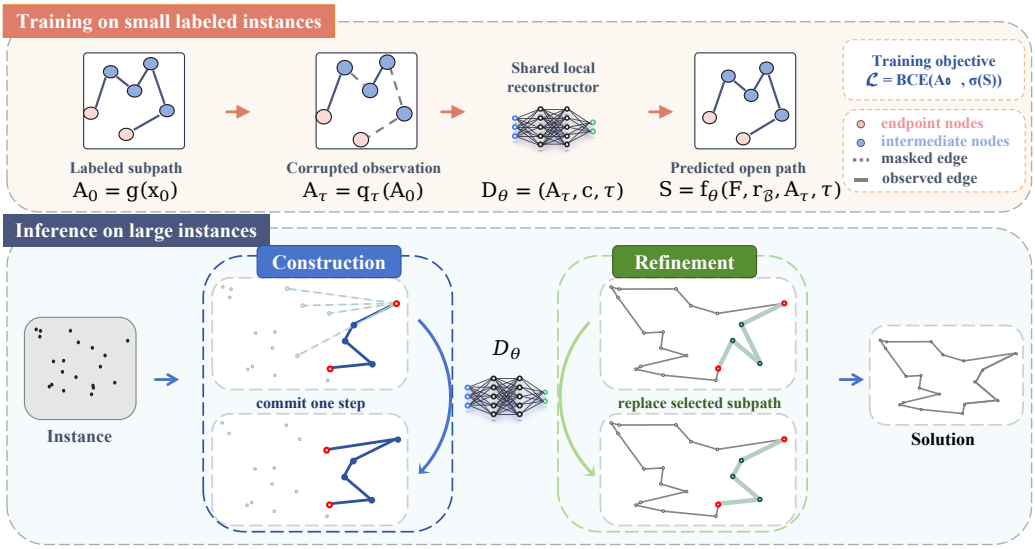


Figure 1: Overview of SeqDiCO. A shared sequence-oriented diffusion operator is trained on fixed-endpoint subpath reconstruction tasks from small instances. During inference, the same reconstructor is repeatedly composed for construction and refinement: construction grows a solution through receding-horizon local reconstruction, while refinement replaces selected subpaths if the reconstructed alternative improves the objective.

complete solution structures [16, 17]. While autoregressive construction naturally accommodates variable-size instances, its sequential nature limits parallelism on large problems; non-autoregressive prediction offers greater parallelism but often operates on full-instance representations that can be difficult to transfer across scales. More recently, generative approaches have extended non-autoregressive NCO beyond one-shot prediction by learning distributions over high-quality solutions, enabling iterative sampling and refinement while retaining parallel generation [18, 19, 20].

Diffusion-based NCO has recently emerged as a promising generative paradigm for learning distributions over high-quality solutions [21, 22]. By formulating solution generation as conditional denoising or reconstruction, recent diffusion solvers have explored increasingly effective corruption schemes, denoising processes, and inference strategies [18, 20, 23, 24, 25, 26]. Despite these advances, most existing methods still perform diffusion over full-instance solution representations, such as adjacency matrices or heatmaps [18, 19, 23, 24]. This modeling granularity introduces a fundamental challenge for cross-scale transfer: as the instance size grows, both the dimensionality of the modeled solution structure and the dependencies among its decision variables change. Consequently, the reconstruction problem itself remains coupled to the problem scale observed during training, making it difficult to directly transfer a model trained on small instances to substantially larger ones [15, 27]. This limitation is particularly relevant in practical routing settings, where problem sizes vary widely while obtaining high-quality training solutions for large instances is increasingly expensive.

In this paper, we reconsider whether diffusion-based NCO must be defined over full-instance solutions. We propose **SeqDiCO**, a sequence-oriented diffusion framework that shifts the modeling granularity from entire solutions to reusable local sequence subproblems. Instead of learning a reconstruction distribution over complete tours of a fixed size, SeqDiCO learns a local diffusion operator for fixed-endpoint subpath reconstruction. Given two endpoint nodes and a set of intermediate nodes, the model progressively reconstructs the open path connecting the endpoints from a corrupted sequence-induced adjacency structure. By defining diffusion over local sequence structures whose size is decoupled from the full instance size, SeqDiCO learns reconstruction rules that can be repeatedly applied across problem scales. During inference, the same reconstructor supports two complementary solving modes. In the construction mode, SeqDiCO grows a solution through receding-horizon local reconstruction; in the refinement mode, it reconstructs alternative subpaths with fixed endpoints to improve an existing solution. This design places SeqDiCO at an intermediate generation granularity between autoregressive construction and full-instance diffusion. Unlike autoregressive solvers that generate the global solution through sequential node-level decisions, SeqDiCO reconstructs multiple structural decisions in parallel within each local sequence subproblem. Unlike full-instance diffusion

solvers that jointly reconstruct a solution representation whose size grows with the problem, SeqDiCO composes the same local reconstruction operator across the instance, retaining diffusion-style parallel reconstruction while decoupling the learned representation from the global problem size.

Our contributions are summarized as follows: 1) We propose SeqDiCO, a sequence-oriented diffusion framework that shifts diffusion from full-instance solutions to fixed-endpoint local sequence subproblems for improved scale generalization. 2) We develop a diffusion-based local reconstructor that progressively recovers open paths from corrupted sequence structures and can be reused across problem scales. 3) We introduce a unified inference procedure that uses the same local reconstructor for both receding-horizon solution construction and subpath-based refinement. 4) Extensive experiments on TSP and CVRP demonstrate strong cross-scale generalization from 100-node training instances to problems with up to 10K nodes, outperforming representative neural solvers on both synthetic and real-world benchmarks.

2 Preliminaries and Related Work

Problem Definition. We study graph-based combinatorial optimization problems. Let $G = (V, E)$ be a problem instance with $|V| = n$, where V and E denote the node set and edge set, respectively. A feasible solution is denoted by a discrete variable $\mathbf{x} \in \Omega(G)$, where $\Omega(G)$ is determined by problem-specific constraints. Given an objective function $l(\mathbf{x}; G)$, the goal is

$$\mathbf{x}^* = \arg \min_{\mathbf{x} \in \Omega(G)} l(\mathbf{x}; G). \tag{1}$$

In routing problems, $\mathbf{x}$ usually specifies an ordering or connection pattern over nodes. For TSP, a solution is a Hamiltonian cycle visiting all nodes exactly once. It can be represented as a permutation $\pi = (\pi_1, \ldots, \pi_n)$, with objective

$$l(\pi; G) = d_{\pi_n, \pi_1} + \sum_{i=1}^{n-1} d_{\pi_i, \pi_{i+1}}, \tag{2}$$

where $d_{i,j}$ denotes the distance between nodes v_i and v_j. For CVRP, the graph contains a depot, customer demands, and vehicle capacity constraints. A solution consists of multiple depot-starting and depot-ending routes that visit each customer exactly once while satisfying capacity limits.

Neural Routing Solvers. Neural combinatorial optimization learns data-driven solvers for routing problems such as TSP and CVRP [2, 10, 11]. Existing methods can be roughly divided into construction-based and improvement-based solvers. Construction methods generate solutions from scratch, either autoregressively by selecting nodes or edges step by step [12, 13, 14, 15], or non-autoregressively by predicting edge probabilities, heatmaps, or solution structures in parallel [16, 17]. Improvement methods refine existing solutions through learned local modifications, including local rewriting, 2-opt-style moves, destroy-and-repair, and reconstruction-based procedures [28, 29, 30, 31, 32, 33]. SeqDiCO is related to both paradigms: it learns a local reconstruction operator that can be used to construct solutions from scratch and to refine existing subpaths.

Diffusion-based CO Solvers. Diffusion-based neural solvers cast CO as conditional denoising [18, 21, 22]. Given an instance G, they learn to recover a high-quality solution $\mathbf{x}_0$ from a corrupted state $\mathbf{x}_\tau$, where τ denotes the diffusion timestep. This process is typically parameterized as

$$p_\theta(\mathbf{x}_0 \mid \mathbf{x}_\tau, G, \tau). \tag{3}$$

In routing problems, the solution is often represented as a binary adjacency matrix $\mathbf{A}_0^G \in \{0, 1\}^{n \times n}$, where $\mathbf{A}_{0,i,j}^G = 1$ indicates that edge (i, j) is selected in the solution. Existing diffusion-based solvers commonly model

$$p_\theta(\mathbf{A}_0^G \mid \mathbf{A}_\tau^G, G, \tau), \tag{4}$$

where $\mathbf{A}_\tau^G$ is a corrupted instance-level solution structure. The corruption process is commonly instantiated either by randomly flipping edge variables over the whole adjacency matrix [18, 19] or by masking selected edges in the ground-truth solution and reconstructing the missing structure [20, 26, 34]. Recent methods further improve this paradigm through structured masking, training–inference consistency, native solution expansion, and test-time refinement [19, 20, 25, 26, 34]. Despite different corruption and inference mechanisms, these methods primarily define reconstruction over full-instance solution representations. Consequently, the dimensionality of the modeled structure grows with the instance size, coupling the learned reconstruction problem to the problem scale [15, 27].

Scale Generalization and Local Reconstruction. Scale generalization is a central challenge for neural routing solvers: models trained on small instances are expected to solve much larger ones, although the solution space, representation size, and dependency structure all change with problem size [15, 27]. Existing approaches improve transfer through stronger architectures, invariant or local views, distance-aware representations, decomposition frameworks, or local construction policies [35, 36, 37, 38, 39, 40, 41, 42]. These studies demonstrate the effectiveness of local and decomposed reasoning for large-scale neural routing, using autoregressive local policies, instance decomposition, or learned reconstruction and repair mechanisms.

From Instance-level to Sequence-level Reconstruction. SeqDiCO retains the diffusion-based reconstruction paradigm while changing its modeling granularity. Rather than reconstructing a full-instance solution structure $\mathbf{A}_0^G \in \{0,1\}^{n\times n}$, SeqDiCO performs diffusion over a local sequence-induced adjacency $\mathbf{A}_0 \in \{0,1\}^{L\times L}$ associated with a fixed-endpoint subpath, where the local reconstruction size L is decoupled from the full instance size n. This sequence-level formulation enables a shared local reconstructor to be learned from small instances and repeatedly composed across problem scales. The next section formalizes the resulting sequence-oriented diffusion framework. A more detailed discussion of related work is provided in Appendix B.

3 SeqDiCO: Sequence-oriented Diffusion-based Combinatorial Optimization

3.1 Overview

SeqDiCO is built on a simple principle: instead of learning to denoise a full-instance solution structure as in existing diffusion-based CO solvers [18, 20, 23, 24], we learn to denoise a local sequence subproblem whose size is decoupled from the full instance size, and repeatedly compose this local operator during inference. This changes the modeled object from an instance-level adjacency matrix of size $n \times n$ to a local sequence-induced adjacency matrix of size $L \times L$, where $L \ll n$ and does not scale with the full problem size.

The framework has three components. First, we define a sequence-oriented mask diffusion process over local ordered structures. Given a clean local sequence, we represent it as an adjacency matrix, partially mask its selected edges, and train a model to reconstruct the clean structure in parallel. Second, we instantiate the local task as fixed-endpoint subpath reconstruction. Given two endpoints and a set of intermediate nodes, the model learns to recover the open path connecting the endpoints. Third, we use the same learned reconstructor for both construction and refinement. During construction, the model repeatedly solves local endpoint-conditioned tasks and commits one step at a time. During refinement, it reconstructs subpaths of an existing solution and accepts improving replacements.

This unified interface is the key to scale generalization. Training exposes the model only to local sequence reconstruction tasks sampled from small labeled instances, while inference solves large instances by composing many such fixed-size reconstructions, echoing the general idea of local or decomposed reasoning for scalable routing [37, 38, 40, 41].

3.2 Sequence-oriented Mask Diffusion

Following the denoising view of discrete diffusion models and diffusion-based CO solvers [18, 22], we define diffusion over a local sequence-induced structure. Let $x_0 = (x_0^1, x_0^2, \ldots, x_0^L)$ denote a clean local solution sequence of length L. For routing problems, x_0 corresponds to an ordered path segment. We map this sequence to an adjacency representation

$$\mathbf{A}_0 = g(x_0) \in \{0,1\}^{L\times L}, \tag{5}$$

where $g(\cdot)$ connects consecutive elements in the local sequence. The local task is conditioned on a context $\mathbf{c}$, which contains the node features, endpoint-role information, and other problem-specific information required for reconstruction.

Forward masking. Given the clean local structure $\mathbf{A}_0$, we construct a corrupted observation $\mathbf{A}_\tau$ through a timestep-dependent masking process:

$$\mathbf{A}_\tau = q_\tau(\mathbf{A}_0), \quad \tau \in (0,1). \tag{6}$$

In our setting, similar to mask-based structured reconstruction [20], q_τ masks a subset of selected edges in $\mathbf{A}_0$, i.e., some entries with value one are changed to zero. The masking ratio is controlled

by τ, so a larger τ corresponds to fewer visible solution edges and a more difficult reconstruction problem. The model must infer the missing local sequence structure from the remaining edges and the context $\mathbf{c}$.

Reverse reconstruction. The reverse process is parameterized by a sequence-oriented denoiser

$$\mathcal{D}_\theta(\mathbf{A}_\tau, \mathbf{c}, \tau) \triangleq p_\theta(\mathbf{A}_0 \mid \mathbf{A}_\tau, \mathbf{c}, \tau). \tag{7}$$

Given a corrupted adjacency, context, and diffusion timestep, $\mathcal{D}_\theta$ predicts edge probabilities for the clean sequence-induced structure. At inference, reconstruction is performed progressively over a decreasing timestep schedule $\tau_K > \tau_{K-1} > \cdots > \tau_1$. Starting from an initially corrupted local structure, the prediction at each step is used to update the structural observation for the next denoising step, progressively revealing the local path. The final prediction is then decoded into an open path.

Model parameterization. We instantiate $\mathcal{D}_\theta$ with a Transformer-based encoder-decoder. Given node features $\mathbf{F}$, endpoint-role indicators $\mathbf{r}_\mathcal{B}$, the corrupted adjacency $\mathbf{A}_\tau$, and timestep τ, the model predicts pairwise edge logits

$$\mathbf{S} = f_\theta(\mathbf{F}, \mathbf{r}_\mathcal{B}, \mathbf{A}_\tau, \tau), \quad \mathbf{S} \in \mathbb{R}^{L \times L}. \tag{8}$$

The encoder produces contextual node representations from node features and endpoint roles. The decoder is conditioned on the timestep and uses $\mathbf{A}_\tau$ as structural guidance, so that reconstruction can adapt to different corruption levels and exploit visible edges. The final logits are computed by pairwise inner products:

$$\mathbf{S}_{ij} = \bar{\mathbf{z}}_i^\top \bar{\mathbf{z}}_j, \tag{9}$$

where $\bar{\mathbf{z}}_i$ and $\bar{\mathbf{z}}_j$ are decoded node representations. These logits parameterize the edge probabilities of the reconstructed local adjacency. A complete architectural description is provided in Appendix C.

3.3 Fixed-endpoint Subpath Reconstruction

We train the local reconstructor through fixed-endpoint subpath reconstruction tasks sampled from high-quality or optimal tours. This fixed-endpoint formulation is related to the broader use of local rewriting, large-neighborhood repair, and insertion-style local construction in neural routing [28, 29, 31, 32, 42], but differs in that the local move is produced by a diffusion-style parallel reconstructor.

Given a reference tour $\pi = (\pi_1, \pi_2, \ldots, \pi_n)$, we sample a contiguous segment of length $L = k + 2$:

$$x_0 = (\pi_i, \pi_{i+1}, \ldots, \pi_{i+k+1}). \tag{10}$$

The two boundary nodes define the endpoint set $\mathcal{B} = \{\pi_i, \pi_{i+k+1}\}$, and the remaining nodes form the intermediate set $\mathcal{M} = \{\pi_{i+1}, \ldots, \pi_{i+k}\}$. The target structure is the open-path adjacency $\mathbf{A}_0 = g(x_0)$, which connects the two endpoints while visiting all intermediate nodes exactly once. For undirected routing problems, the reversed sequence represents the same open path, so the two endpoints are treated symmetrically.

To construct the model input, we randomly permute the intermediate nodes and keep the endpoints fixed. This prevents the model from exploiting the input order and forces it to infer the path structure from geometry, endpoint conditions, and partial edge observations. We then sample a timestep τ and corrupt the clean adjacency:

$$\mathbf{A}_\tau = q_\tau(\mathbf{A}_0). \tag{11}$$

The model predicts pairwise logits

$$\mathbf{S} = f_\theta(\mathbf{F}, \mathbf{r}_\mathcal{B}, \mathbf{A}_\tau, \tau), \tag{12}$$

and is trained to recover $\mathbf{A}_0$ with a binary reconstruction loss:

$$\mathcal{L} = -\frac{1}{|\Omega_{\text{pair}}|} \sum_{(i,j) \in \Omega_{\text{pair}}} \Big[\mathbf{A}_{0,ij} \log \sigma(\mathbf{S}_{ij}) + (1 - \mathbf{A}_{0,ij}) \log(1 - \sigma(\mathbf{S}_{ij})) \Big], \tag{13}$$

where Ω_{pair} denotes the set of node pairs considered in the local task.

This training scheme turns a limited set of labeled tours into a large number of local reconstruction tasks through random subpath sampling, intermediate-node permutation, varying sequence lengths, and different corruption strengths. As a result, the model learns transferable local routing patterns without requiring labels for large-scale instances.

3.4 Unified Construction and Refinement

After training, the same reconstructor $\mathcal{D}_\theta$ is used as a reusable local operator for both construction and refinement. In both modes, inference forms a fixed-endpoint local task and progressively reconstructs its local structure. The resulting edge scores are then used either to select the next construction move or to decode an alternative subpath for refinement.

Construction. In construction, SeqDiCO incrementally builds a solution from scratch. Starting from an initial node, it repeatedly selects a fixed-size candidate set around the current endpoint, forms an endpoint-conditioned local reconstruction task, and applies $\mathcal{D}_\theta$ to predict a local open path. Instead of committing the entire predicted path, the algorithm commits only the first move and then forms the next local task. This receding-horizon strategy keeps each model call close to the training distribution while allowing the complete solution to grow to arbitrary sizes, similar in spirit to local construction strategies for large-scale routing [38, 42]. For TSP, candidate moves are restricted to unvisited nodes. For CVRP, candidate customers must additionally satisfy the remaining vehicle capacity; when no feasible customer remains, the route returns to the depot.

Algorithm 1 SeqDiCO Construction

Require: Instance $G = (V, E)$, reconstructor $\mathcal{D}_\theta$, local length L
Ensure: Initial solution π
$\pi \leftarrow \emptyset, \mathcal{U} \leftarrow V$
Select an initial node and update $\pi, \mathcal{U}$
while $\mathcal{U} \neq \emptyset$ **do**
  Form an endpoint-conditioned local task from the current state
  $\mathbf{S} \leftarrow \textsc{ProgressiveDenoise}(\mathcal{D}_\theta, \mathbf{c})$
  Select the next feasible move from $\mathbf{S}$
  Update π and $\mathcal{U}$
end while
return π

Refinement. In refinement, SeqDiCO improves an existing solution by local replacement. Given a current tour or route set, we select a contiguous segment, fix its two boundary nodes as $\mathcal{B}$, perturb the intermediate nodes, and invoke $\mathcal{D}_\theta$ to reconstruct an alternative open path connecting the same endpoints. The reconstructed segment is inserted back into the solution if it reduces the objective value. This procedure resembles learned local improvement and large-neighborhood search [28, 29, 30, 31, 33], but the move proposal is generated by a learned diffusion reconstructor rather than by a manually designed operator such as 2-opt or k-opt. For CVRP, refinement operates on depot-bounded route segments, and a reconstructed candidate is accepted only if it satisfies the vehicle-capacity constraints and improves the objective.

Algorithm 2 SeqDiCO Refinement

Require: Solution π, reconstructor $\mathcal{D}_\theta$, local length L, steps T
Ensure: Refined solution π
for $t = 1$ to T **do**
  Select a segment from π and fix its endpoints
  Form local task $(\mathbf{A}_\tau, \mathbf{c}, \tau)$
  $\hat{\mathbf{A}}_0 \leftarrow \mathcal{D}_\theta(\mathbf{A}_\tau, \mathbf{c}, \tau)$
  Decode $\hat{\mathbf{A}}_0$ into an alternative subpath
  Replace the segment if the objective improves
end for
return π

Constrained local decoding. Both solving modes use a simple greedy decoder to convert the predicted edge scores into routing decisions. Given $\mathbf{S}$, the decoder starts from a fixed endpoint and repeatedly selects the highest-scoring feasible unvisited node. For refinement, this process continues until all intermediate nodes are visited, after which the other fixed endpoint is appended to form an open path. For construction, a prefix of the reconstructed local path is committed before the next local reconstruction task is formed, with a single committed move used by default.

Overall, training and inference share the same fixed-endpoint reconstruction interface. The model operates on local sequence subproblems whose size is decoupled from the full instance size, while large-scale solutions are obtained by composing many such reconstructions. This allows SeqDiCO to retain diffusion-style parallel reconstruction at the local level while extending the learned reconstructor across problem scales.

4 Experiments

We evaluate SeqDiCO on TSP and CVRP under both synthetic and real-world settings. The experiments are designed to answer three questions: (i) how well SeqDiCO generalizes from small training

Table 1: Results on synthetic TSP instances. Results marked with an asterisk (*) are from the original papers. OOM indicates out-of-memory. Time denotes the total inference time for the full test set. The best neural results are bold.

Method	TSP100			TSP1K			TSP10K		
	Length	Gap	Time	Length	Gap	Time	Length	Gap	Time
LKH3	7.76325	0.000%	56m	23.119	0.000%	8.2h	71.778	0.000%	8.8h
Concorde	7.76324	0.000%	34m	23.118	-0.003%	7.8h	72	0.310%	22.4h
Att-GCN+MCTS*	7.764	0.037%	15m	23.863	3.224%	13m	74.93	4.390%	1.8h
DIFUSCO*	7.78	0.240%	-	23.56	1.900%	12.1m	73.62	2.580%	47m
T2T*	7.76	0.060%	-	23.3	0.780%	54.7m	-	-	-
StruDiCO	7.764	0.020%	2.8m	23.862	3.220%	10.2m	85.802	19.537%	1.1h
NExCO	7.764	0.009%	32.7m	23.694	2.692%	2.2h	-	-	-
MaskCO	7.763	0.000%	9.5m	32.676	40.684%	5.7m	-	-	-
NeurOpt	7.765	0.023%	1.8h	307.578	>100%	3h	-	-	-
SO-mixed*	-	-	-	23.766	2.800%	55.5m	74.299	3.517%	2.034h
H-TSP	-	-	-	24.718	6.912%	34s	77.75	8.320%	25m
GLOP	7.767	0.048%	1.4h	23.84	3.119%	2.4m	75.04	4.545%	25m
UDC-x_{50} ($\alpha = 50$)	7.788	0.319%	1.38h	23.53	1.782%	6.21m	OOM	OOM	OOM
POMO aug×8	7.774	0.138%	0.7m	32.5	40.577%	6.1m	OOM	OOM	OOM
INViT-3V aug×16	7.92	2.019%	3.02h	24.343	5.299%	30.43m	76.494	6.570%	1.19h
ELG	7.781	0.229%	2.2m	25.738	11.328%	1.3m	OOM	OOM	OOM
BQ bs16	7.764	0.010%	20m	23.432	1.354%	21m	OOM	OOM	OOM
LEHD RRC1000	7.763	0.002%	2.3h	23.288	0.731%	5.2h	80.9	12.709%	3.6h
L2C-Insert (I=1000)	7.763	0.000%	8.0h	23.230	0.481%	46.4m	73.270	2.079%	16.64m
SeqDiCO Greedy	7.773	0.131%	0.2m	23.702	2.522%	0.5m	77.196	7.475%	1.86m
SeqDiCO Refined	**7.763**	**0.000%**	24.1m	**23.188**	**0.281%**	4.4m	**73.19**	**1.967%**	4.64m

instances to substantially larger problem scales; (ii) how sequence-level modeling and diffusion-based reconstruction contribute to cross-scale generalization; and (iii) how SeqDiCO compares with classical and neural solvers in solution quality, scalability, and computational efficiency.

Setup. SeqDiCO is trained only on 100-node synthetic instances and evaluated without fine-tuning on 100-, 1K-, and 10K-node instances. For TSP, labels are generated by Concorde; for CVRP, labels are generated by HGS [43]. Local sequence reconstruction tasks are extracted from labeled tours, with varying subpath lengths and corruption levels used during training. We also evaluate on real-world EUC_2D instances from TSPLIB [44] and CVRPLIB [45].

Baselines and metrics. We compare with classical solvers, including Concorde, LKH3, and HGS, as well as representative neural solvers covering autoregressive construction, non-autoregressive prediction, diffusion-based solving, decomposition-based solving, and improvement-based solving. We report average objective value, optimality gap, and total inference time for the full test set. For SeqDiCO, we report both the initial construction and the refined result to evaluate the effect of local refinement. Refinement budgets and inference settings are detailed in Appendix D, with matched budgets used for direct comparisons with improvement-based baselines. The code is available here.

For runtime evaluation, publicly available neural baselines are reproduced on the same GPU when supported by their official implementations, while runtimes quoted from the original papers are marked separately in the tables. Classical solvers use their standard CPU implementations and mainly serve as solution-quality references. Detailed dataset statistics, model hyperparameters, baseline configurations, inference settings, and runtime measurement protocols are provided in Appendix D.

4.1 Main Results

Tables 1 and 2 report the results on synthetic TSP and CVRP benchmarks. Since SeqDiCO is trained only on 100-node instances and directly evaluated on larger instances without fine-tuning, the results on 1K and 10K nodes evaluate its cross-scale generalization.

Synthetic TSP. On TSP100, SeqDiCO reaches near-optimal performance, comparable to the strongest neural solvers. The difference becomes more pronounced as the problem scale increases. On TSP1K, SeqDiCO achieves a gap of 0.281%, substantially outperforming the other neural baselines, including recent diffusion-based solvers. This advantage persists on TSP10K, where SeqDiCO maintains a

Table 2: Results on synthetic CVRP instances. Results marked with an asterisk (*) are from the original papers. OOM indicates out-of-memory. Time denotes the total inference time for the full test set. The best neural results are bold.

Method	CVRP100			CVRP1K			CVRP10K		
	Length	Gap	Time	Length	Gap	Time	Length	Gap	Time
HGS	15.563	0.000%	4.5h	41.161	0.000%	5h	226.177	0.000%	8h
LKH3	15.65	0.540%	12h	42.156	2.416%	6.1h	290.575	28.472%	10.3h
NeurOpt	15.66	0.627%	3h	-	-	-	-	-	-
GLOP-G (LKH3)	-	-	-	45.753	11.154%	1.93m	276.252	22.140%	27.98m
UDC-x$_{50}$ ($\alpha = 50$)	16.291	4.699%	2.69h	43.583	5.884%	11.96m	OOM	OOM	OOM
POMO aug×8	15.75	1.230%	0.86m	100.990	>100%	5.83m	OOM	OOM	OOM
INViT-3V aug×16	16.538	6.282%	3.93h	46.607	13.230%	26.97m	275.691	21.892%	1.28h
ELG	15.84	1.760%	3.4m	46.847	13.812%	3.16m	277.685	22.773%	11.92m
BQ bs16	15.81	1.560%	18m	43.121	4.762%	15m	275.494	21.805%	1.36h
LEHD RRC1000	15.63	0.420%	2.0h	42.408	3.028%	4.15h	248.604	9.916%	8.1h
L2C-Insert (I=1000)	15.627	0.413%	10.6h	43.152	4.836%	2.05h	245.732	8.646%	34.72m
SeqDiCO Greedy	15.99	2.834%	2.26m	44.355	7.76%	0.58m	249.563	10.34%	1.95m
SeqDiCO Refined	**15.589**	**0.166%**	40.2m	**42.344**	**2.87%**	7.55m	**235.805**	**4.25%**	3.7m

Table 3: Results on real-world TSPLIB and CVRPLIB instances. We report average gaps with respect to the best-known solutions. The best neural results are bold.

Method	TSPLIB			CVRPLIB		
	N≤200	200<N≤1K	All	N≤200	200<N≤1K	All
NeurOpt	9.456%	26.345%	16.350%	24.878%	25.301%	25.190%
UDC-x$_{250}$ ($\alpha = 50$)	0.533%	6.578%	3.072%	9.635%	8.864%	9.033%
GLOP	0.561%	2.012%	1.153%	-	-	-
INViT-3V aug×16	2.238%	5.654%	3.673%	8.231%	11.504%	10.784%
ELG	1.180%	5.910%	3.111%	4.510%	7.752%	7.039%
POMO aug×8	2.022%	21.447%	9.951%	6.869%	46.984%	38.159%
BQ bs16	1.289%	1.946%	1.557%	7.661%	8.195%	8.078%
Fast T2T	0.28%	0.93%	0.55%	-	-	-
LEHD RRC1000	0.215%	0.620%	0.380%	2.022%	3.932%	3.512%
L2C-Insert (I=1000)	0.197%	0.468%	0.308%	1.604%	3.651%	3.201%
SeqDiCO Refined	**0.09%**	**0.175%**	**0.125%**	**0.215%**	**2.49%**	**1.99%**

gap of 1.967% while several neural solvers either exhibit substantially larger degradation or cannot scale to this problem size. These results highlight the cross-scale generalization of the learned sequence-level reconstructor beyond the instance size seen during training.

Synthetic CVRP. SeqDiCO also exhibits strong cross-scale generalization on the more constrained CVRP setting. It achieves gaps of 0.166%, 2.870%, and 4.250% on CVRP100, CVRP1K, and CVRP10K, respectively, consistently outperforming the neural baselines across the three scales. The advantage becomes particularly clear on larger instances, where several competing methods suffer substantial performance degradation or fail to scale to 10K nodes. These results show that sequence-level reconstruction remains effective under route and capacity constraints.

Construction and refinement. The construction-only variant evaluates the ability of SeqDiCO to generate feasible initial solutions without subsequent local refinement. It provides fast initial solutions even on large instances, requiring only 0.5 minutes on TSP1K and 1.86 minutes on TSP10K. Applying the same local reconstructor for refinement substantially improves solution quality: the gap decreases from 7.475% to 1.967% on TSP10K and from 10.34% to 4.250% on CVRP10K. This demonstrates that the learned sequence-level reconstructor can serve both as a building block for solution construction and as a reusable operator for local improvement.

Efficiency. SeqDiCO maintains a favorable quality–efficiency trade-off as the problem size increases. Because each model invocation operates on a local sequence subproblem rather than a full-instance solution representation, inference remains practical on 1K- and 10K-node instances while preserving

Table 4: Effect of reconstruction granularity on synthetic TSP. The global variant uses the same learning and inference components as SeqDiCO but performs over the full-instance representation.

Representation	TSP100			TSP1K			TSP10K		
	Length	Gap	Time	Length	Gap	Time	Length	Gap	Time
Global	7.763	0.000%	9.15m	32.257	39.526%	22.25m	-	-	-
Sequence-level	**7.763**	**0.000%**	24.1m	**23.188**	**0.281%**	4.4m	**73.190**	**1.967%**	4.64m

Table 5: Local reconstruction transfer from TSP100 to TSP1000.

Keep ratio	Reconstruction gap (%)	
	TSP100	TSP1000
0.1	0.099	0.392
0.3	**0.032**	**0.067**
0.5	0.077	0.103
0.7	0.207	0.394
0.9	0.824	0.797

Table 6: Effect of construction commit length on TSP1K.

Commit length	Gap	Time
1	**2.099%**	0.500m
2	2.252%	0.317m
5	2.574%	0.200m
10	2.759%	0.167m
20	3.466%	**0.133m**

strong solution quality. In particular, SeqDiCO solves TSP10K and CVRP10K within a few minutes for the full test sets, whereas several neural baselines incur substantially higher computational costs or cannot directly scale to these settings.

Real-world benchmarks. Table 3 reports results on TSPLIB and CVRPLIB instances. SeqDiCO achieves the best average gap across all reported subsets, with overall gaps of 0.125% on TSPLIB and 1.990% on CVRPLIB. Compared with L2C-Insert, these correspond to reductions from 0.308% and 3.201%, respectively. The consistent performance on real-world instances further indicates that the learned local reconstructor transfers beyond the training distribution to various spatial structures.

4.2 Understanding Cross-Scale Generalization

We further analyze the cross-scale generalization of SeqDiCO from three perspectives.

Effect of representation granularity. We construct a global variant that retains the same masking objective, backbone, and decoding procedure, but performs reconstruction over the full-instance adjacency representation. As shown in Table 4, both variants perform nearly optimally on TSP100, whereas the global variant degrades to a 39.526% gap on TSP1K and fails to scale to TSP10K, compared with 0.281% and 1.967% for the sequence-level representation. This controlled comparison demonstrates the importance of local reconstruction granularity for cross-scale generalization.

Transferability of local reconstruction. We evaluate the same TSP100-trained reconstructor on 100-node fixed-endpoint subpaths extracted from TSP100 and TSP1000 solutions. As shown in Table 5, reconstruction quality remains stable across global problem scales at moderate corruption levels; for example, at a keep ratio of 0.3, the gap changes only from 0.032% to 0.067% on TSP1000. This result supports the transferability of the learned local reconstruction task across instance scales.

Extrapolation to longer subproblems. Finally, we vary the inference subproblem length beyond the training range of 30–100. Figure 2 shows that increasing the local length initially improves solution quality by providing richer context, with the best performance occurring beyond the training range, while overly long subproblems eventually degrade performance due to increased reconstruction difficulty. These results indicate that the learned reconstructor can moderately extrapolate beyond the local scales observed during training.

4.3 Ablation Study

Effect of construction commit length. We vary the number of local decisions committed after each reconstruction during construction on TSP1K. As shown in Table 6, committing a single move achieves the best solution quality, while committing more moves progressively reduces construction time at the cost of solution quality. For example, increasing the commit length to 10 reduces

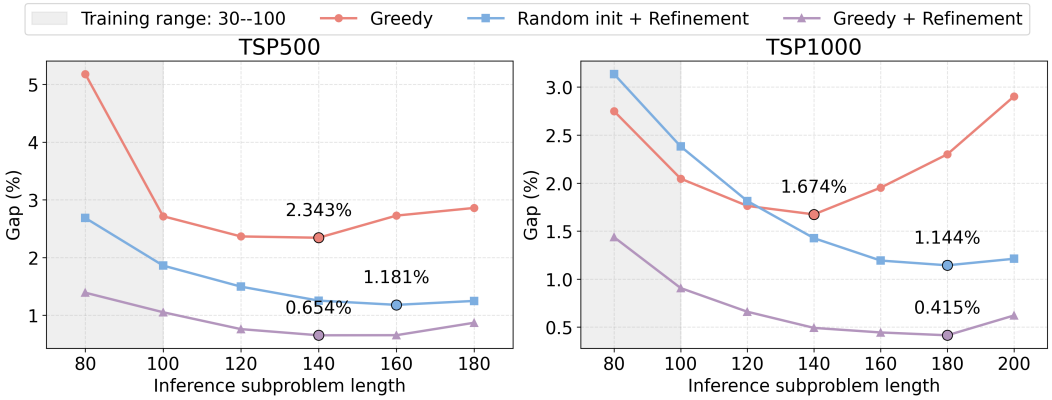


Figure 2: Effect of inference subproblem length on TSP500 and TSP1000. The shaded region denotes the training subpath length range, i.e., 30–100, sampled from TSP100 instances.

Table 7: Effect of diffusion-based reconstruction. The non-diffusion variant retains the same local representation and inference procedure but directly predicts the clean local structure.

Method	TSP			CVRP		
	100	1K	10K	100	1K	10K
Non-diffusion	0.004%	2.947%	16.149%	3.859%	15.195%	27.333%
SeqDiCO	**0.000%**	**0.281%**	**1.967%**	**0.166%**	**2.870%**	**4.250%**

the runtime from 0.5m to 0.167m, with the gap increasing to 2.759%. We therefore use a single committed move by default, while larger commit lengths provide a direct quality–efficiency trade-off.

Effect of diffusion-based reconstruction. To isolate the contribution of diffusion, we compare SeqDiCO with a non-diffusion variant that retains the same fixed-endpoint representation, backbone, candidate selection, decoder, and inference procedure, but directly predicts the clean local structure without corruption-based reconstruction. As shown in Table 7, the two variants perform similarly on TSP100, while the difference grows substantially with problem scale. The non-diffusion variant reaches gaps of 2.947% and 16.149% on TSP1K and TSP10K, compared with 0.281% and 1.967% for SeqDiCO; a similar trend appears on CVRP. This shows that sequence-level representation and diffusion-based reconstruction provide complementary benefits for cross-scale generalization.

5 Conclusion

In this work, we introduced SeqDiCO, a sequence-oriented diffusion framework for scale-generalizable neural combinatorial optimization. By shifting diffusion from full-instance solution structures to fixed-endpoint sequence subproblems, SeqDiCO learns a local reconstructor that can be composed across problem sizes and reused for both solution construction and refinement. Experiments on TSP and CVRP demonstrate strong cross-scale generalization from 100-node training instances to substantially larger synthetic and real-world problems, suggesting sequence-level reconstruction as a promising modeling granularity for scalable diffusion-based neural solvers.

Acknowledgement

This work was partly supported by the National Key R&D Program of China (2023YFF0905400), the National Natural Science Foundation of China (U2341229), the Fundamental and Interdisciplinary Disciplines Breakthrough Plan of the Ministry of Education of China (JYB2025XDXM903), the New Cornerstone Science Foundation through the XPLORER PRIZE, and the Natural Science Foundation of Jilin Province (YDZJ202601ZYTS441).

References

[1] Keld Helsgaun. An extension of the lin-kernighan-helsgaun tsp solver for constrained traveling salesman and vehicle routing problems. *Roskilde: Roskilde University*, 12:966–980, 2017.

[2] Yoshua Bengio, Andrea Lodi, and Antoine Prouvost. Machine learning for combinatorial optimization: a methodological tour d'horizon. *European Journal of Operational Research*, 290(2):405–421, 2021.

[3] George B Dantzig and John H Ramser. The truck dispatching problem. *Management science*, 6 (1):80–91, 1959.

[4] Geoff Clarke and John W Wright. Scheduling of vehicles from a central depot to a number of delivery points. *Operations research*, 12(4):568–581, 1964.

[5] Jean-François Cordeau and Gilbert Laporte. A tabu search heuristic for the static multi-vehicle dial-a-ride problem. *Transportation Research Part B: Methodological*, 37(6):579–594, 2003.

[6] Vera C Hemmelmayr, Jean-François Cordeau, and Teodor Gabriel Crainic. An adaptive large neighborhood search heuristic for two-echelon vehicle routing problems arising in city logistics. *Computers & operations research*, 39(12):3215–3228, 2012.

[7] Jesús Alegre, Manuel Laguna, and Joaquín Pacheco. Optimizing the periodic pick-up of raw materials for a manufacturer of auto parts. *European Journal of Operational Research*, 179(3): 736–746, 2007.

[8] Thibaut Vidal, Teodor Gabriel Crainic, Michel Gendreau, and Christian Prins. Heuristics for multi-attribute vehicle routing problems: A survey and synthesis. *European Journal of Operational Research*, 231(1):1–21, 2013.

[9] Eugene L Lawler and David E Wood. Branch-and-bound methods: A survey. *Operations research*, 14(4):699–719, 1966.

[10] Aigerim Bogyrbayeva, Meraryslan Meraliyev, Taukekhan Mustakhov, and Bissenbay Dauletbayev. Machine learning to solve vehicle routing problems: A survey. *IEEE Transactions on Intelligent Transportation Systems*, 25(6):4754–4772, 2024.

[11] Yunpeng Ba, Xi Lin, Changliang Zhou, Ruihao Zheng, Zhenkun Wang, Xinyan Liang, Zhichao Lu, Jianyong Sun, Yuhua Qian, and Qingfu Zhang. Survey on neural routing solvers. *arXiv preprint arXiv:2602.21761*, 2026.

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

[13] Yeong-Dae Kwon, Jinho Choo, Byoungjip Kim, Iljoo Yoon, Youngjune Gwon, and Seungjai Min. Pomo: Policy optimization with multiple optima for reinforcement learning. *Advances in neural information processing systems*, 33:21188–21198, 2020.

[14] Darko Drakulic, Sofia Michel, Florian Mai, Arnaud Sors, and Jean-Marc Andreoli. Bq-nco: Bisimulation quotienting for efficient neural combinatorial optimization. *Advances in Neural Information Processing Systems*, 36:77416–77429, 2023.

[15] Fu Luo, Xi Lin, Fei Liu, Qingfu Zhang, and Zhenkun Wang. Neural combinatorial optimization with heavy decoder: Toward large scale generalization. *Advances in Neural Information Processing Systems*, 36:8845–8864, 2023.

[16] Chaitanya K Joshi, Thomas Laurent, and Xavier Bresson. An efficient graph convolutional network technique for the travelling salesman problem. *arXiv preprint arXiv:1906.01227*, 2019.

[17] Ruizhong Qiu, Zhiqing Sun, and Yiming Yang. Dimes: A differentiable meta solver for combinatorial optimization problems. *Advances in Neural Information Processing Systems*, 35: 25531–25546, 2022.

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

[19] Yang Li, Lvda Chen, Haonan Wang, Runzhong Wang, and Junchi Yan. Generation as search operator for test-time scaling of diffusion-based combinatorial optimization. *Advances in Neural Information Processing Systems*, 38:127168–127196, 2026.

[20] Yu Wang, Yang Li, Junchi Yan, and Yi Chang. Strudico: Structured denoising diffusion with gradient-free inference-stage boosting for memory and time efficient combinatorial optimization. *Advances in Neural Information Processing Systems*, 38:71201–71228, 2026.

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

[22] 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.

[23] 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.

[24] 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, 2024.

[25] Haoyu Lei, Kaiwen Zhou, Yinchuan Li, Zhitang Chen, and Farzan Farnia. Boosting cross-problem generalization in diffusion-based neural combinatorial solver via inference time adaptation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pages 36289–36297, 2026.

[26] Yu Wang, Yang Li, Jiale Ma, Junchi Yan, and Yi Chang. Native adaptive solution expansion for diffusion-based combinatorial optimization. In *International Conference on Learning Representations*, volume 2026, pages 36466–36486, 2026.

[27] 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, pages 7474–7482, 2021.

[28] Xinyun Chen and Yuandong Tian. Learning to perform local rewriting for combinatorial optimization. *Advances in neural information processing systems*, 32, 2019.

[29] Yaoxin Wu, Wen Song, Zhiguang Cao, Jie Zhang, and Andrew Lim. Learning improvement heuristics for solving routing problems. *IEEE transactions on neural networks and learning systems*, 33(9):5057–5069, 2021.

[30] Paulo R d O Costa, Jason Rhuggenaath, Yingqian Zhang, and Alp Akcay. Learning 2-opt heuristics for the traveling salesman problem via deep reinforcement learning. In *Asian conference on machine learning*, pages 465–480. PMLR, 2020.

[31] Yining Ma, Jingwen Li, Zhiguang Cao, Wen Song, Le Zhang, Zhenghua Chen, and Jing Tang. Learning to iteratively solve routing problems with dual-aspect collaborative transformer. *Advances in Neural Information Processing Systems*, 34:11096–11107, 2021.

[32] Fu Luo, Xi Lin, Mengyuan Zhong, Fei Liu, Zhenkun Wang, Jianyong Sun, and Qingfu Zhang. Learning to insert for constructive neural vehicle routing solver. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025.

[33] Ke Li, Fei Liu, Zhenkun Wang, and Qingfu Zhang. Destroy and repair using hyper-graphs for routing. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pages 18341–18349, 2025.

[34] Lvda Chen, Yang Li, and Junchi Yan. Maskco: Masked generation drives effective representation learning and exploiting for combinatorial optimization. In *International Conference on Learning Representations*, volume 2026, pages 133725–133753, 2026.

[35] Chengrui Gao, Haopu Shang, Ke Xue, Dong Li, and Chao Qian. Towards generalizable neural solvers for vehicle routing problems via ensemble with transferrable local policy. *arXiv preprint arXiv:2308.14104*, 2023.

[36] Han Fang, Zhihao Song, Paul Weng, and Yutong Ban. Invit: A generalizable routing problem solver with invariant nested view transformer. In *International Conference on Machine Learning*, pages 12973–12992. PMLR, 2024.

[37] Xuanhao Pan, Yan Jin, Yuandong Ding, Mingxiao Feng, Li Zhao, Lei Song, and Jiang Bian. H-tsp: Hierarchically solving the large-scale traveling salesman problem. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, pages 9345–9353, 2023.

[38] Qingchun Hou, Jingwei Yang, Yiqiang Su, Xiaoqing Wang, and Yuming Deng. Generalize learned heuristics to solve large-scale vehicle routing problems in real-time. In *The Eleventh International Conference on Learning Representations*, 2023.

[39] Minsu Kim, Jinkyoo Park, et al. Learning collaborative policies to solve np-hard routing problems. *Advances in Neural Information Processing Systems*, 34:10418–10430, 2021.

[40] Zhi Zheng, Changliang Zhou, Xialiang Tong, Mingxuan Yuan, and Zhenkun Wang. Udc: A unified neural divide-and-conquer framework for large-scale combinatorial optimization problems. *Advances in Neural Information Processing Systems*, 37:6081–6125, 2024.

[41] Haoran Ye, Jiarui Wang, Helan Liang, Zhiguang Cao, Yong Li, and Fanzhang Li. Glop: Learning global partition and local construction for solving large-scale routing problems in real-time. In *Proceedings of the AAAI conference on artificial intelligence*, volume 38, pages 20284–20292, 2024.

[42] Fu Luo, Xi Lin, Yaoxin Wu, Zhenkun Wang, Tong Xialiang, Mingxuan Yuan, and Qingfu Zhang. Boosting neural combinatorial optimization for large-scale vehicle routing problems. In *The Thirteenth International Conference on Learning Representations*, 2025.

[43] Thibaut Vidal. Hybrid genetic search for the cvrp: Open-source implementation and swap* neighborhood. *Computers & Operations Research*, 140:105643, 2022.

[44] Gerhard Reinelt. Tsplib—a traveling salesman problem library. *ORSA journal on computing*, 3 (4):376–384, 1991.

[45] Eduardo Uchoa, Diego Pecin, Artur Pessoa, Marcus Poggi, Thibaut Vidal, and Anand Subramanian. New benchmark instances for the capacitated vehicle routing problem. *European Journal of Operational Research*, 257(3):845–858, 2017.

[46] Benjamin Hudson, Qingbiao Li, Matthew Malencia, and Amanda Prorok. Graph neural network guided local search for the traveling salesperson problem. In *International Conference on Learning Representations*, 2022.

[47] Liang Xin, Wen Song, Zhiguang Cao, and Jie Zhang. Neurolkh: Combining deep learning model with lin-kernighan-helsgaun heuristic for solving the traveling salesman problem. *Advances in Neural Information Processing Systems*, 34:7472–7483, 2021.

[48] Wouter Kool, Herke van Hoof, and Max Welling. Attention, learn to solve routing problems! In *International Conference on Learning Representations*, 2019.

[49] Hanni Cheng, Haosi Zheng, Ya Cong, Weihao Jiang, and Shiliang Pu. Select and optimize: Learning to solve large-scale tsp instances. In *International conference on artificial intelligence and statistics*, pages 1219–1231. PMLR, 2023.

Appendix

A Broad Impacts, Limitations and Future Work

Broad impacts. SeqDiCO aims to improve the scalability of neural solvers for routing problems by reducing the dependence on large-scale labeled instances. This may benefit applications where high-quality routing decisions are needed under limited computational or labeling budgets, such as logistics planning, transportation scheduling, and resource allocation. By learning reusable local reconstruction patterns from small instances, the proposed framework can potentially lower the cost of adapting neural combinatorial optimization methods to larger problem sizes. At the same time, routing and logistics technologies can have broader social and economic effects, including changes in labor allocation, delivery efficiency, and energy consumption. Therefore, deployment in real-world decision systems should consider domain-specific constraints, fairness requirements, safety margins, and human oversight.

Limitations. The cross-scale generalization of SeqDiCO is empirical and does not provide formal approximation guarantees on solution quality or accumulated suboptimality when local reconstructions are repeatedly composed. Feasibility is enforced algorithmically during construction and refinement rather than guaranteed by the learned diffusion model itself. Moreover, although local reconstruction substantially reduces the dependence on global problem size, its effectiveness can still be affected by shifts in local problem structure and the choice of subproblem scale. Developing stronger theoretical characterizations and more adaptive local reconstruction mechanisms remains an important direction for future work.

B Detailed Related Work

Neural Combinatorial Optimization. Neural combinatorial optimization (NCO) learns data-driven solvers for combinatorial optimization by replacing manually designed heuristic rules with neural decision policies [2]. For routing problems such as TSP and CVRP, existing neural solvers can be broadly organized along two axes: the role they play in the search process and the way they generate solution components. From the first perspective, solvers are commonly divided into construction-based methods, which build solutions from scratch, and improvement-based methods, which refine existing solutions through iterative modifications [10, 11]. From the second perspective, solvers can be categorized as autoregressive or non-autoregressive, depending on whether solution components are generated sequentially or in parallel.

Autoregressive construction methods select nodes or edges step by step conditioned on the current partial solution [12, 13, 14, 15]. Their sequential decision process naturally matches the ordered structure of routing solutions and supports variable-size generation, but it also limits parallelism and becomes costly on large-scale instances. Non-autoregressive construction methods instead predict multiple solution components in parallel, often as edge probabilities or heatmaps, and then decode feasible solutions from these predictions [16, 17]. These methods improve inference efficiency, but their full-instance predictions often require additional decoding or post-processing and may be sensitive to the problem size used during training.

Improvement-based solvers start from an initial solution and learn to guide local modifications, closely related to local search and large neighborhood search [28, 29]. Some methods select small local moves, such as swaps or edge exchanges [30], while others modify larger solution segments through destroy-and-repair or reconstruction procedures [31, 32, 33]. These improvement mechanisms can be instantiated either autoregressively, by decomposing a move into sequential decisions, or non-autoregressively, by predicting scores, heatmaps, or reconstructed structures in parallel [18, 46, 47]. SeqDiCO is related to both construction and improvement methods, but differs in its operator design: it learns a sequence-oriented diffusion reconstructor that can be used both to grow solutions and to replace local subpaths, rather than relying on a purely autoregressive policy or a task-specific local move operator.

Generative Models for Combinatorial Optimization. Generative models have recently become an important direction in NCO because they can represent distributions over high-quality solutions rather than learning a single deterministic mapping [18]. Diffusion models are particularly relevant

to routing problems, as they solve CO through a denoising or reconstruction process that recovers clean solution structures from corrupted observations [21, 22]. Existing diffusion-based CO solvers commonly represent routing solutions as edge sets, adjacency matrices, or heatmaps, and define the denoising process over the entire instance-level solution structure [18, 20, 23, 24, 25, 26]. This makes diffusion a representative non-autoregressive generative paradigm for parallel solution prediction [19].

Recent diffusion-based solvers mainly improve the full-instance denoising pipeline from different angles. DIFUSCO learns graph-based diffusion over solution adjacency structures [18]. T2T and Fast T2T improve the training-to-testing transition through gradient-based search and optimization consistency [23, 24]. Other recent studies further enhance diffusion-based CO through structured denoising, inference-time adaptation, native solution expansion, or iterative re-generation during test-time search [19, 20, 25, 26]. These methods improve denoising objectives, training consistency, or inference strategies, but they still largely operate on full-instance solution representations. As a result, the learned reconstruction distribution can remain tied to the solution space and representation size seen during training [15, 27].

SeqDiCO is orthogonal to these improvements. Instead of designing a stronger objective or search procedure within the same full-instance diffusion space, we change the modeling granularity of diffusion. The denoising target in SeqDiCO is not a complete solution of the whole instance, but a fixed-endpoint sequence subproblem induced by a local ordered segment. This allows the model to preserve parallel reconstruction while learning a reusable local operator that can be composed across different problem sizes. In this sense, SeqDiCO does not challenge the distributional view of diffusion-based CO; rather, it redefines the object over which the reconstruction distribution is learned.

Scale Generalization and Local Reconstruction. Scale generalization is a central challenge for neural routing solvers: models trained on small instances are expected to solve much larger ones, although the solution space, representation size, and dependency structure all change with problem size. Autoregressive methods naturally support variable-length generation, but their sequential inference can be expensive on large instances. Non-autoregressive and diffusion-based methods offer more parallelism, but often rely on full-instance representations that are coupled with the training scale. Existing approaches address this issue through improved architectures, such as stronger decoders, dynamic state representations, distance-aware modules, and local-view attention [35, 36]; through search-space reduction based on geometric proximity; or through decomposition-based frameworks that solve smaller subproblems and integrate them into a complete solution [37, 38, 39, 40, 41].

These works are closely related to SeqDiCO, but differ in how local information is used. INViT improves scale generalization through invariant nested views and local representations [36]. LEHD strengthens autoregressive decoding for large-scale generalization by using a heavier decoder [15]. Decomposition-based methods such as H-TSP, GLOP, and UDC divide large instances into smaller subproblems and then merge or coordinate their solutions [37, 40, 41]. Such methods demonstrate the value of local or decomposed reasoning, but they usually rely on specialized architectures, external partitioning procedures, or problem-specific integration pipelines. SeqDiCO instead learns a single diffusion-style local reconstruction operator. The same operator is trained on local sequence subproblems and then repeatedly composed during inference, without requiring the model to predict a full-instance solution at once.

Our work is also connected to sequence modeling and local reconstruction for routing. Routing solutions are inherently ordered structures rather than unordered edge sets, and several methods have explored insertion-based construction, partial-solution reconstruction, and large-neighborhood repair [15, 42]. L2C-Insert, for example, emphasizes insertion-based construction and local decision making for large-scale routing [42]. These methods suggest that local sequence operations can provide a useful balance between flexibility, scalability, and solution quality. Different from autoregressive sequence construction or task-specific repair pipelines, SeqDiCO instantiates local reconstruction through sequence-oriented mask diffusion. The modeled object is a reusable fixed-endpoint sequence subproblem whose clean open-path adjacency is reconstructed from corrupted observations. This combines the structural bias of sequence modeling, the parallel reconstruction capability of diffusion-style generation, and the scale transferability of local subproblem solving.

C Model Architecture Details

This appendix provides the detailed architecture of the sequence-oriented diffusion model used in SeqDiCO. The model takes a local sequence reconstruction task as input and predicts pairwise edge logits for reconstructing the clean sequence-induced adjacency. Let L denote the number of nodes in a local subproblem, d_{model} the embedding dimension, h the number of attention heads, and $d_{\text{head}} = d_{\text{model}}/h$ the head dimension. For each local task, the input consists of node features $\mathbf{F} \in \mathbb{R}^{L \times d_{\text{in}}}$, endpoint-role indicators $\boldsymbol{\rho} \in \{0,1\}^L$, a diffusion timestep τ, and a corrupted adjacency matrix $\mathbf{A}_\tau \in \{0,1\}^{L \times L}$.

C.1 Input Representation

Each node feature vector is first projected into the model embedding space:

$$\mathbf{E} = \mathbf{F}\mathbf{W}_{\text{in}} + \mathbf{b}_{\text{in}}, \qquad \mathbf{E} \in \mathbb{R}^{L \times d_{\text{model}}}, \tag{14}$$

where $\mathbf{W}_{\text{in}} \in \mathbb{R}^{d_{\text{in}} \times d_{\text{model}}}$ and $\mathbf{b}_{\text{in}} \in \mathbb{R}^{d_{\text{model}}}$ are learnable parameters. For endpoint-conditioned reconstruction, we use a learnable role embedding table $\mathbf{R} \in \mathbb{R}^{2 \times d_{\text{model}}}$ to indicate whether a node belongs to the endpoint set $\mathcal{B}$:

$$\tilde{\mathbf{E}}_i = \mathbf{E}_i + \mathbf{R}_{\rho_i}, \qquad \rho_i = \mathbb{I}[v_i \in \mathcal{B}]. \tag{15}$$

The two endpoints are assigned the same role type, so the endpoint set is encoded symmetrically. The resulting embeddings $\tilde{\mathbf{E}}$ are used as the input to the encoder.

C.2 Transformer Encoder

The encoder consists of N_{enc} Transformer layers. Given the input hidden states $\mathbf{X}^{(0)} = \tilde{\mathbf{E}}$, each encoder layer updates the representations through pre-norm multi-head self-attention followed by a feed-forward network:

$$\mathbf{U}^{(\ell)} = \mathbf{X}^{(\ell)} + \text{MHSA}^{(\ell)}_{\text{enc}}\left(\text{Norm}(\mathbf{X}^{(\ell)})\right), \tag{16}$$

$$\mathbf{X}^{(\ell+1)} = \mathbf{U}^{(\ell)} + \text{FFN}^{(\ell)}_{\text{enc}}\left(\text{Norm}(\mathbf{U}^{(\ell)})\right), \tag{17}$$

for $\ell = 0, \ldots, N_{\text{enc}} - 1$. The encoder output is denoted as

$$\mathbf{H} = \mathbf{X}^{(N_{\text{enc}})} \in \mathbb{R}^{L \times d_{\text{model}}}. \tag{18}$$

The encoder captures contextual relations among nodes in the local subproblem based on their geometric features and endpoint roles.

C.3 Timestep-conditioned Decoder Input

Before decoding, the encoder output is projected into the decoder embedding space:

$$\mathbf{G} = \mathbf{H}\mathbf{W}_{\text{mid}} + \mathbf{b}_{\text{mid}}, \qquad \mathbf{G} \in \mathbb{R}^{L \times d_{\text{model}}}, \tag{19}$$

where $\mathbf{W}_{\text{mid}}$ and $\mathbf{b}_{\text{mid}}$ are learnable parameters. The diffusion timestep τ is embedded by a timestep embedding module $\phi(\cdot)$:

$$\mathbf{t}_\tau = \phi(\tau) \in \mathbb{R}^{d_{\text{model}}}. \tag{20}$$

We add this timestep embedding to every node representation:

$$\mathbf{Y}^{(0)}_i = \mathbf{G}_i + \mathbf{t}_\tau, \qquad i = 1, \ldots, L. \tag{21}$$

This allows the decoder to adapt its reconstruction behavior according to the corruption level.

C.4 Adjacency-biased Multi-head Self-attention

The decoder uses the corrupted adjacency matrix $\mathbf{A}_\tau$ as structural guidance. Given decoder hidden states $\mathbf{Y} \in \mathbb{R}^{L \times d_{\text{model}}}$, each attention head computes

$$\mathbf{Q}_r = \mathbf{Y}\mathbf{W}^Q_r, \quad \mathbf{K}_r = \mathbf{Y}\mathbf{W}^K_r, \quad \mathbf{V}_r = \mathbf{Y}\mathbf{W}^V_r, \tag{22}$$

where $\mathbf{W}_r^Q, \mathbf{W}_r^K, \mathbf{W}_r^V \in \mathbb{R}^{d_{\text{model}} \times d_{\text{head}}}$ are learnable projections for head r. The corrupted adjacency is injected into the attention logits as an additive structural bias:

$$\text{Attn}_r(\mathbf{Y}, \mathbf{A}_\tau) = \text{Softmax}\left(\frac{\mathbf{Q}_r \mathbf{K}_r^\top}{\sqrt{d_{\text{head}}}} + \lambda \mathbf{A}_\tau\right)\mathbf{V}_r, \tag{23}$$

where λ controls the strength of the structural bias. The outputs of all heads are concatenated and linearly projected:

$$\text{MHSA}_{\text{dec}}(\mathbf{Y}, \mathbf{A}_\tau) = \text{Concat}\left(\text{Attn}_1, \ldots, \text{Attn}_h\right)\mathbf{W}^O. \tag{24}$$

This adjacency-biased attention enables the decoder to use visible edges in $\mathbf{A}_\tau$ while inferring missing sequence-induced connections.

C.5 Transformer Decoder

The decoder contains N_{dec} Transformer layers. Each layer follows the pre-norm residual structure:

$$\mathbf{V}^{(\ell)} = \mathbf{Y}^{(\ell)} + \text{MHSA}_{\text{dec}}^{(\ell)}\left(\text{Norm}(\mathbf{Y}^{(\ell)}), \mathbf{A}_\tau\right), \tag{25}$$

$$\mathbf{Y}^{(\ell+1)} = \mathbf{V}^{(\ell)} + \text{FFN}_{\text{dec}}^{(\ell)}\left(\text{Norm}(\mathbf{V}^{(\ell)})\right), \tag{26}$$

for $\ell = 0, \ldots, N_{\text{dec}} - 1$. The decoder output is

$$\mathbf{Z} = \mathbf{Y}^{(N_{\text{dec}})} \in \mathbb{R}^{L \times d_{\text{model}}}. \tag{27}$$

In our implementation, the feed-forward sublayer can be instantiated with an MLP or SwiGLU block.

C.6 Pairwise Edge Prediction Head

After decoding, we apply a lightweight feed-forward projection to refine node embeddings:

$$\bar{\mathbf{Z}} = \text{MLP}_{\text{out}}(\mathbf{Z}). \tag{28}$$

The pairwise edge logits are computed by inner products between node embeddings:

$$\mathbf{S} = \bar{\mathbf{Z}}\bar{\mathbf{Z}}^\top, \qquad \mathbf{S}_{ij} = \bar{\mathbf{z}}_i^\top \bar{\mathbf{z}}_j. \tag{29}$$

The matrix $\mathbf{S} \in \mathbb{R}^{L \times L}$ parameterizes the edge probabilities of the reconstructed local structure:

$$\hat{\mathbf{A}}_{0,ij} = \sigma(\mathbf{S}_{ij}), \tag{30}$$

where $\sigma(\cdot)$ denotes the sigmoid function. During training, $\mathbf{S}$ is supervised by the clean sequence-induced adjacency $\mathbf{A}_0$. During inference, $\mathbf{S}$ is decoded into an open path connecting the endpoint set $\mathcal{B}$.

C.7 Discussion

The architecture has two important properties. First, it performs computation primarily on node representations and avoids explicit edge-feature networks, which reduces the overhead of repeatedly applying the diffusion operator during search. Second, structural information is still incorporated through the corrupted adjacency matrix $\mathbf{A}_\tau$ as an attention bias in the decoder. Therefore, the model can exploit partially observed sequence structure while maintaining efficient parallel edge prediction.

D Additional Experimental Details

Datasets. For synthetic benchmarks, we follow the standard generation protocol in neural routing literature, where node coordinates are sampled uniformly from the unit square. For each problem, the training set contains one million 100-node instances. TSP labels are generated by Concorde, and CVRP labels are generated by HGS. Different from full-instance diffusion solvers, SeqDiCO does not directly learn to generate complete 100-node solutions. Instead, it extracts fixed-size sequence reconstruction tasks from the labeled tours and trains the model on these local subproblems.

For evaluation, we test on synthetic instances with 100, 1K, and 10K nodes. We denote each setting as TSPN or CVRPN. Following common practice, TSP100 and CVRP100 contain 10,000 instances; TSP1K contains 128 instances; CVRP1K contains 100 instances; and each 10K-node setting contains 16 instances. The vehicle capacity is set to 50 for CVRP100, 200 for CVRP1K, and 300 for larger CVRP instances. We also evaluate on real-world EUC_2D instances from TSPLIB [44] and CVRPLIB Set-X [45] with no more than 1K nodes to test robustness under irregular spatial distributions.

Model and training. SeqDiCO adopts a Transformer-based encoder-decoder with a timestep-conditioned decoder and a pairwise edge prediction head. The default model uses embedding dimension $d = 256$, 8 attention heads, 16 encoder layers, and 6 decoder layers. The model is trained with supervised reconstruction on sequence subproblems sampled from high-quality tours. At each training step, we sample a local sequence, perturb its intermediate nodes, corrupt its clean adjacency according to a sampled timestep, and train the model to reconstruct the clean open-path adjacency. We use the Muon optimizer and batch size 1024. Unless otherwise specified, all models are trained only on 100-node instances and evaluated on larger scales without fine-tuning.

Baselines. We compare against both classical solvers and representative neural solvers. Classical baselines include Concorde and LKH3 [1] for TSP, and HGS [43] for CVRP. Neural baselines include autoregressive constructive methods such as POMO [13], INViT [36], ELG [35], BQ [14], LEHD [15], and L2C-Insert [32]; non-autoregressive or heatmap-based methods such as Att-GCN+MCTS [48], DIFUSCO [18], T2T [23], StruDiCO [20], NExCO [26], and MaskCO [34]; decomposition-based methods such as H-TSP [37], SO-mixed [49], GLOP [41], and UDC [40]; and improvement-based methods such as NeuOpt [47]. We use public implementations and reported configurations whenever available. Results marked with an asterisk are taken from the original papers.

Inference configuration. During inference, SeqDiCO applies the learned sequence-oriented reconstruction operator to local fixed-size tasks. For construction, it repeatedly selects a candidate set, reconstructs a local open path, and commits only the next move. For refinement, it reconstructs selected solution segments and accepts the replacement if the objective improves. The number of refinement iterations is denoted by I. We report both the greedy construction result and the refined result to separate the contribution of construction from that of local refinement.

For large-scale instances, we restrict local candidates to nearest neighbors and apply coordinate normalization to reduce distribution shift between local subproblems sampled during training and those formed during inference. Reference solutions are obtained from Concorde or LKH3 for TSP, HGS for CVRP, and best-known solutions for real-world benchmarks.

Runtime measurement. We report the total inference time required to solve the full test set. Runtime comparisons should be interpreted with care: neural solvers are evaluated on GPU, whereas classical solvers are evaluated with their standard CPU implementations. Therefore, our main runtime comparison focuses on neural baselines, while classical solvers mainly serve as solution-quality references. All neural solvers are evaluated on *NVIDIA GeForce RTX 3090*, and classical solvers are evaluated on *AMD EPYC 7773X 64-Core Processor*.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The abstract and introduction explicitly state the claims made, including the contributions made in the paper (Sec. 1).

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: We discuss the limitations in Appendix A.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [N/A]

Justification: The paper does not include theoretical results.

Guidelines:

- The answer [N/A] means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: The experimental details are in Appendix D.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
     (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
     (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
     (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
     (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

Answer: [No]

Justification: The source code will be made publicly available upon acceptance.

Guidelines:

- The answer [N/A] means that paper does not include experiments requiring code.
- Please see the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
- The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
- The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes]

   Justification: The experimental details are in Appendix D.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [No]

   Justification: We follow the setting of previous works [15, 18] to report the average solution quality.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
   - The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
   - The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
   - The assumptions made should be given (e.g., Normally distributed errors).
   - It should be clear whether the error bar is the standard deviation or the standard error of the mean.

- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: The experimental details are in Appendix D.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: The research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [Yes]

    Justification: We discuss the borader impacts in Appendix A.

    Guidelines:

    - The answer [N/A] means that there is no societal impact of the work performed.
    - If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
    - Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.

- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

    Answer: [No]

    Justification: The paper poses no such risks.

    Guidelines:

    - The answer [N/A] means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: The original papers introducing the models and datasets used in this work are properly cited.

    Guidelines:

    - The answer [N/A] means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
    - If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
    - For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.

- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [N/A]

    Justification: Currently, the paper does not release new assets. Our source code will be released upon the acceptance of the paper with comprehensive documents.

    Guidelines:

    - The answer [N/A] means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [N/A]

    Justification: The paper does not involve crowdsourcing nor research with human subjects.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [N/A]

    Justification: This paper does not incur such risks.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
    - We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
    - For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

Answer: [N/A]

Justification: The core method development in this research does not involve LLMs as any important, original, or non-standard components.

Guidelines:

- The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
- Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.