SIZE GENERALIZATION FOR DIFFUSION ROUTING SOLVERS

Anonymous authors
Paper under double-blind review

ABSTRACT

Routing problems, like the Traveling Salesman Problem (TSP) and the Capacitated Vehicle Routing Problem (CVRP), are NP-hard and slow to solve for large instances. Diffusion-based solvers can be trained in a supervised way using sets of optimal solutions, and then quickly solve new test instances. However, for large instances, optimal solutions are hard to find, and diffusion models fail to generalize to instance sizes outside their training distribution. Here we describe ResizeCO, a label-free size-generalization approach that addresses both the change in statistical properties of larger instances, and the computational challenge of solving them quickly. It uses reinforcement learning on large-scale instances to adapt a diffusion model trained on small instances. Specifically, we adapt GRPO to this problem, using route length as the training signal. ResizeCO also formulates k-opt as a tree search, solved by a beam search on the GPU. In size-generalization experiments on TSP and CVRP instances with up to 10000 nodes, ResizeCO achieves a Pareto front over route quality and time that dominates learned solvers, and also outperforms other solvers at short run times, setting a new state of the art.

1 INTRODUCTION

Combinatorial optimization (CO) problems, and specifically routing problems, appear in many real-world systems, from logistics and supply-chain planning to chip design and scheduling. Among these, probably the most well-studied and widely deployed are the *Traveling Salesman Problem* (TSP) and the *Capacitated Vehicle Routing Problem* (CVRP), both NP-hard. For small instances, heuristic approaches can efficiently find near-optimal solutions for these problems, but solving large instances could take hours. Solving within seconds would open new possibilities for real-world problems like on-the-fly routing, interactive "what-if" planning, and large-scale routing, like coordinating deliveries within a city.

Supervised learning methods offer a promising approach for fast combinatorial optimization, because the long training cost is amortized over many new instances, each solved quickly at inference (Joshi et al., 2019; Luo et al., 2023). Diffusion-based generative solvers, trained by adding noise to precomputed optimal solutions, now match or beat hand-designed heuristics when tested on in-distribution instances (Li et al., 2025; Chen et al., 2026). Unfortunately, supervised training on large instances may be prohibitively expensive because it requires creating a large number of near-optimal solutions as labels. This effect is quantified in Appendix D.1 for TSP and CVRP. At the same time, current supervised approaches also fail at size generalization, namely, diffusion models trained on small instances perform poorly on large instances. Together, scaling supervised training to large instances is challenging.

Here we describe ResizeCO, a label-free size-generalization approach, designed to generalize from small to large instances. **ResizeCO trains a diffusion model on small instances and then fine-tunes the model using reinforcement learning (RL) on large-scale instances.** ResizeCO tackles the two main challenges of size generalization: the fact that statistical properties of instances may change with size, and the efficiency challenge of optimizing larger instances. ResizeCO addresses these using an integrative approach that handles architecture, fine-tuning and inference. (1) Architecture: We design a variant of Graph Transformer to generalize better across instance sizes. (2) RL fine-tuning: We adapt GRPO to provide label-free fine-tuning of a diffusion model trained on small instances. Specifically, we use a node-walk decoder, which provides a better estimate of a trajectory probability

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.

given the model. (3) Inference: We design a new and efficient *beam k-opt search* that runs in parallel on the GPU, which improves inference quality.

We evaluate ResizeCO on TSP and CVRP and in several size-generalization setups, comparing to both machine-learning and fast non-learned approaches. ResizeCO achieves a new state-of-the-art (SoTA) quality for short run times for several size-generalization setups, and dominates learning-based methods over the full Pareto front.

Our main contributions are: (1) ResizeCO is the first diffusion solver trained on small instances and adapted to large instances without labels, becoming the first diffusion solver trained for CVRP instances of size 10000. (2) Using node-walk decoding, we introduce an exact probability model over the routes sampled from a masked diffusion solver, then used to weight GRPO trajectories. (3) An efficient formulation of k-opt as a tree search and a beam-search approach that can be run on the GPU to solve it efficiently. (4) A new SoTA in size generalization on both CVRP and TSP for short durations, and Pareto-domination over learning-based methods in terms of time vs. solution quality.

2 Related Work

Size Generalization in Neural Routing Solvers Supervised heatmap solvers, from the graph convolutional network of Joshi et al. (2019) to the diffusion-based solvers (Sun & Yang, 2023; Li et al., 2023; 2024; 2025; Chen et al., 2026), learn from labeled routes. None of them targets size generalization in its main method, so a model trained at one size runs unchanged on larger instances. However, a solver trained on small instances loses accuracy on larger ones (Joshi et al., 2022). Other methods reach larger sizes with size-robust inductive biases in autoregressive solvers (Luo et al., 2023; Drakulic et al., 2023; Fang et al., 2024; Huang et al., 2025; Gao et al., 2024), with meta-learning across sizes and distributions (Zhou et al., 2023), or by splitting a large instance into small subproblems (Ye et al., 2024; Hou et al., 2023; Li et al., 2021). In contrast, our method continues training at the target size, without labels.

Label-Free Training at the Target Size Label-free solvers learn from the route length, as a reinforcement learning reward (Kool et al., 2019; Kwon et al., 2020; Qiu et al., 2022) or inside an unsupervised loss (Min et al., 2023), and concurrent work trains autoregressive solvers this way with GRPO (Sepúlveda & Ruz, 2026). However, these solvers train from scratch, and Joshi et al. (2022) show that training from scratch on large instances is sample-inefficient, while a model trained on small instances improves further with fine-tuning at the large size. SIT (Luo et al., 2025a) and MaskCO's self-training variant (Chen et al., 2026) follow this approach, but imitate their own improved routes as pseudo-labels. CADO (Song et al., 2026) fine-tunes a supervised diffusion solver with RL instead, but only at its labeled training sizes. EAS (Hottung et al., 2022) also fine-tunes a trained model with RL, but per test instance. In contrast, our method fine-tunes a diffusion model supervised on small instances at the target size before test time, with GRPO and the route length as its reward.

Size-Generalizing Architectures Generative solvers encode instances with a message-passing network on a k-nearest-neighbor graph (Sun & Yang, 2023; Li et al., 2024) or a transformer (Chen et al., 2026), which gives a local or a global view, not both. Graph Transformers keep both by adding graph structure to attention and generalize better than message-passing networks out of distribution (Niv & Rabin, 2025; Wu et al., 2025), so we adapt the Graph Transformer of Chen et al. (2022).

***k*-opt Local Search** Diffusion-based solvers often post-process (also known as "polish") each decoded route with a local operator called 2-opt (Croes, 1958). It swaps two edges of the route for a shorter pair. LKH-3 (Helsgaun, 2017) chains a sequence of edge exchanges into deeper k-opt moves with Lin-Kernighan search (Lin & Kernighan, 1973; Helsgaun, 2000), and HGS (Vidal, 2022) combines 2-opt with other moves. However, a Lin-Kernighan chain grows one exchange at a time. Prior work already parallelized k-opt on GPUs, searching all moves of one route in parallel for k up to three (Rocki & Suda, 2012) or six (Qiao et al., 2026). In contrast, our method runs Lin-Kernighan chains as a GPU beam search, keeping only the best chains per step for many routes at once.

3 Preliminaries and Problem Definition

Routing as Edge Selection A routing instance is an undirected weighted graph $\mathbb{G} = (V, E)$ with $N = |V|$ nodes and edge lengths $d_{ij} = d_{ji}$. We encode a solution as a flattened binary adjacency

matrix $x \in \{0,1\}^{N^2}$, where $x_{ij} = x_{ji} = 1$ iff edge (i,j) participates in the route (see also Sun & Yang, 2023; Chen et al., 2026). The length of a route x is $\text{length}(x) = \frac{1}{2} d^\top x$. Routing problems minimize this length over Ω, the set of feasible solutions.

TSP and CVRP In TSP, Ω holds the routes that visit every node once and return to where they started (Hamiltonian cycles). In CVRP, one node is a depot and every other node i is a customer with a demand q_i. A route in Ω returns to the depot once per vehicle, so it consists of several *sub-routes* from the depot and back. Every customer is on exactly one sub-route, and the total demand on each sub-route is at most the vehicle capacity.

Masked Diffusion for CO Diffusion-based solvers learn a distribution over solutions by adding noise to optimal solutions during training and predicting that noise. Our pipeline is based on MaskCO (Chen et al., 2026). At inference, the denoiser receives a *partial solution* $\hat{x}$ and outputs an *edge heatmap* $p_\theta(\mathbb{G}, \hat{x}) \in [0,1]^{N^2}$. Starting with all edges masked, a decoder iteratively selects edges from this heatmap over K *unmasking steps*. The decoded route is improved with 2-opt (Croes, 1958). MaskCO can then run multiple correction cycles that re-mask part of the current route, rebuild it, and retain the best solution (see Figure 1C). The decoders we use are described in Appendix B.4.

During supervised training, each edge of an optimal route $x^\star$ is masked independently at a rate $p \sim \mathcal{U}(0,1)$, and masking edge (i,j) masks the two entries x_{ij} and x_{ji}. For TSP, the loss averages over the $2N$ entries of $x^\star$ that equal one, $\mathcal{L}_{\text{SL}}(\theta) = \text{CE}\big(p_\theta(\mathbb{G}, \hat{x}), x^\star\big) = -\frac{1}{2N} \sum_{(i,j)\,:\,x^\star_{ij}=1} \log p_{\theta,ij}$.

Group Relative Policy Optimization GRPO (Shao et al., 2024) is a reinforcement learning algorithm that uses a sampled group's mean as its own baseline instead of a value network. For each input, the policy π_θ samples a *group* of G outputs. Output g receives a reward R_g, and its *advantage* A_g is the reward standardized within the group, $A_g = \big(R_g - \text{mean}(R_1 \dots R_G)\big) / \text{std}(R_1 \dots R_G)$. For generators with T steps, such as flow and masked diffusion models (Liu et al., 2025; Luo et al., 2025b; Zhou et al., 2025), GRPO treats generation as a T-step Markov decision process with actions a_t taken from states s_t by the policy $\pi_{\theta_{\text{old}}}$ that sampled the group. The loss raises the probability of actions with positive advantage, lowers it for negative ones, and keeps the policy near the *reference* π_{ref}, a frozen copy of the starting model:

$$\mathcal{L}_{\text{GRPO}}(\theta) = \frac{1}{G} \sum_{g=1}^{G} \frac{1}{T} \sum_{t=1}^{T} \Big[-\min\big(r_t^g(\theta)\, A_g, \text{clip}\big(r_t^g, 1-\varepsilon, 1+\varepsilon\big) A_g\big) + \beta\, D_{\text{KL}}\big(\pi_\theta \,\|\, \pi_{\text{ref}}\big)(s_t) \Big],$$
$$r_t^g(\theta) = \pi_\theta(a_t \mid s_t) \,/\, \pi_{\theta_{\text{old}}}(a_t \mid s_t). \tag{1}$$

4 Our Method: ResizeCO

To allow diffusion-based combinatorial optimization to generalize to large instances, we need to address two challenges: (1) *Size generalization* – handling instances that come from a distribution of large instances; and (2) *computational scalability* – handling larger search spaces. We take a three-pronged approach to handle these. (A) An architecture that supports a scalable representation that captures global structures, while emphasizing local structure that stays meaningful as instances grow (Section 4.1), (B) training for *size generalization without ground-truth labels*, learning from route length alone at the target size (Section 4.2), and (C) *fast parallel inference* whose quality scales with the test-time budget (Section 4.3). Figure 1 draws the three as one pipeline, with our contributions marked in yellow.

4.1 The Graph Transformer Encoder: Local Structure in Every Layer

Current approaches (Chen et al., 2026) encode the routing graphs using transformers, since they capture global structure and are fast. However, transformers can be less sensitive to local structure. This can hurt size generalization significantly, because the distribution of local structures may be preserved when moving to larger instances (Fang et al., 2024). We therefore design a variant of *Graph Transformer* for the current optimization problem.

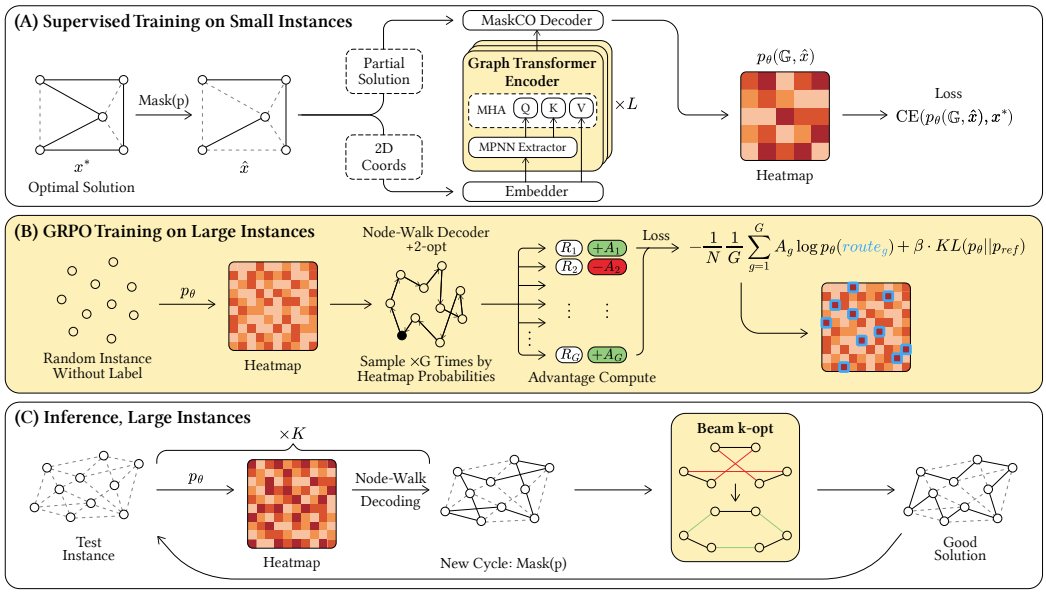


Figure 1: The three stages of ResizeCO. Yellow marks our novel contributions. (A) **Supervised training.** A masked-diffusion model is trained using small labeled instances with a Graph Transformer encoder. (B) **RL training.** Label-free GRPO training on larger instances. A node-walk decoder samples G routes from the heatmap. Each route is denoted route_g, and their lengths alone determine the advantage. (C) **Inference on large instances.** Multiple correction cycles are executed. Each cycle runs K unmasking steps to build an edge heatmap and decodes the heatmap into a route with the node-walk decoder. The route is then "polished" with our GPU beam k-opt search. Before the next cycle, the route is partly re-masked.

Specifically, we construct, for every instance, a k-nearest-neighbor graph $\mathbb{G}_k$ from its node coordinates, and assign weights w_{ij} to the edges of that graph, based on the distances between the nodes, $w_{ij} = 1/(1 + \|v_i - v_j\|)$ for nodes i and j. This weighting assigns larger weight to closer nodes.

We then follow the Graph Transformer architecture of Chen et al. (2022). We build a two-layer *MPNN extractor* that receives as input the graph $\mathbb{G}_k$, where each layer has the form $h' = h + W_{\text{out}} \cdot \mathrm{RMSNorm}\big(\mathrm{SiLU}(W_{\text{self}} \cdot h + W_{\text{neigh}} \cdot \mathrm{mean}_{j \in \mathcal{N}_k(i)}\, w_{ij} \cdot h_j)\big)$. This is fully defined in Appendix B.1. We initialize W_{out} to zero before supervised training, so the encoder starts as a plain transformer, and train it in both the supervised and the RL stages. In each attention layer, we then compute the queries and the keys from the extractor output and, as in a standard transformer, the values from the plain node states. This makes the attention weights sensitive to the local structure around each node. All L encoder layers take this form, each running the extractor anew, and the decoder is untouched.

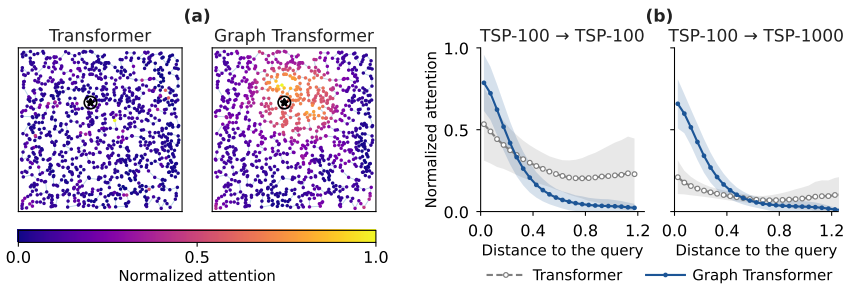


Figure 2: Encoder attention, averaged over all layers and normalized per row. (a) One query node (star) at TSP-1000. (b) Mean attention by distance to the query node. Shaded bands denote std over query nodes.

We analyze the effect of the architecture change on the distribution of encoder attention, comparing the transformer and the Graph Transformer, trained on TSP-100, and then tested on TSP-100 (in distribution) and on TSP-1000 (size generalization), see Figure 2. Both models attend more strongly to close neighbors when tested in distribution (Figure 2b left $100 \rightarrow 100$), but only the Graph Transformer keeps that preference in size generalization (Figure 2b right $100 \rightarrow 1000$). This demonstrates how the locality preference learned by the Graph Transformer survives the size change.

4.2 Label-Free GRPO at the Target Size

Our second main contribution addresses size generalization with no ground-truth labels. True labels are not available for large instances because finding their optimal solutions is costly. We therefore take a reinforcement learning approach that uses the cost of a route as the reward. We adapt GRPO to CO with three changes: (a) a reward that includes the 2-opt polish operator, (b) a node-walk decoder that defines an exact probability model over routes, and (c) a loss with a single update per batch and a KL penalty on the whole heatmap.

Algorithm 1 Label-free GRPO at the target size. Blue text marks our changes to standard GRPO.

Require: $\mathcal{D}$ unlabeled instances, θ_0 supervised model, G group size, K unmasking steps per route, η learning rate

$\theta \leftarrow \theta_0,\ p_{\text{ref}} \leftarrow p_{\theta_0}$ ▷ frozen
repeat
  draw an instance $\mathbb{G} \sim \mathcal{D}$ with N nodes
  for $g = 1 \dots G$ **do** ▷ distinct start nodes
    $\hat{x}_g \leftarrow \emptyset,\ n_1 \leftarrow$ the g-th start node, $U_1 \leftarrow V \setminus \{n_1\}$
    for $t = 1 \dots K$ **do**
      $p_\theta^t \leftarrow p_\theta(\mathbb{G}, \hat{x}_g)$ ▷ one forward pass
      for $j = \lfloor (t-1)N/K \rfloor + 1 \dots \lfloor tN/K \rfloor$ **do**
        draw $n_{j+1} \sim \pi_\theta(\cdot \mid s_j; p_\theta^t)$ ▷ Eq. (2)
        $\hat{x}_g \leftarrow \hat{x}_g \cup \{(n_j, n_{j+1})\}$ ▷ unmask the edge
        $U_{j+1} \leftarrow U_j \setminus \{n_{j+1}\}$
      end for
    end for
    $\text{route}_g \leftarrow \hat{x}_g$
    $R_g \leftarrow -\text{length}\big(\text{2-opt}(\text{route}_g)\big)$
  end for
  $A_g \leftarrow$ advantage of $R_1 \dots R_G$
  $\theta \leftarrow \theta - \eta \nabla \mathcal{L}_{\text{ResizeCO}}(\theta)$ ▷ one update, Eq. (4)
until the step budget runs out

First, let us formally define the RL problem, adapting GRPO for masked diffusion to our combinatorial optimization case. We view the construction of one route as a Markov decision process with one decision per edge, and each of the K unmasking steps supplies the heatmap for the next N/K decisions. At decision $j \in \{1, \dots, N\}$, **the state** $s_j = (\mathbb{G}, \hat{x}_j)$ contains the instance and the edges unmasked so far. The **action** $a_j \sim \pi_\theta(\cdot \mid s_j; p_\theta^t)$ selects which single edge of the route is unmasked next from the heatmap $p_\theta^t = p_\theta(\mathbb{G}, \hat{x}_t)$ of the most recent unmasking step t. Algorithm 1 presents the full training procedure, where the outer loop runs over the unmasking steps t and the inner loop over the decisions j.

(a) The polished cost as a reward. Unlike previous approaches in this area, we use as reward the negative route length *after* a 2-opt polish operator, $R_g = -\text{length}\big(\text{2-opt}(\text{route}_g)\big)$. This provides two benefits: it is more consistent with inference, and it removes errors accumulated during iterative node-walk decoding. Here, the advantage ranking corresponds better to solution quality.

(b) Node-walk decoder. This decoder constructs a route sequentially, node by node. A random node is set as the root. Then, at each step, another node is sampled using the heatmap p_θ^t, which represents a conditional distribution because a softmax was applied to each row. The probability of each decision is therefore

$$\pi_\theta(a_j \mid s_j; p_\theta^t) = \frac{p^t_{\theta,\, n_j n_{j+1}}}{\sum_{n' \in U_j} p^t_{\theta,\, n_j n'}}, \tag{2}$$

where n_{j+1} is the node at the end of the edge (n_j, n_{j+1}) selected at decision j, and U_j is the set of nodes not yet visited, except $U_N = \{n_1\}$, so the last decision closes the route with probability one. Multiplying the decisions within a step and over steps, we obtain the total probability of route_g

$$\log p_\theta(\text{route}_g) = \sum_{t=1}^{K} \sum_{j=\lfloor (t-1)N/K \rfloor + 1}^{\lfloor tN/K \rfloor} \log \pi_\theta(a_j \mid s_j; p_\theta^t). \tag{3}$$

In contrast with other masked generative models (Luo et al., 2025b; Zhou et al., 2025), which score all positions unmasked in the same step as independent, our approach scores masked edges sequentially, each edge being conditioned on its predecessor. As a result, Eq. (3) provides the exact Markovian probability of a route under the node-walk policy, rather than an approximation.

Node-walk decoding differs from the greedy decoding that is commonly used in the field. In greedy decoding, all N^2 edges of the graph are ranked by their heatmap score, and edges are selected one

by one as long as they are feasible. This decoding does not allow us to compute the exact route probability, because it does not compute a proper normalization. GRPO improves the model with node-walk decoding and not with greedy decoding (Appendix D.4).

(c) Our training procedure and loss. Our loss differs from the standard GRPO loss of Eq. (1) in two ways (see Appendix E). First, since rollouts are fast to compute, we take a single update per batch on fresh rollouts. Second, the KL penalty is the row average over the whole edge heatmap rather than over the visited states only. This yields our final loss

$$\mathcal{L}_{\mathrm{ResizeCO}}(\theta) = -\frac{1}{N}\frac{1}{G}\sum_{g=1}^{G} A_g \log p_\theta(\mathrm{route}_g) + \beta\, D_{\mathrm{KL}}\big(p_\theta \,\|\, p_{\mathrm{ref}}\big). \tag{4}$$

4.3 Accelerating Inference Using a GPU Beam k-opt Search

We now describe the third technical contribution, which accelerates inference using a GPU beam k-opt search. At inference, a decoded route usually still contains local mistakes that a few edge exchanges can fix, so every route ends in a local search. MaskCO uses 2-opt for this search, and we replace it with the stronger k-opt.

The k-opt procedure is a highly useful sequential procedure. Simply put, in each move k edges are removed, and one aims to add back another set of k edges that improves the cost. For example, 2-opt replaces a pair of crossing edges with non-crossing edges. While k-opt is highly useful, finding sets of k edges that improve the cost can be computationally expensive.

Following LKH (Helsgaun, 2000), we simplify and view the k-opt process as a series of decisions that form a tree. The root of the tree is the current solution (for TSP, a Hamiltonian cycle). It has $\Theta(N^2)$ outgoing tree branches. Moving from the root to one of these branches means that we remove an edge (n_i, n_j) in the solution (one among N edges of the solution), and add another edge, outgoing from n_i or n_j (say, n_i), to another node (the "front node"). This breaks the validity of the solution. Every deeper level removes one of the two solution edges of the last "front node", and adds another edge, from the other node of that removed edge to another node, which becomes the new "front node", or back to n_j. Looping back to n_j closes the route, which is feasible again if it is a single cycle. k-opt means that we look for solutions that are feasible at depth at most k. LKH searches the tree in a sequential way, which can be thought of intuitively as a depth-first approach, and continues as long as it finds a better route.

This sequential search fits a GPU poorly, because each decision depends on the one before it and the work differs from route to route. **We instead search over the tree using a parallel beam search.** At each level, we keep the top M branches active, advance the set of M branches deeper until depth k, and also close every branch to record the best feasible solutions. All branches of all the routes in a batch perform the same computation at the same moment, so they are computed in parallel on a GPU. At the end, we apply together all the recorded moves that do not overlap on the route, whereas LKH applies one move at a time. We provide the details in Appendix B.5.

5 Experiments

We evaluate ResizeCO on TSP and CVRP instances of various sizes, against baselines from four families. Every experiment tests size generalization. We take models trained at one instance size and test them on larger ones.

Datasets For synthetic data, we follow the standard data generation protocol of Kool et al. (2019), for both TSP and CVRP. Each instance is a set of N nodes sampled uniformly in the unit square. CVRP demands are integers drawn uniformly from 1 to 9, with a vehicle capacity of 50 (Chen et al., 2026). The training and test sets cover sizes 100, 500, 1000 and 10000, and only the supervised stage uses the solutions that come with the training sets. For real-world instances, we test on the benchmark libraries TSPLIB (Reinelt, 1991) and CVRPLIB (Uchoa et al., 2017; Arnold et al., 2019), in three groups per library matched to the test sizes 500, 1000 and 10000. Every set is detailed in Appendix A.

Metrics Following prior work (Sun & Yang, 2023; Li et al., 2025; Chen et al., 2026), we report three metrics per test size. (1) **Objective**: the average total route cost over the test set (a single route for

TSP, the sum over all sub-routes for CVRP). (2) **Gap**: the average relative excess in percent over a near-optimal reference solution, $(\text{cost} - \text{ref})/\text{ref} \times 100$. (3) **Time**: the total wall-clock time to solve all test instances (hardware details in Appendix B.6).

Baseline Methods We compare ResizeCO with four families of solvers. (1) Classical solvers: an exact solver and the leading heuristic solvers. (2) Supervised learning (SL): autoregressive solvers trained on labeled instances. (3) Label-free learning: solvers trained with RL or self-improvement, without labels. For fair evaluation, they were run with their released models trained nearest to the target size. (4) Diffusion-based solvers, among them MaskCO (Chen et al., 2026). MaskCO Self-Train in Figures 3 and 4 is its self-training variant (Appendix C). For each baseline, we use the released model and the hyperparameters specified in its original paper, selecting the fastest reported configuration and retaining all refinements and post-processing. Each table row uses the configuration name from the corresponding paper.

Evaluation Setting and Hyperparameters In the experiments below, the quality of solutions depends on the computational budget T, and this in turn depends on three hyperparameters K, C, S. K is the number of edge-unmasking steps. The unmasking process is then repeated for C correction cycles, and each correction cycle runs with S parallel samples. Together they yield the total forward-pass budget $T = K \cdot C \cdot S$. For ResizeCO, we use $C = 1$ and $K = 1$ and vary S as the compute budget knob. For MaskCO, we use the optimal values reported by the authors, $S = 8$, with $K = 1, 2$ on TSP and $K = 16$ on CVRP, and vary the values of C as the compute budget knob. For all other methods, we use the configurations that the authors provide for the test size.

5.1 CVRP Results

Table 1 shows three size-generalization experiments, each training on a source size and testing on a target size, $100 \to 500$, $500 \to 1000$ and $1000 \to 10000$. In this table, we focus on short inference time, and therefore use configurations reported by competing methods that yielded their fastest results. Among our diffusion-based baselines, only MaskCO supports CVRP, and among the classical solvers we report HGS alone, because LKH-3 stays above 7% at targets 500 and 1000. Figure 3 depicts the full Pareto plane of gap against inference run time, focusing on leading approaches.

Table 1: CVRP results. Dashes indicate a setting we could not evaluate. An asterisk (*) indicates a number quoted from its paper at a different capacity convention, not comparable to the other rows. S counts parallel samples and T total model forward passes. Bold font marks the best model at a comparable run time.

	CVRP-100 → CVRP-500			**CVRP-500 → CVRP-1000**			**CVRP-1000 → CVRP-10000**		
Method	**Obj**	**Gap**	**Time**	**Obj**	**Gap**	**Time**	**Obj**	**Gap**	**Time**
Classical solvers									
HGS (reference) (Vidal, 2022)	62.15	0.000%	55 h	121.07	0.000%	197 h	1120.23	0.000%	512 h
HGS (Vidal, 2022)	63.99	2.96%	52 s	124.02	2.44%	2.4 m	1129.62	0.846%	2.2 h
Supervised learning									
BQ-NCO (G) (Drakulic et al., 2023)	64.39	3.67%	26 s	-	-	-	-	-	-
LEHD (greedy) (Luo et al., 2023)	64.40	3.63%	18 s	-	-	-	-	-	-
Label-free learning									
INViT-3V (Fang et al., 2024)	67.69	9.03%	6.1 m	131.78	8.97%	10.8 m	1192.43	6.49%	2.1 h
GLOP-G (LKH-3) (Ye et al., 2024)	-	-	-	-	8.83%*	1.4 m*	-	8.27%*	23.9 m*
SIT (PRC10) (Luo et al., 2025a)	-	-	-	-	4.52%*	45 s*	-	4.43%*	14.5 m*
ReLD (augx8) (Huang et al., 2025)	64.16	3.29%	21 s	125.06	3.38%	40 s		OOM	
Diffusion-based									
MaskCO (T=128) (Chen et al., 2026)	65.79	5.90%	11 s	125.79	3.98%	11 s	1197.55	7.97%	1 h
MaskCO (T=1280) (Chen et al., 2026)	65.39	5.25%	1.6 m	125.79	3.98%	1.6 m	1146.56	2.36%	10.6 h
Ours									
ResizeCO (S=1)	**64.10**	**3.18%**	**14 s**	**124.35**	**2.76%**	**13 s**	**1118.83**	**−0.096%**	**49.9 m**

ResizeCO reaches the lowest gap of every learned solver in all three experiments, at a similar or lower run time. Its margin over the strongest learned baseline grows with the target size, and from CVRP-1000 to CVRP-10000 every other learned solver stays far from the reference solutions while ResizeCO reaches them, ending below the reference cost as the sample budget grows. Against the classical solver, ResizeCO leads at short run times in every experiment (Figure 3).

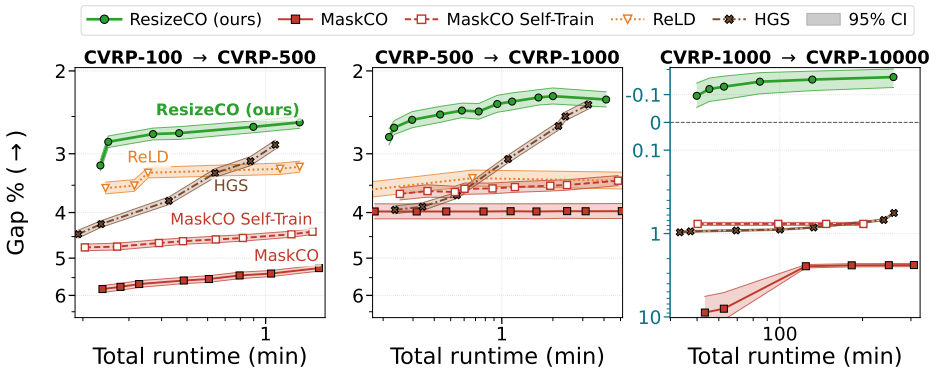


Figure 3: CVRP solution quality (gap) against total run time for three source-to-target pairs. Each line varies the compute budget of one method. Our approach ResizeCO in green achieves superior quality in short durations. **From CVRP-1000 to CVRP-10000, ResizeCO's routes are shorter than the HGS reference solutions (negative gap).**

Real-World Instances We also test on real-world instances from CVRPLIB, whose geometry differs from the synthetic instances. We train on synthetic instances only, so this test changes the geometry as well as the size. **Averaged over the instances of each group, ResizeCO reaches the lowest gap among the learned solvers in all three groups of Table 2, at a comparable run time.**

Table 2: Mean gap and run time on real-world instances from CVRPLIB. Times are per instance. Bold font marks Pareto-domination (best gap and time) in each column group.

CVRPLIB	100 → [321,684]		100 → [748,956]		1000 → [7000,15000]	
	Gap	Time	Gap	Time	Gap	Time
ResizeCO (ours, S=1)	**3.67%**	**0.4 s**	**3.80%**	**0.7 s**	**8.64%**	**30.6 s**
MaskCO (T=128)	6.17%	1.1 s	5.60%	1.1 s	18.37%	39 s
ReLD (augx8)	3.76%	0.5 s	4.26%	0.9 s	OOM	

5.2 TSP Results

Table 3 and Figure 4 repeat the three experiments and the short-inference focus of Section 5.1 on TSP. **At a comparable run time, ResizeCO reaches the lowest gap of every learned solver in all three experiments.** From TSP-1000 to TSP-10000, every other learned solver reaches a higher gap, most of them at a far longer run time.

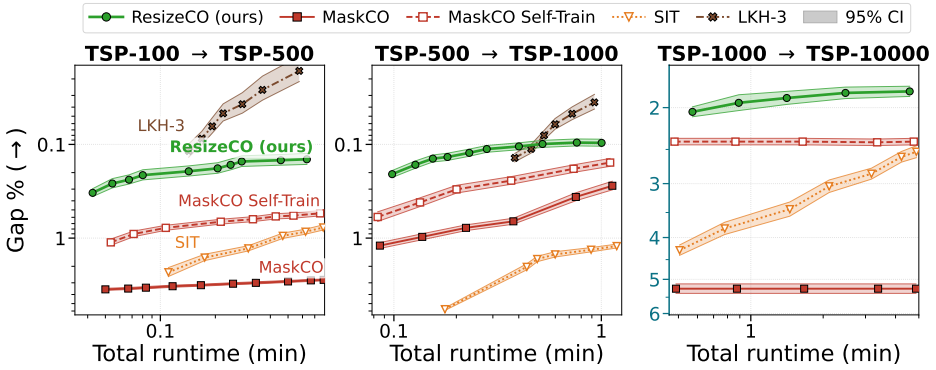


Figure 4: TSP solution quality (gap) against total run time for three source-to-target pairs. Each line varies the compute budget of one method. Our approach ResizeCO in green achieves superior quality in short durations. We limit runs on TSP-10000 to less than 5 minutes, which LKH-3 exceeds. Error bars correspond to 95% confidence intervals.

Against the classical solver, ResizeCO runs faster than LKH-3, and from TSP-500 to TSP-1000 it reaches a lower gap than the fastest LKH-3 setting at a shorter run time (Figure 4).

Table 3: TSP size-generalization results. Bold font marks the best model at a comparable run time. "Time Limit" denotes cases that did not finish within 4 hours. A dash indicates a setting we did not evaluate. C counts correction cycles and S parallel samples. Inference budgets (T, T_s, T_g) follow each paper's own notation. Larger sample counts lie on the curves of Figure 4.

	TSP-100 → TSP-500			TSP-500 → TSP-1000			TSP-1000 → TSP-10000		
Method	**Obj**	**Gap**	**Time**	**Obj**	**Gap**	**Time**	**Obj**	**Gap**	**Time**
Classical solvers									
Concorde / LKH-3 (reference)	16.55	0.000%	37.66 m	23.12	0.000%	6.65 h	71.83	0.000%	8.8 h
LKH-3 (trials=32) (Helsgaun, 2017)	16.55	0.026%	22 s	23.13	0.036%	56 s	71.81	−0.023%	7.7 m
LKH-3 (trials=1) (Helsgaun, 2017)	16.57	0.118%	8.6 s	23.15	0.139%	23 s	71.89	0.085%	5.6 m
Supervised learning									
BQ-NCO (G) (Drakulic et al., 2023)	16.74	1.19%	23 s	-	-	-	-	-	-
LEHD (greedy) (Luo et al., 2023)	16.80	1.53%	15 s	-	-	-	-	-	-
Label-free learning									
INViT-3V (Fang et al., 2024)	17.27	4.38%	4.9 m	24.30	5.10%	9.8 m	76.57	6.61%	33.2 m
GLOP (Ye et al., 2024)	17.07	3.20%	19 s	24.01	3.85%	38 s	75.61	5.27%	29 s
SIT (PRC10) (Luo et al., 2025a)	16.70	0.947%	28 s	23.39	1.18%	1.3 m	74.01	3.04%	2.1 m
Diffusion-based									
DIFUSCO (T_s=50, G) (Sun & Yang, 2023)	17.09	3.27%	2.2 m	23.64	2.27%	9.6 m	74.11	3.18%	29.6 m
DIFUSCO (T_s=100, S=4) (Sun & Yang, 2023)	16.97	2.53%	21.6 m	23.48	1.56%	1.2 h		Time Limit	
Fast T2T (T_s=5, T_g=5, G) (Li et al., 2024)	17.30	4.58%	2.5 m	23.46	1.48%	8.9 m	75.20	4.70%	3.6 h
Fast T2T (T_s=5, T_g=5, S=4) (Li et al., 2024)	17.16	3.71%	11.2 m	23.37	1.10%	20.1 m		OOM	
GenSCO (C=1, S=8) (Li et al., 2025)	17.19	3.88%	2.5 s	23.17	0.245%	20 s	89.77	24.97%	3.4 m
GenSCO (C=10, S=8) (Li et al., 2025)	17.05	3.06%	25 s	23.15	0.138%	3.3 m	89.34	24.38%	34 m
MaskCO (T=8/16) (Chen et al., 2026)	17.21	3.99%	1 s	23.27	0.662%	23 s	75.60	5.26%	3.4 m
MaskCO (T=80/160) (Chen et al., 2026)	17.07	3.18%	10 s	23.15	0.138%	3.8 m	75.24	4.75%	33.9 m
Ours									
ResizeCO (S=1)	**16.60**	**0.329%**	**2.6 s**	**23.17**	**0.208%**	**5.9 s**	**73.30**	**2.05%**	**34 s**

Real-World Instances We also test on real-world instances from TSPLIB. **Averaged over the instances of each group, ResizeCO reaches the lowest gap among the learned solvers in the first two groups of Table 4.** In the last group, DIFUSCO reaches a lower gap at a 13× longer run time.

Table 4: Mean gap and run time on real-world instances from TSPLIB. Times are per instance. Bold font marks Pareto-domination (best gap and time) in each column group.

TSPLIB	**100 → [400,657]**		**100 → [724,1432]**		**1000 → [11849,14051]**	
	Gap	**Time**	**Gap**	**Time**	**Gap**	**Time**
ResizeCO (ours, S=4)	**1.26%**	**0.7 s**	**1.46%**	**1.2 s**	5.40%	20.7 s
MaskCO (T=32/32/16)	3.17%	2.1 s	7.48%	1.4 s	7.23%	14 s
DIFUSCO (T_s=50, G)	4.62%	0.9 s	5.03%	5.3 s	5.14%	4.4 m
SIT (PRC10)	2.24%	10.3 s	2.57%	20.4 s	5.89%	27.6 s

5.3 Ablation Studies

Finally, we evaluate the contribution of various components of ResizeCO. Table 5 shows that removing any component reduces performance (larger gap). See more experiments in Table D.5.

Table 5: Component ablation of ResizeCO in size generalization. Each row removes a single component.

	TSP-100 → TSP-500		TSP-500 → TSP-1000	
Method	**Gap**	**Time**	**Gap**	**Time**
ResizeCO (full)	**0.329%**	**2.6 s**	**0.208%**	**5.9 s**
w/o label-free GRPO	2.71%	5.6 s	0.774%	9.1 s
w/o small-model initialization	2.29%	3.6 s	2.75%	10.0 s
w/o Graph Transformer encoder	0.521%	1.6 s	0.269%	3.2 s
w/o GPU beam k-opt search	0.456%	2.5 s	0.282%	6.4 s

6 Conclusion

We described ResizeCO, a diffusion-based approach for routing optimization that generalizes a model trained on small instances to large instances, using RL on large instances without labels. It combines a Graph Transformer encoder, label-free fine-tuning with GRPO and a GPU beam k-opt search, and our ablation shows that removing any one of these components increases the gap. In size-generalization experiments on TSP and CVRP instances with up to 10000 nodes, ResizeCO reaches the lowest gap of every learned solver at a comparable run time, and on CVRP its margin over the strongest learned baseline grows with the target size. From CVRP-1000 to CVRP-10000, its routes are shorter than the HGS reference solutions. ResizeCO also reaches the lowest gap among the learned solvers in five of the six groups of real-world instances from TSPLIB and CVRPLIB, although we train on synthetic instances only. Together, these results set a new SoTA at short run times, which may open new applications like on-the-fly interactive routing.

AI USE STATEMENT

We defined the problem and its scope, conceived the technical ideas of the solution and wrote the paper. We used generative AI tools only as an aid, and we checked every detail of their output. Specifically, we used them to help develop the conceptual framework, design experiments, implement methods, clean and reformat datasets and interpret results, but not to propose or refine hypotheses, formulate or prove mathematical claims, generate synthetic datasets, translate text or analyze qualitative data. We also used them to write and run code, make figures and artifacts, draft, edit and review the paper text, find and summarize literature, format references and brainstorm. We reviewed all AI-assisted work, tested the code, and checked every number against the raw evaluation outputs and every citation against its source. We take full responsibility for the final content, including anything produced with generative AI tools.

REPRODUCIBILITY STATEMENT

We have made every effort to ensure that our results are reproducible. Appendix A describes the synthetic and real-world datasets, and Appendix B gives the full implementation details, including the architecture, the supervised and label-free GRPO training, the inference settings, the GPU beam k-opt search and the hardware. Appendix C gives the configurations of the baselines. We attach the complete source code as supplementary material.

REFERENCES

David L. Applegate, Robert E. Bixby, Vašek Chvátal, and William J. Cook. *The Traveling Salesman Problem: A Computational Study*. Princeton University Press, 2006.

Florian Arnold, Michel Gendreau, and Kenneth Sörensen. Efficiently solving very large-scale routing problems. *Computers & Operations Research*, 107:32–42, 2019. ISSN 0305-0548.

Dexiong Chen, Leslie O'Bray, and Karsten Borgwardt. Structure-aware transformer for graph representation learning. In *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pp. 3469–3489. PMLR, 17–23 Jul 2022.

Lvda Chen, Yang Li, and Junchi Yan. MaskCO: Masked generation drives effective representation learning and exploiting for combinatorial optimization. In *The Fourteenth International Conference on Learning Representations*, 2026.

G. A. Croes. A method for solving traveling-salesman problems. *Operations Research*, 6(6):791–812, 1958.

Darko Drakulic, Sofia Michel, Florian Mai, Arnaud Sors, and Jean-Marc Andreoli. BQ-NCO: Bisimulation quotienting for efficient neural combinatorial optimization. In *Advances in Neural Information Processing Systems*, volume 36, pp. 77416–77429, 2023.

Han Fang, Zhihao Song, Paul Weng, and Yutong Ban. INViT: A generalizable routing problem solver with invariant nested view transformer. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 12973–12992. PMLR, 21–27 Jul 2024.

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. In *Proceedings of the Thirty-Third International Joint Conference on Artificial Intelligence, IJCAI-24*, pp. 6914–6922. International Joint Conferences on Artificial Intelligence Organization, 2024. Main Track.

Keld Helsgaun. An effective implementation of the Lin-Kernighan traveling salesman heuristic. *European Journal of Operational Research*, 126(1):106–130, 2000. ISSN 0377-2217.

Keld Helsgaun. *An Extension of the Lin-Kernighan-Helsgaun TSP Solver for Constrained Traveling Salesman and Vehicle Routing Problems: Technical report*. Roskilde Universitet, 2017.

André Hottung, Yeong-Dae Kwon, and Kevin Tierney. Efficient active search for combinatorial optimization problems. In *International Conference on Learning Representations*, 2022.

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.

Ziwei Huang, Jianan Zhou, Zhiguang Cao, and Yixin Xu. Rethinking light decoder-based solvers for vehicle routing problems. In *The Thirteenth International Conference on Learning Representations*, 2025.

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.

Chaitanya K. Joshi, Quentin Cappart, Louis-Martin Rousseau, and Thomas Laurent. Learning the travelling salesperson problem requires rethinking generalization. *Constraints*, 27(1):70–98, Apr 2022.

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

Yeong-Dae Kwon, Jinho Choo, Byoungjip Kim, Iljoo Yoon, Youngjune Gwon, and Seungjai Min. POMO: Policy optimization with multiple optima for reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 33, pp. 21188–21198, 2020.

Sirui Li, Zhongxia Yan, and Cathy Wu. Learning to delegate for large-scale vehicle routing. In *Advances in Neural Information Processing Systems*, volume 34, pp. 26198–26211, 2021.

Yang Li, Jinpei Guo, Runzhong Wang, and Junchi Yan. T2T: From distribution learning in training to gradient search in testing for combinatorial optimization. In *Advances in Neural Information Processing Systems*, volume 36, pp. 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. In *Advances in Neural Information Processing Systems*, volume 37, pp. 30179–30206, 2024.

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 *Advances in Neural Information Processing Systems*, volume 38, Main Conference, pp. 127168–127196, 2025.

S. Lin and B. W. Kernighan. An effective heuristic algorithm for the traveling-salesman problem. *Operations Research*, 21(2):498–516, 1973.

Jie Liu, Gongye Liu, Jiajun Liang, Yangguang Li, Jiaheng Liu, Xintao Wang, Pengfei Wan, Di Zhang, and Wanli Ouyang. Flow-GRPO: Training flow matching models via online RL. In *Advances in Neural Information Processing Systems*, volume 38, Main Conference, pp. 40783–40818, 2025.

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

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

Yifu Luo, Xinhao Hu, Keyu Fan, Haoyuan Sun, Zeyu Chen, Bo Xia, Tiantian Zhang, Yongzhe Chang, and Xueqian Wang. Reinforcement learning meets masked generative models: Mask-GRPO for text-to-image generation. In *Advances in Neural Information Processing Systems*, volume 38, Main Conference, pp. 108460–108485, 2025b.

Yimeng Min, Yiwei Bai, and Carla Gomes. Unsupervised learning for solving the travelling salesman problem. In *Advances in Neural Information Processing Systems*, volume 36, pp. 47264–47278, 2023.

Itay Niv and Neta Rabin. Exploring graph-transformer out-of-distribution generalization abilities. *arXiv preprint arXiv:2506.20575*, 2025.

Wen-Bao Qiao, Jean-Charles Créput, Nan Wang, Kun Meng, and Abdelkhalek Mansouri. Breaking TSP local search barriers: scalable multi-GPU parallelisation of 4-opt, 5-opt, 6-opt and hybrid variable λ-opt. *Expert Systems with Applications*, 296:129110, 2026. ISSN 0957-4174.

Ruizhong Qiu, Zhiqing Sun, and Yiming Yang. DIMES: A differentiable meta solver for combinatorial optimization problems. In *Advances in Neural Information Processing Systems*, volume 35, pp. 25531–25546, 2022.

Gerhard Reinelt. TSPLIB - A traveling salesman problem library. *INFORMS J. Comput.*, 3:376–384, 1991.

Kamil Rocki and Reiji Suda. Accelerating 2-opt and 3-opt local search using GPU in the travelling salesman problem. In *2012 International Conference on High Performance Computing & Simulation (HPCS)*, pp. 489–495, 2012.

Carlos S. Sepúlveda and Gonzalo A. Ruz. Baseline-free policy optimization for neural combinatorial optimization. *arXiv preprint arXiv:2606.10321*, 2026.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. DeepSeekMath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.

Hyungseok Song, Deunsol Yoon, Kanghoon Lee, Han-Seul Jeong, Soonyoung Lee, and Woohyung Lim. CADO: From imitation to cost minimization for heatmap-based solvers in combinatorial optimization. *Transactions on Machine Learning Research*, 2026. ISSN 2835-8856.

Zhiqing Sun and Yiming Yang. DIFUSCO: Graph-based diffusion solvers for combinatorial optimization. In *Advances in Neural Information Processing Systems*, volume 36, pp. 3706–3731, 2023.

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. ISSN 0377-2217.

Thibaut Vidal. Hybrid genetic search for the CVRP: Open-source implementation and SWAP* neighborhood. *Computers & Operations Research*, 140:105643, 2022.

Qitian Wu, Chenxiao Yang, Kaipeng Zeng, and Michael M. Bronstein. Supercharging graph transformers with advective diffusion. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 67959–67985. PMLR, 13–19 Jul 2025.

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, pp. 20284–20292, 2024.

Jianan Zhou, Yaoxin Wu, Wen Song, Zhiguang Cao, and Jie Zhang. Towards omni-generalizable neural methods for vehicle routing problems. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pp. 42769–42789. PMLR, 23–29 Jul 2023.

Renping Zhou, Zanlin Ni, Tianyi Chen, Zeyu Liu, Yang Yue, Yulin Wang, Yuxuan Wang, Jingshu Liu, and Gao Huang. Co-GRPO: Co-optimized group relative policy optimization for masked diffusion model. *arXiv preprint arXiv:2512.22288*, 2025.

APPENDIX

This appendix details the datasets (Appendix A), the implementation of ResizeCO (Appendix B), the configurations of the correction-cycle baselines (Appendix C), additional results and ablations (Appendix D), and the derivation of the training loss (Appendix E).

A DATASETS

Synthetic Instances The labeled training sets cover sizes 100, 500 and 1000 for both problems, and the supervised stage uses them with their solutions. The RL stage reuses the same instances without their solutions, and adds instance sets that we generate at size 10000. The labeled training sets and the test sets are the ones released with prior work (Sun & Yang, 2023; Li et al., 2024; Chen et al., 2026), except the CVRP-10000 test set, which we generate from the same distribution. The test sets cover sizes 100, 500, 1000 and 10000, and every test instance comes with a reference solution, against which we measure the gap. For TSP, the test references are optimal routes from Concorde (Applegate et al., 2006), except at TSP-10000, where exact solving is out of reach and the test references come from long LKH-3 runs (Helsgaun, 2017). For CVRP, the test references come from the strong heuristic solver HGS (Vidal, 2022), with four-hour runs at CVRP-10000, so a CVRP gap is not a distance from the optimum. Table A.1 lists the set sizes and the solver behind every test reference and every training label.

Table A.1: Test and training sets. A dash marks a training set without labels.

		Test		Training	
Problem	**Size**	**Instances**	**Reference**	**Instances**	**Labels**
TSP	100	1280	Concorde	1.26 M	LKH-3
TSP	500	128	Concorde	128 k	LKH-3
TSP	1000	128	Concorde	64 k	LKH-3
TSP	10000	16	LKH-3	6.4 k	-
CVRP	100	1280	HGS	1.54 M	HGS
CVRP	500	128	HGS	200 k	HGS
CVRP	1000	64	HGS	100 k	HGS
CVRP	10000	128	HGS	20 k	-

Real-World Instances The benchmark libraries TSPLIB (Reinelt, 1991) and CVRPLIB (Uchoa et al., 2017; Arnold et al., 2019) provide real-world instances, and we test on them at the end of Sections 5.1 and 5.2. Each library contributes three groups matched to the test sizes 500, 1000 and 10000, and a group holds every instance within a factor of 0.7 to 1.5 of its size, 24 TSP and 22 CVRP instances in total. The two smaller CVRPLIB groups also stay within 7 to 14 customers per sub-route, near the 10 of our training convention, and the CVRPLIB 10000 group takes its whole band, with 21 to 100 customers per sub-route. The solvers keep their main-table models, and the budget of each row is given in Tables 2 and 4. The two smaller groups of each library run the models trained at size 100, and the 10000 groups run the source-1000 models, without further training, as in the main-table experiment from size 1000 to size 10000.

B IMPLEMENTATION DETAILS

B.1 ARCHITECTURE

The structure extractor of ResizeCO is a small graph neural network that sits inside every encoder layer of the denoiser. It is a two-layer message-passing neural network (MPNN), where each node updates its state from the states of the neighboring nodes. It runs on the k-nearest-neighbor graph $\mathbb{G}_k$, whose edge between nodes i and j carries the weight $w_{ij} = 1/(1 + \|v_i - v_j\|)$. One extractor layer updates the node states h as

$$h' = h + W_{\text{out}} \cdot \text{RMSNorm}\big(\text{SiLU}(W_{\text{self}} \cdot h + W_{\text{neigh}} \cdot \text{mean}_{j \in \mathcal{N}_k(i)}\, w_{ij} \cdot h_j)\big), \tag{5}$$

with W_{out} initialized to zero before supervised training, so the extractor starts as the identity and the encoder starts as a plain transformer. At the first encoder layer, h is a learned linear embedding of the input features of each node (Table B.1), and at every later layer it is the output of the layer before. In Eq. (5), W_{self}, W_{neigh} and W_{out} are learned weight matrices and $\mathcal{N}_k(i)$ is the set of the k nearest neighbors of node i. SiLU is the activation function $\mathrm{SiLU}(x) = x \cdot \sigma(x)$, where σ is the logistic sigmoid. RMSNorm divides a vector by the root mean square of all entries and multiplies each entry by a learned gain. An encoder layer runs the extractor of Eq. (5) on the node states h and computes

$$\begin{aligned} s &= \mathrm{LN}\big(\mathrm{MPNN}(h, \mathbb{G}_k)\big), \qquad Q = W_Q \cdot s, \quad K = W_K \cdot s, \quad V = W_V \cdot \mathrm{LN}(h), \\ h' &= h + W_O \cdot \mathrm{softmax}\left(\frac{QK^\top}{\sqrt{d}}\right) \cdot V, \qquad h'' = h' + \mathrm{FFN}\big(\mathrm{LN}(h')\big). \end{aligned} \tag{6}$$

All L encoder layers take this form, each running the extractor anew, and the decoder is untouched. The neighborhood size k is the only architectural setting that changes with the test size. Table B.1 lists the parameters.

Table B.1: Architecture parameters.

Parameter	**TSP**	**CVRP**
Input features	2-D coordinates	2-D coordinates + demand
Encoder layers	16	16
Decoder layers	6	6
Embedding dimension	256	512
Attention heads	8	8
Structure-extractor GNN layers	2	2
Neighborhood size k	$\max(20,\ N/10)$	$\max(20,\ N/10)$

B.2 Supervised Training

The supervised stage trains the denoiser to complete partial solutions on the labeled training sets of Table A.1, with the configuration in Table B.2. A warm start from the previous size means that the TSP-500 model starts from the trained TSP-100 model, and the TSP-1000 model from the trained TSP-500 model.

Table B.2: Supervised training parameters. Values in parentheses are training sizes.

Parameter	**TSP**	**CVRP**
Batch size	256 (100), 128 (500), 16 (1000)	512 (100), 64 (500), 16 (1000)
Training steps	500000	225000 (100), 150000 (500, 1000)
Warm start	from scratch (100), previous (500, 1000)	from scratch

B.3 Label-Free GRPO Training

Table B.3 lists the configuration of the RL stage. The batch size and the group size G are the largest that fit in GPU memory at each size. The 2-opt pass inside the reward needs enough steps to remove the sampling error of the node-walk decoder (Section 4.2). It must also stop before the routes of a group converge to the same local optimum, because the advantage then loses its signal. Because a 2-opt step costs more on larger instances, the CVRP runs to size 10000 use fewer steps, and the TSP runs to size 10000 keep the step count but restrict the 2-opt pass to the 20 nearest neighbors of each node.

B.4 Inference

Decoding the Heatmap A decoder turns the edge heatmap into a route. ResizeCO uses the *node-walk decoder*, which grows the route one node at a time by sampling the next node from the current node's row of p_θ, restricted to the unvisited nodes. On CVRP, it also respects capacity, skipping customers whose demand exceeds the load left in the vehicle and returning to the depot to open a new sub-route.

Table B.3: RL fine-tuning parameters. Values in parentheses are the target size or the source and target sizes of the run.

Parameter	TSP	CVRP
Group size G	8, 16 (500 to 1000), 64 (to 10000)	8, 64 (to 10000)
Unmasking steps K	1	1
Sampling temperature	1.0	1.0
KL coefficient β	0.005, 0 (to 10000)	0.005, 0 (to 10000)
Batch size	64 (500), 24 (1000), 96 (500 to 1000), 4 (to 10000)	16 (500), 4 (1000), 1 (to 10000)
Training steps	200000, 150000 (500 to 1000), 30000 (to 10000)	200000, 30000 (to 10000)
Reward 2-opt steps	100	100, 10 (to 10000)
Reward 2-opt neighbors	all, 20 (to 10000)	all
Reward	route length after 2-opt	route cost $+ \rho \cdot$ overload after 2-opt
Overload penalty ρ	-	3.0

MaskCO, like most heatmap solvers, uses the *greedy decoder*, which adds edges in decreasing heatmap score, skipping any edge that would give a node a third edge or close the loop early. On CVRP, the greedy decoder applies the two-edge rule to customers only, because the depot takes two edges per sub-route. Either route can then be shortened by *2-opt* local search (Croes, 1958), which deletes two edges and reconnects the two paths the other way around when that is shorter.

Inference Hyperparameters Table B.4 lists the inference settings of every ResizeCO row. The decoding configuration is set by the test size, not the source size. At inference, ResizeCO launches S runs in parallel, each building one route with the node-walk decoder from its own start node, and returns the shortest of the S routes. Each of the S parallel runs makes one forward pass of the denoiser, that is, $K = 1$ unmasking step and $C = 1$ correction cycle. The local search that follows decoding, the GPU beam k-opt search, is described with its settings in Appendix B.5.

Table B.4: ResizeCO inference parameters. Values in parentheses are test sizes.

Parameter	TSP	CVRP
Decoder	node-walk	node-walk
Unmasking steps K	1	1
Correction cycles C	1	1
Batch size	16 (500, 1000), 1 (10000)	16 (500), 8 (1000), 1 (10000)

B.5 The GPU Beam k-opt Search

The GPU beam k-opt search is the local search that follows decoding in ResizeCO (Section 4.3). A k-opt move removes k edges of a route and adds k new edges that give a shorter route. For example, 2-opt, the case $k = 2$, replaces a pair of crossing edges with a non-crossing pair. A move with a larger k is stronger, but the number of candidate moves is $\Theta(N^k)$, so a search must find promising moves rather than enumerate all of them.

The *k*-opt Search Tree of LKH LKH (Helsgaun, 2000), which implements the Lin-Kernighan search (Lin & Kernighan, 1973), finds such moves by building them in steps. As in Section 4.3, we view this process as a tree whose root is the current route. A step moves one level down the tree. It removes one route edge, which exposes one node of that edge, and adds a new edge from the exposed node to another node, which becomes the "front node". The first step can remove any route edge (n_i, n_j) and expose either node, say n_i, and we call the other node, n_j, the start node. Every later step removes one of the two route edges of the "front node" and exposes the other node of that edge. A branch of depth ℓ is a sequence of ℓ steps from the root, and its gain is $g_\ell = \sum_{s \le \ell}(d_s^- - d_s^+)$, where d_s^- and d_s^+ are the lengths of the edges removed and added at step s.

A branch alone does not give a valid route, because its start node has one edge and its "front node" has three. Therefore, a step can also close the branch, by joining the exposed node to the start node instead of another node. Every node then has two edges again, and the ℓ removals and ℓ additions of the closed branch form one ℓ-opt move, whose gain g_ℓ is the decrease in route length. The move is valid when the reconnected route is a single cycle. Otherwise, the closing step is discarded, but

the branch stays in the tree, because a deeper closing step can still be valid. A branch can close at every depth, so a k-opt search, whose tree has at most k levels, can make an ℓ-opt move for every $2 \leq \ell \leq k$.

However, some improving moves are out of reach for every branch. The standard example is the double bridge, a 4-opt move made of two 2-opt moves. The first is a *splitting move*, that is, a 2-opt move with the reconnection that reverses no segment and therefore splits the route into two loops. The second is the *fix step*, a 2-opt move that removes one edge from each loop and joins the two loops into a single cycle. Applied alone, the fix step is a splitting move too, so neither of the two is valid alone and the tree discards each of them as a closing step. In addition, no branch contains both, because every step of a branch removes an edge of the "front node" of the step before it, and the two moves share no node.

The Sequential Search of LKH LKH searches this tree depth-first, that is, it follows one branch as deep as it can, backtracks when the gain is no longer positive, and tries the next branch. When a closed branch gives a shorter route, LKH applies this one move and starts again from the new route. When no branch gives a shorter route, LKH runs a non-sequential search for the moves that no branch contains. The search continues as long as it finds a better route. Each decision therefore depends on the result of the decision before it, and the work differs from route to route. In contrast, a GPU is efficient when many routes perform the same computation at the same moment, so this search runs poorly on a GPU.

Our Parallel Beam Search Our approach keeps the steps, the gain and the closing rule of LKH. One *sweep* searches the tree once, from the current route down to the largest depth k, and then updates the route, and sweeps repeat until no route improves. To fit a GPU, a sweep searches the tree with five changes. Figure B.1 draws one sweep.

(1) A candidate set for every added edge. Instead of trying every node in the graph as the new "front node", each step adds an edge only to one of the few nodes nearest the exposed node, its **candidate set**, which loses little because a short route rarely uses a long edge. At the root, this reduces the $\Theta(N^2)$ branches of Section 4.3 to $\Theta(N)$, because the candidate set size is fixed.

(2) A beam of M branches. Instead of one branch that deepens and backtracks, **the beam** advances M branches together, level by level, where level ℓ holds the branches of depth ℓ. Each level extends every surviving branch by both route edges of its "front node" and by every candidate of the exposed node, computes all gains in one pass, and keeps the M branches with the largest positive gains with one sort. Every route in the batch therefore performs the same computation at the same moment. In addition, the beam narrows with depth by a width-shrink factor f, keeping M branches at the first level, M/f at the second, and M/f^2 at every deeper level, so deep levels cost less.

(3) All closing steps of a level at once. Instead of testing one closing step at a time, each level also closes every branch in the beam. Closing a branch only records a move, so the route does not change during a sweep, and the branch itself is still extended to the next level as in (2). The valid closed branches are the *closures* of the level, and across its levels a sweep keeps the closures with the largest gains, up to a fixed number per sweep.

(4) Double-bridge moves in every sweep. Instead of a separate non-sequential search, every sweep also searches for double-bridge moves. It scores, in one pass, every splitting move that adds an edge between a node and one of the candidates of that node, and keeps the splitting moves with the largest gains. Two of them whose cuts alternate along the route form a double bridge, because the second is the fix step of the first, and the gain of the double bridge is the sum of their gains. A sweep keeps as many double-bridge moves as closures.

(5) Many moves per sweep. Instead of applying one move at a time, a sweep applies many kept moves together, closures and double-bridge moves alike, as long as they do not overlap on the route. Specifically, each kept move spans a range of route positions, from the first to the last node it touches, and moves whose ranges are at least one position apart can be applied together, because their gains add up and the route stays a single cycle. We find the set of such ranges with the largest total gain by dynamic programming over the ranges sorted by their right end (weighted interval scheduling).

CVRP Variant A CVRP solution is written as one closed route that returns to the depot once per vehicle, so the GPU beam k-opt search carries over with two changes. First, the gain of a move becomes its decrease in **the penalized objective** $c(x) = \frac{1}{2}\, d^\top x + \rho \sum_r \max\big(0,\, \mathrm{load}(r) - Q\big)$,

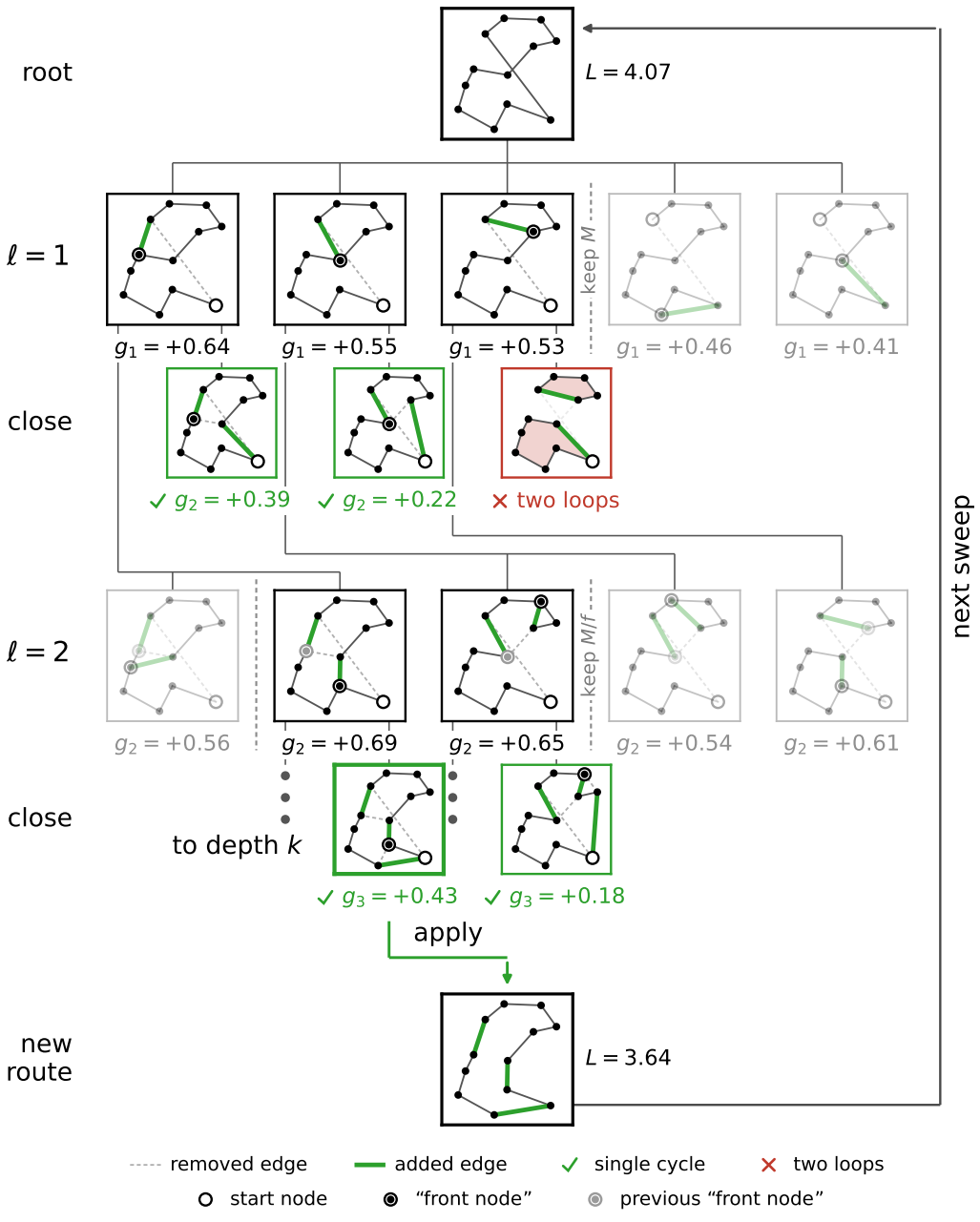


Figure B.1: An example sweep of the GPU beam k-opt search on a 12-node instance. Every panel draws the same nodes and differs only in its edges. Each level extends every branch of the beam and keeps the branches with the largest gains, and the beam narrows with depth. Each level also closes every branch back to the start node, and a closing that splits the route into two loops is discarded. The sweep applies the kept closures that do not overlap on the route, here only the one with the largest gain. Sweeps repeat until no route improves.

where $\mathrm{load}(r)$ sums the demands of sub-route r, Q is the vehicle capacity and ρ weighs overload, so moves are scored on load as well as on length, and the beam k-opt search can pass through a briefly overloaded route to reach a better feasible one. Second, following LKH-3 (Helsgaun, 2017), which extends Lin-Kernighan to capacitated routing, the moves also transfer customers and segments between sub-routes rather than only rearranging edges within one. In addition, a beam sweep is

expensive, so instead of running the beam k-opt search on each of the S parallel routes, we rank them with a short 2-opt pass and run the search once, on the best of them.

Beam k-opt Search Hyperparameters Table B.5 lists the settings of the GPU beam k-opt search, which are the same at every test size. Its last row counts the steps of the 2-opt ranking pass of the CVRP variant, and TSP runs no 2-opt before the beam k-opt search.

Table B.5: GPU beam k-opt search parameters.

Parameter	TSP	CVRP
Candidate set size	15	20
Largest k of a k-opt move (tree depth)	5	5
Beam width M	131072	32768
Beam width-shrink factor f	4	4
Closures kept per sweep	256	128
2-opt steps before the beam k-opt search	0	500

B.6 Hardware

All evaluations were conducted on a single NVIDIA L40S GPU, on a machine with an Intel Xeon Gold 6430 CPU, and so was every training run up to target size 1000. The runs that target size 10000 train on a single NVIDIA B200 or H200 GPU. The classical solvers run single-threaded on one core of this CPU.

C Configurations of MaskCO and GenSCO

Correction-Cycle Baselines MaskCO (Chen et al., 2026) and GenSCO (Li et al., 2025) run their released per-size decoding settings, except the values in Table C.1. MaskCO releases no setting above test size 1000, so the size-10000 rows extend its size-1000 settings.

Table C.1: Correction-cycle baseline settings. Values in parentheses are test sizes.

Parameter	TSP	CVRP
Correction cycles C	1, 10	1, 10
2-opt moves per cycle	500 (500), 1000 (1000, 10000)	6000
Settings at test size 10000		
Top-k edge slice	2×10^6	10^6
Batch size	1	1

MaskCO Self-Train MaskCO's optimal-solution-free training (Chen et al., 2026) replaces the labels of the target size with the model's own solutions. We run it from the released MaskCO model of the source size. In each round, the best model so far by gap solves a set of newly generated target-size instances, and the model is fine-tuned on these solutions with the supervised objective. The gap flattens within four rounds, and the curves of Figures 3 and 4 use the best model by gap of those rounds.

D Additional Results and Ablations

D.1 The Cost of Labels Against Label-Free GRPO

We compare two ways to train a model for a large target size, both starting from the same supervised model trained at size 1000. The first continues supervised training at the target size, which requires labels computed by a classical solver, LKH-3 for TSP and HGS for CVRP. The second is the label-free GRPO training of ResizeCO, which requires no labels. We measure the cost of each in total device hours, the CPU hours of computing the labels plus the GPU hours of training, and report the gap on TSP-10000 and CVRP-10000 as a function of the total device hours in Figures D.1 and D.2.

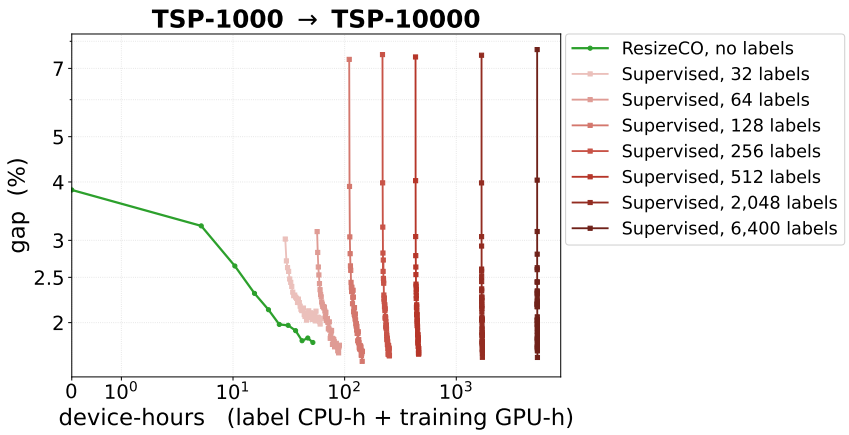


Figure D.1: Gap on TSP-10000 against total device hours, the CPU hours of computing the LKH-3 labels plus the GPU hours of training. Every run starts from the same supervised model trained at size 1000. The green line is the label-free GRPO training of ResizeCO, and each red line is supervised training with a fixed number of labels.

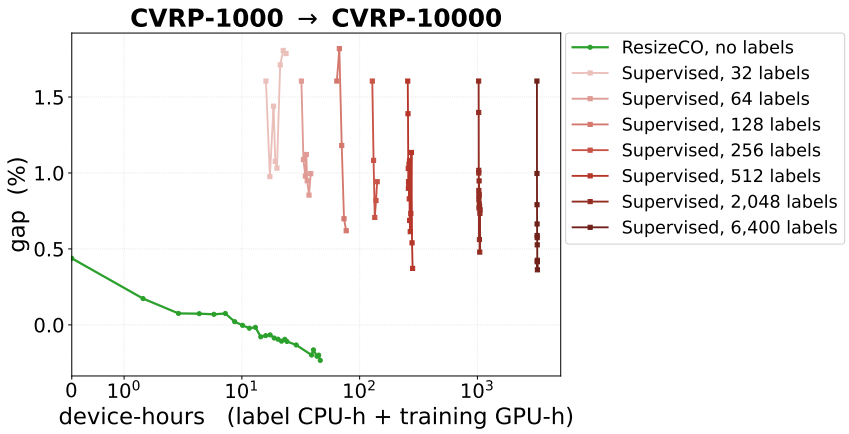


Figure D.2: Gap on CVRP-10000 against total device hours, the CPU hours of computing the HGS labels plus the GPU hours of training. Every run starts from the same supervised model trained at size 1000. The green line is the label-free GRPO training of ResizeCO, and each red line is supervised training with a fixed number of labels.

D.2 All Source-to-Target Pairs

This subsection reports all six source-to-target pairs of each problem, grouped by source size. The baseline rows follow Tables 1 and 3. Every ResizeCO row runs one correction cycle ($C = 1$) with one unmasking step ($K = 1$), and the rows vary the number of parallel samples from $S = 1$ to $S = 32$, so each column group shows the gap and the run time at six compute budgets.

Tables D.1 and D.2 hold the three pairs that start from the models trained at size 100, with two blocks of ResizeCO rows. The first block trains one RL run for every pair, as in the main tables. The second block reports a single RL run that starts from the size-100 supervised model and draws the instance size of each training batch from a mix of sizes, so one model serves all three target sizes.

Tables D.3 and D.4 hold the three pairs that start from the models trained at sizes 500 and 1000. They carry only the block with one RL run for every pair, because the mixed-size run starts from the size-100 supervised model.

At target size 10000 the ResizeCO models of the first block reach the target in two RL stages. A model of size 100 or 500 is first trained at size 1000 and only then at size 10000, a curriculum that

keeps each stage within one order of magnitude of its starting size, while the pair from size 1000 is a single stage. It applies to these rows alone.

Table D.1: Size-generalization CVRP results from the models trained at size 100. Conventions follow Table 1.

	CVRP-100 → CVRP-500			**CVRP-100 → CVRP-1000**			**CVRP-100 → CVRP-10000**		
Method	**Obj**	**Gap**	**Time**	**Obj**	**Gap**	**Time**	**Obj**	**Gap**	**Time**
Classical solvers									
HGS (reference) (Vidal, 2022)	62.15	0.000%	55 h	121.07	0.000%	197 h	1120.23	0.000%	512 h
HGS (Vidal, 2022)	63.99	2.96%	52 s	124.02	2.44%	2.4 m	1129.62	0.846%	2.2 h
Supervised learning									
BQ-NCO (G) (Drakulic et al., 2023)	64.39	3.67%	26 s	125.19	3.53%	36 s	1221.13	9.12%	1.3 h
LEHD (greedy) (Luo et al., 2023)	64.40	3.63%	18 s	125.41	3.66%	55 s		Time Limit	
Label-free learning									
INViT-3V (Fang et al., 2024)	67.69	9.03%	6.1 m	131.78	8.97%	10.8 m	1192.43	6.49%	2.1 h
GLOP-G (LKH-3) (Ye et al., 2024)	-	-	-	-	8.83%*	1.4 m*	-	8.27%*	23.9 m*
SIT (PRC10) (Luo et al., 2025a)	-	-	-	-	4.52%*	45 s*	-	4.43%*	14.5 m*
ReLD (augx8) (Huang et al., 2025)	64.16	3.29%	21 s	125.06	3.38%	40 s		OOM	
Diffusion-based									
MaskCO (T=128) (Chen et al., 2026)	65.79	5.90%	11 s	128.25	6.03%	11 s	1167.39	4.31%	1 h
MaskCO (T=1280) (Chen et al., 2026)	65.39	5.25%	1.6 m	127.67	5.54%	1.6 m	1163.54	3.94%	10.2 h
Ours, one RL run per target									
ResizeCO (S=1)	64.10	3.18%	14 s	125.13	3.43%	16 s	1119.27	−0.056%	49.7 m
ResizeCO (S=2)	64.02	3.04%	12 s	124.89	3.23%	20 s	1119.27	−0.057%	54.7 m
ResizeCO (S=4)	63.91	2.88%	14 s	124.86	3.20%	25 s	1119.09	−0.073%	1.2 h
ResizeCO (S=8)	63.88	2.83%	15 s	124.76	3.12%	28 s	1118.98	−0.082%	1.5 h
ResizeCO (S=16)	63.82	2.72%	22 s	124.64	3.02%	48 s	1118.93	−0.088%	2.1 h
ResizeCO (S=32)	63.81	2.71%	28 s	124.63	3.01%	1.3 m	1118.78	−0.102%	4.1 h
Ours, one RL run for all targets									
ResizeCO (S=1)	64.27	3.47%	12 s	124.56	2.95%	17 s	1121.31	0.130%	45.5 m
ResizeCO (S=2)	64.22	3.38%	11 s	124.49	2.90%	15 s	1121.08	0.109%	50.1 m
ResizeCO (S=4)	64.09	3.18%	13 s	124.37	2.80%	17 s	1120.90	0.092%	58.7 m
ResizeCO (S=8)	64.04	3.10%	14 s	124.29	2.73%	23 s	1120.82	0.085%	1.3 h
ResizeCO (S=16)	63.98	2.99%	17 s	124.23	2.68%	37 s	1120.70	0.074%	2 h
ResizeCO (S=32)	63.96	2.96%	19 s	124.17	2.63%	1.1 m	1120.62	0.067%	3.6 h

Table D.2: Size-generalization TSP results from the models trained at size 100. Conventions follow Table 3.

	TSP-100 → TSP-500			TSP-100 → TSP-1000			TSP-100 → TSP-10000		
Method	**Obj**	**Gap**	**Time**	**Obj**	**Gap**	**Time**	**Obj**	**Gap**	**Time**
Classical solvers									
Concorde / LKH-3 (reference)	16.55	0.000%	37.66 m	23.12	0.000%	6.65 h	71.83	0.000%	8.8 h
LKH-3 (trials=32) (Helsgaun, 2017)	16.55	0.026%	22 s	23.13	0.036%	56 s	71.81	−0.023%	7.7 m
LKH-3 (trials=1) (Helsgaun, 2017)	16.57	0.118%	8.6 s	23.15	0.139%	23 s	71.89	0.085%	5.6 m
Supervised learning									
BQ-NCO (G) (Drakulic et al., 2023)	16.74	1.19%	23 s	23.65	2.29%	54 s	89.07	24.00%	9.4 m
LEHD (greedy) (Luo et al., 2023)	16.80	1.53%	15 s	23.84	3.12%	1.7 m	91.47	27.34%	2.8 h
Label-free learning									
INViT-3V (Fang et al., 2024)	17.27	4.38%	4.9 m	24.30	5.10%	9.8 m	76.57	6.61%	33.2 m
GLOP (Ye et al., 2024)	17.07	3.20%	19 s	24.01	3.85%	38 s	75.61	5.27%	29 s
SIT (PRC10) (Luo et al., 2025a)	16.70	0.947%	28 s	23.39	1.18%	1.3 m	74.01	3.04%	2.1 m
Diffusion-based									
DIFUSCO (T_s=50, G) (Sun & Yang, 2023)	17.09	3.27%	2.2 m	24.09	4.19%	10 m	76.68	6.75%	41.7 m
DIFUSCO (T_s=100, S=4) (Sun & Yang, 2023)	16.97	2.53%	21.6 m	23.94	3.53%	1.2 h		Time Limit	
Fast T2T (T_s=5, T_g=5, G) (Li et al., 2024)	17.30	4.58%	2.5 m	24.39	5.48%	10.9 m		Time Limit	
Fast T2T (T_s=5, T_g=5, S=4) (Li et al., 2024)	17.16	3.71%	11.2 m	24.22	4.75%	24.6 m		OOM	
GenSCO (C=1, S=8) (Li et al., 2025)	17.19	3.88%	2.5 s	24.96	7.99%	20 s	351.17	388.91%	2.8 m
GenSCO (C=10, S=8) (Li et al., 2025)	17.05	3.06%	25 s	24.80	7.26%	3.3 m	350.60	388.11%	28.1 m
MaskCO (T=8/16) (Chen et al., 2026)	17.21	3.99%	1 s	24.99	8.10%	28 s	332.06	362.30%	3.1 m
MaskCO (T=80/160) (Chen et al., 2026)	17.07	3.18%	10 s	24.80	7.27%	3.9 m	321.28	347.30%	31 m
Ours, one RL run per target									
ResizeCO (S=1)	16.60	0.329%	2.6 s	23.31	0.813%	7.5 s	74.72	4.03%	43 s
ResizeCO (S=2)	16.59	0.261%	3.3 s	23.28	0.689%	9.6 s	74.62	3.89%	1.3 m
ResizeCO (S=4)	16.58	0.213%	4.8 s	23.26	0.599%	16 s	74.57	3.82%	2.2 m
ResizeCO (S=8)	16.58	0.180%	12 s	23.24	0.541%	30 s	74.55	3.80%	4 m
ResizeCO (S=16)	16.57	0.152%	17 s	23.24	0.515%	52 s	74.49	3.71%	7.7 m
ResizeCO (S=32)	16.57	0.144%	38 s	23.23	0.482%	1.9 m	74.45	3.65%	15.3 m
Ours, one RL run for all targets									
ResizeCO (S=1)	16.61	0.360%	2.2 s	23.34	0.968%	8.1 s	74.90	4.27%	48 s
ResizeCO (S=2)	16.59	0.283%	4.1 s	23.31	0.838%	11 s	74.80	4.14%	1.4 m
ResizeCO (S=4)	16.58	0.214%	5.9 s	23.29	0.746%	18 s	74.76	4.09%	2.6 m
ResizeCO (S=8)	16.57	0.174%	13 s	23.28	0.690%	33 s	74.74	4.06%	4.7 m
ResizeCO (S=16)	16.57	0.147%	23 s	23.27	0.645%	59 s	74.71	4.01%	9 m
ResizeCO (S=32)	16.57	0.138%	43 s	23.26	0.615%	1.9 m	74.67	3.95%	17 m

Table D.3: Size-generalization CVRP results from the models trained at sizes 500 and 1000. Conventions follow Table 1.

	CVRP-500 → CVRP-1000			CVRP-1000 → CVRP-10000			CVRP-500 → CVRP-10000		
Method	**Obj**	**Gap**	**Time**	**Obj**	**Gap**	**Time**	**Obj**	**Gap**	**Time**
Classical solvers									
HGS (reference) (Vidal, 2022)	121.07	0.000%	197 h	1120.23	0.000%	512 h	1120.23	0.000%	512 h
HGS (Vidal, 2022)	124.02	2.44%	2.4 m	1129.62	0.846%	2.2 h	1129.62	0.846%	2.2 h
Supervised learning									
BQ-NCO (G) (Drakulic et al., 2023)	-	-	-	-	-	-	-	-	-
LEHD (greedy) (Luo et al., 2023)	-	-	-	-	-	-	-	-	-
Label-free learning									
INViT-3V (Fang et al., 2024)	131.78	8.97%	10.8 m	1192.43	6.49%	2.1 h	1192.43	6.49%	2.1 h
GLOP-G (LKH-3) (Ye et al., 2024)	-	8.83%*	1.4 m*	-	8.27%*	23.9 m*	-	8.27%*	23.9 m*
SIT (PRC10) (Luo et al., 2025a)	-	4.52%*	45 s*	-	4.43%*	14.5 m*	-	4.43%*	14.5 m*
ReLD (augx8) (Huang et al., 2025)	125.06	3.38%	40 s		OOM			OOM	
Diffusion-based									
MaskCO (T=128) (Chen et al., 2026)	125.79	3.98%	11 s	1197.55	7.97%	1 h	1639.21	45.15%	1 h
MaskCO (T=1280) (Chen et al., 2026)	125.79	3.98%	1.6 m	1146.56	2.36%	10.6 h	1127.41	0.686%	10.5 h
Ours, one RL run per target									
ResizeCO (S=1)	124.35	2.76%	13 s	1118.83	−0.096%	49.9 m	1117.51	−0.217%	46 m
ResizeCO (S=2)	124.21	2.64%	14 s	1118.60	−0.117%	55.4 m	1117.36	−0.231%	50.9 m
ResizeCO (S=4)	124.08	2.54%	18 s	1118.50	−0.125%	1 h	1117.27	−0.240%	1 h
ResizeCO (S=8)	124.00	2.48%	26 s	1118.32	−0.143%	1.4 h	1117.25	−0.241%	1.3 h
ResizeCO (S=16)	123.96	2.44%	44 s	1118.20	−0.151%	2.2 h	1117.09	−0.256%	1.9 h
ResizeCO (S=32)	123.82	2.32%	1.2 m	1118.08	−0.162%	4.3 h	1117.06	−0.259%	3.8 h

Table D.4: Size-generalization TSP results from the models trained at sizes 500 and 1000. Conventions follow Table 3.

	TSP-500 → TSP-1000			TSP-1000 → TSP-10000			TSP-500 → TSP-10000		
Method	**Obj**	**Gap**	**Time**	**Obj**	**Gap**	**Time**	**Obj**	**Gap**	**Time**
Classical solvers									
Concorde / LKH-3 (reference)	23.12	0.000%	6.65 h	71.83	0.000%	8.8 h	71.83	0.000%	8.8 h
LKH-3 (trials=32) (Helsgaun, 2017)	23.13	0.036%	56 s	71.81	−0.023%	7.7 m	71.81	−0.023%	7.7 m
LKH-3 (trials=1) (Helsgaun, 2017)	23.15	0.139%	23 s	71.89	0.085%	5.6 m	71.89	0.085%	5.6 m
Supervised learning									
BQ-NCO (G) (Drakulic et al., 2023)	-	-	-	-	-	-	-	-	-
LEHD (greedy) (Luo et al., 2023)	-	-	-	-	-	-	-	-	-
Label-free learning									
INViT-3V (Fang et al., 2024)	24.30	5.10%	9.8 m	76.57	6.61%	33.2 m	76.57	6.61%	33.2 m
GLOP (Ye et al., 2024)	24.01	3.85%	38 s	75.61	5.27%	29 s	75.61	5.27%	29 s
SIT (PRC10) (Luo et al., 2025a)	23.39	1.18%	1.3 m	74.01	3.04%	2.1 m	74.01	3.04%	2.1 m
Diffusion-based									
DIFUSCO (T_s=50, G) (Sun & Yang, 2023)	23.64	2.27%	9.6 m	74.11	3.18%	29.6 m	75.15	4.63%	33.3 m
DIFUSCO (T_s=100, S=4) (Sun & Yang, 2023)	23.48	1.56%	1.2 h		Time Limit			Time Limit	
Fast T2T (T_s=5, T_g=5, G) (Li et al., 2024)	23.46	1.48%	8.9 m	75.20	4.70%	3.6 h	76.16	6.03%	3.9 h
Fast T2T (T_s=5, T_g=5, S=4) (Li et al., 2024)	23.37	1.10%	20.1 m		OOM			OOM	
GenSCO (C=1, S=8) (Li et al., 2025)	23.17	0.245%	20 s	89.77	24.97%	3.4 m	118.85	65.46%	3.3 m
GenSCO (C=10, S=8) (Li et al., 2025)	23.15	0.138%	3.3 m	89.34	24.38%	34 m	118.41	64.85%	33.3 m
MaskCO (T=16) (Chen et al., 2026)	23.27	0.662%	23 s	75.60	5.26%	3.4 m	75.79	5.52%	3.4 m
MaskCO (T=160) (Chen et al., 2026)	23.15	0.138%	3.8 m	75.24	4.75%	33.9 m	75.41	4.99%	33.8 m
Ours, one RL run per target									
ResizeCO (S=1)	23.17	0.208%	5.9 s	73.30	2.05%	34 s	74.10	3.16%	42 s
ResizeCO (S=2)	23.16	0.165%	7.6 s	73.23	1.95%	53 s	74.01	3.04%	1.1 m
ResizeCO (S=4)	23.15	0.136%	11 s	73.19	1.90%	1.4 m	73.99	3.01%	1.9 m
ResizeCO (S=8)	23.14	0.112%	17 s	73.16	1.85%	2.5 m	73.95	2.96%	3.5 m
ResizeCO (S=16)	23.14	0.099%	31 s	73.14	1.83%	4.6 m	73.92	2.91%	6.6 m
ResizeCO (S=32)	23.14	0.096%	1 m	73.14	1.83%	8.8 m	73.89	2.87%	12.8 m

D.3 Component Ablation from TSP-100 to TSP-1000

Table 5 ablates the components of ResizeCO on two source-to-target pairs. Table D.5 repeats the ablation on a third pair, TSP-100 → TSP-1000. Each row removes a single component. Removing any component increases the gap, most of all label-free GRPO, from 0.813% to 3.23%.

Table D.5: Component ablation of ResizeCO from TSP-100 to TSP-1000. Conventions follow Table 5.

	TSP-100 → TSP-1000	
Method	**Gap**	**Time**
ResizeCO (full)	**0.813%**	**7.5 s**
w/o label-free GRPO	3.23%	10.7 s
w/o small-model initialization	2.67%	10.6 s
w/o Graph Transformer encoder	2.42%	8.7 s
w/o GPU beam k-opt search	1.29%	7.7 s

D.4 Why Does Node-Walk Decoding Help Training?

Previous heatmap-based CO solvers decode with the greedy decoder (Sun & Yang, 2023). The decoder a GRPO run uses for its rollouts decides whether the training improves the model at all, so we compare three runs from the same supervised model that differ only in the decoder (Figure D.3). Node-walk decoding scored by its exact likelihood, Eq. (3), improves steadily. Greedy decoding scored by that same likelihood falls below the supervised model it started from, because the node-walk likelihood never sampled the routes a random edge ranking produces, which biases the gradient. **Greedy decoding scored by the exact probability of its own ranking removes the bias, and the collapse with it, yet the model still does not learn.** The obstacle is therefore not the validity of the gradient.

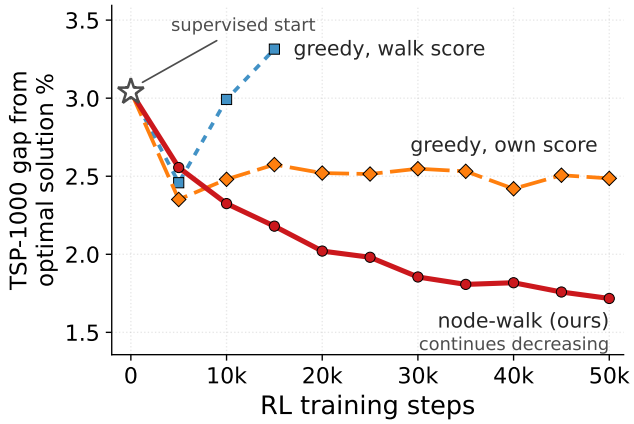


Figure D.3: Three GRPO runs from the same supervised model. The runs differ only in how routes are sampled and scored during RL training. **Blue:** sampling with the greedy decoder and scoring with the node-walk likelihood, the TSP-1000 gap rises above the supervised starting point. **Orange:** sampling with the greedy decoder and scoring with its own exact likelihood, the gap stays flat. **Red:** sampling and scoring with node-walk decoding (our approach), the gap falls steadily.

We attribute the failure to the way each decoder normalizes the transition probability p_θ. The node-walk decoder weighs an edge against the edges from n_j to the unvisited nodes U_j, giving $p^t_{\theta,\,n_j n_{j+1}} / \sum_{n' \in U_j} p^t_{\theta,\,n_j n'}$, a normalization within a row, as in the supervised loss. In contrast, the greedy decoder weighs it against the set $\mathcal{B}$ of all edges still masked in the graph, giving $p^t_{\theta,\,uv} / \sum_{(u',v') \in \mathcal{B}} p^t_{\theta,\,u'v'}$, which compares unrelated edges. At inference, the greedy decoder works well, but we deploy the node-walk decoder, so training and inference use the same decoder.

E Derivation of the Training Loss

This appendix derives the training loss of Section 4.2. We start from the standard GRPO loss of Eq. (1), written for our N decisions,

$$\mathcal{L}_{\mathrm{GRPO}}(\theta) = \frac{1}{G} \sum_{g=1}^{G} \frac{1}{N} \sum_{j=1}^{N} \Big[- \min\big(r_j^g(\theta)\, A_g,\, \mathrm{clip}\big(r_j^g, 1{-}\varepsilon, 1{+}\varepsilon\big) A_g\big) + \beta\, D_{\mathrm{KL}}\big(\pi_\theta \,\|\, \pi_{\mathrm{ref}}\big)(s_j) \Big],$$

where $r_j^g(\theta) = \pi_\theta(a_j \mid s_j) \,/\, \pi_{\theta_{\mathrm{old}}}(a_j \mid s_j)$ is the ratio between the current policy and the policy that sampled the group. Throughout this appendix, $\pi_\theta(a_j \mid s_j)$ is short for the node-walk policy $\pi_\theta(a_j \mid s_j; p^t_\theta)$ of Eq. (2).

In our case, we adapt the loss in two ways. First, we take a **single update per batch** and resample fresh rollouts, which are cheap to draw. The rollouts are therefore always sampled by the current policy, so $\theta_{\mathrm{old}} = \theta$. The ratios r_j^g then equal one, the clipping never takes effect, and the first term of Eq. (1) reduces to $- r_j^g(\theta)\, A_g$. This term has the same gradient as $- A_g \log \pi_\theta(a_j \mid s_j)$, because

$$\nabla_\theta\, r_j^g(\theta) = \frac{\nabla_\theta\, \pi_\theta(a_j \mid s_j)}{\pi_{\theta_{\mathrm{old}}}(a_j \mid s_j)} = \frac{\nabla_\theta\, \pi_\theta(a_j \mid s_j)}{\pi_\theta(a_j \mid s_j)} = \nabla_\theta \log \pi_\theta(a_j \mid s_j). \tag{7}$$

The first equality treats $\pi_{\theta_{\mathrm{old}}}$ as a constant, because θ_{old} is fixed during the update. The second substitutes $\theta_{\mathrm{old}} = \theta$. The third is the derivative of a logarithm. Summing over the N decisions and using the route log-probability of Eq. (3), the loss of one route becomes its advantage times the route log-probability,

$$- \frac{1}{N}\, A_g \sum_{j=1}^{N} \log \pi_\theta(a_j \mid s_j) = - \frac{1}{N}\, A_g \log p_\theta(\mathrm{route}_g). \tag{8}$$

Second, we apply the **KL penalty to the whole edge heatmap**, not only to the visited states. This is because a gradient update changes the model weights θ, and with them every row of the heatmap,

even rows that no rollout read. At inference, the node-walk decoder reads such rows as well, so the whole heatmap must stay near the reference p_{ref}. Specifically, the per-state KL of Eq. (1) is replaced by the row average

$$D_{\mathrm{KL}}\big(p_\theta \,\|\, p_{\mathrm{ref}}\big) = \frac{1}{N}\sum_{i=1}^{N} D_{\mathrm{KL}}\big(p_{\theta,\,i} \,\|\, p_{\mathrm{ref},\,i}\big), \tag{9}$$

where $p_{\theta,\,i}$ is row i of the heatmap, averaged over all heatmaps in the rollout.

Putting the two changes together, **the final loss** averages the route loss over the group and adds the KL penalty,

$$\mathcal{L}_{\mathrm{ResizeCO}}(\theta) = -\frac{1}{N}\frac{1}{G}\sum_{g=1}^{G} A_g \log p_\theta(\mathrm{route}_g) + \beta\, D_{\mathrm{KL}}\big(p_\theta \,\|\, p_{\mathrm{ref}}\big). \tag{10}$$