Learning to Search, Searching to Learn: A Closed-Loop Framework for Large-Scale Routing Problems

Yongji Fu[1] **Yi Zhou**[2] **Gaojie Jin**[3] **Guanqun Cao**[4†]
[1]University of Bristol [2]Cardiff University
[3]University of Macau
[4]Bristol Robotics Laboratory, University of the West of England, Bristol

Abstract

Scaling neural solvers to large Vehicle Routing Problems (VRPs) runs into two recurring difficulties. To stay within compute budgets, scalable solvers tend to rely on partitioning, local candidate restriction, or staged decisions, which can restrict the cross-region structure available to a fixed-budget solver. To improve the final solution, many neural pipelines fall back on a classical search, but use it as a one-shot post-processor. The model predicts, search repairs, and no sustained feedback is formed between them. We propose **LSSL** (**L**earning to **S**earch, **S**earching to **L**earn), a closed-loop learning-search framework for large-scale VRP that addresses both difficulties in one framework. LSSL predicts a search-friendly guidance heatmap on a sparse candidate graph. A classical heuristic backend then uses this heatmap to refine the incumbent solution, and the structural state returned by the backend is fed back to recondition the model. As a result, learning and search interact within a closed loop, rather than operating as two detached stages. Built on a sparse diffusion transformer over an $O(NK)$ candidate graph, the proposed method scales to ultra-large instances on a single consumer-grade GPU without hard decomposition. Experiments on large-scale TSP and CVRP benchmarks demonstrate that LSSL achieves superior scalability, efficiency, and solution quality. Ablation studies further reveal that the closed learning–search loop architecture accounts for a substantial portion of the observed performance gains. Code is available at https://github.com/yongjifu7/LSSL

1 Introduction

The Vehicle Routing Problem (VRP) is a central problem in combinatorial optimization, with applications in logistics and supply-chain planning [1–4]. Because VRP is NP-hard, exact optimization is impractical at large scale [5]. Strong heuristics and metaheuristics such as Lin-Kernighan-Helsgaun (LKH) for the Traveling Salesman Problem (TSP) and Hybrid Genetic Search (HGS) for the Capacitated VRP (CVRP) remain the strong practical baselines, but rely on engineered candidate sets, neighborhoods, and search control [6, 7]. Neural Combinatorial Optimization (NCO) learns solving policies or structural priors directly from data [8–14]. While this idea has worked well on small and medium instances, making it work reliably on ultra-large VRPs remains difficult.

A core difficulty is scaling learning-based solvers to very large instances (e.g., 100K nodes) under a bounded compute budget while still capturing global structure. Recent work has improved cross-scale generalization through stronger constructive solvers, better inductive biases, or test-time adaptation [15–21]. Some works decompose ultra-large instances into subproblems and solve them

†Corresponding author.

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

hierarchically or in stages [22–28]. Tight budgets can localize decisions and make long-range coordination difficult, although decomposition can also preserve useful structure and complement iterative refinement. This is especially critical in ultra-large VRP, because many local connections can only be judged properly in the context of a full route, or even the coordination among multiple routes.

Another core difficulty is that neural pipelines still use classical heuristic search almost exclusively as a single post-processing pass on the model output. Many works show that edge heatmaps, candidate-edge scores, or initial solutions produced by neural models can substantially improve final results when combined with downstream search or refinement [29–37]. For example, NeuroLKH demonstrates the effectiveness of combining neural guidance with classical heuristic search, showing that a learned module can provide valuable candidate information to an LKH backend and thereby improve final solution quality [38]. However, this coupling is usually still one-way, with only a single prediction followed by a single refinement, while trajectory structure is neither fed back during inference nor mined for training. Search improves the initial solution but rarely participates as a sustained, closed-loop part of learning. Although recent work has started to question the limitation of treating search only as post-hoc refinement [39–42], how to unify learning with a classical heuristic search backend into a closed-loop system that operates in both inference and training remains an open question for large-scale VRP.

To address the above difficulties, we propose **LSSL** (**L**earning to **S**earch, **S**earching to **L**earn), a closed-loop framework that connects learning and search during training and inference. *Learning to Search* means the model does not calculate a full route in one pass; it predicts a sparse edge heatmap on a sparse candidate graph, and heuristic search uses that heatmap as guidance. In the *Searching to Learn* step, LSSL retains multiple high-quality route sets encountered during search on each training instance, projects them onto the shared candidate graph, and aggregates these projections into row-wise soft targets for model updates. These targets place higher mass on edges repeatedly supported by good solutions, while keeping uncertainty where several local connections remain plausible. At inference time, LSSL closes the complementary loop on the same instance with multiple learning–search cycles. After each search round, the incumbent route set is projected back to a structural heatmap that conditions the next model pass, and search spends another budget refining the incumbent under the refreshed guidance. As a result, training targets distill consensus from search trajectories, while inference propagates search feedback through repeated reconditioning. All quantities exchanged between the model and the search backend are defined on a sparse candidate graph with $O(NK)$ edges for fixed K. Prediction and feedback are therefore updated row-wise over only K neighbours, yielding near-linear scaling in N rather than the quadratic cost of dense $O(N^2)$ models. This makes large VRP instances feasible under limited computational resources.

Experiments on large-scale TSP and CVRP benchmarks show that LSSL simultaneously delivers strong solution quality, inference efficiency, and scalability, attaining state-of-the-art results in multiple settings while remaining tractable on a single consumer GPU at node N up to 100K. The contributions of this paper are as follows:

- **LSSL framework.** A sparse diffusion Transformer D_θ and a heuristic backend S are coupled through a shared sparse candidate graph. At inference, they alternate to refine the same instance; at training, search-refined solutions are aggregated into row-wise soft targets that supervise D_θ.
- $O(NK)$ **scaling.** Block-streamed online-softmax attention keeps memory and compute near-linear in N, letting the closed loop run end-to-end on one consumer GPU at 100K nodes without divide-and-conquer.
- **Empirical findings.** LSSL matches or outperforms strong heuristics and neural-guided baselines on large-scale TSP and CVRP under matched compute; ablations attribute distinct gains to inference reconditioning and training-time consensus supervision.

2 LSSL: Closed-loop Learning–Search Framework

2.1 Large-scale VRP problem formulation

Given a VRP instance on a node set $\mathcal{V}$ with $N = |\mathcal{V}|$ nodes, we seek a feasible solution $\mathbf{x}^\star$ that minimizes the total tour or route length. TSP and CVRP differ in their feasibility constraints: TSP solutions form a single Hamiltonian tour that visits every node exactly once, while CVRP solutions decompose into depot-rooted routes that jointly serve all customers under a vehicle-capacity constraint.

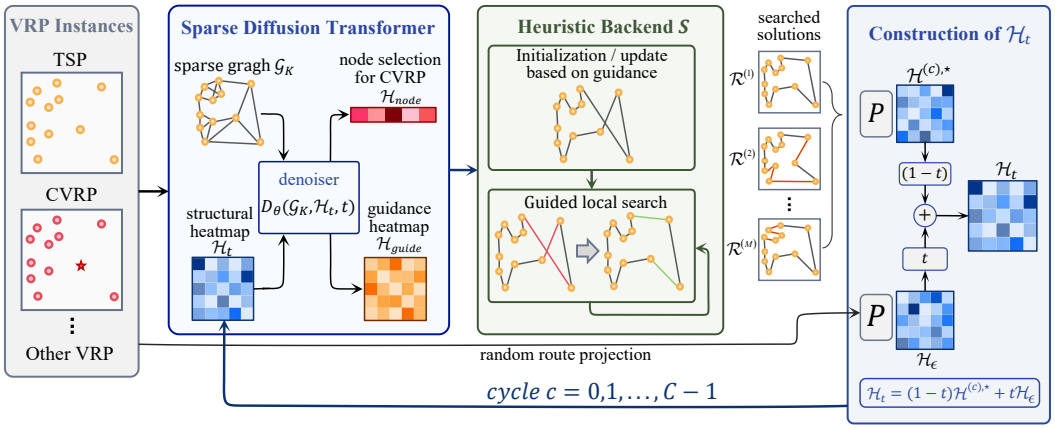


Figure 1: **Overview of LSSL.** The Sparse Diffusion Transformer takes the current structural state of the solution and the instance graph as input, and outputs a guidance heatmap $\mathcal{H}_{\text{guide}}$ over promising local connections. For CVRP, it additionally outputs a node-selection heatmap $\mathcal{H}_{\text{node}}$ to anchor route boundaries. The backend uses this guidance first to construct an initial solution and then to locally improve it; the improved solutions are used to update the structural state and fed back as the starting point of the next cycle. We instantiate this loop for TSP and CVRP; additional constraints would require new features, training data, and a feasibility-preserving backend.

At the scales we target (N up to 100K), materialising the full $O(N^2)$ edge set is prohibitive in memory and time. However, for Euclidean instances, the optimal solution rarely uses long edges, so most of those $O(N^2)$ candidates are not actually useful. LSSL therefore operates entirely on a sparse K-NN candidate subgraph of size $O(NK)$ with $K \ll N$ (Section 2.3). This sparse representation is the foundation on which both LSSL's neural backbone and its search backend operate.

2.2 Framework overview

The LSSL framework (Figure 1) closes a loop between a sparse diffusion Transformer D_θ and a classical heuristic backend S on a shared sparse candidate graph: D_θ produces guidance heatmaps that S consumes, and S's refined output is projected back as the input for the next call to D_θ. Particularly, the framework rests on three components. The *denoiser* D_θ is a sparse diffusion Transformer that consumes the instance graph $\mathcal{G}_K$, a structural heatmap $\mathcal{H}_t$ over candidate edges, and a noise level $t \in [0, 1]$, and produces a guidance heatmap $\mathcal{H}_{\text{guide}}$. The *search backend* S consumes $\mathcal{H}_{\text{guide}}$ and refines the incumbent route set under a per-cycle budget; we instantiate it as Guided-LKH [6] for TSP and Guided-HGS [43, 44] for CVRP (Appendices H and I). The *projection* P maps any candidate solution $\mathcal{R}$ (a TSP tour or a CVRP route set) onto the sparse $N \times K$ adjacency by indicating which candidate edges are actually used in $\mathcal{R}$.

In our framework, training (Section 2.5) supplies D_θ with targets prepared by S: for each instance, several S-refined route sets are projected through P and aggregated into a consensus heatmap $\mathcal{H}^\star$, and D_θ is trained to recover $\mathcal{H}^\star$ from a noisy interpolation between $\mathcal{H}^\star$ and a random projection. Inference (Section 2.6) runs C cycles per instance, starting from a random feasible solution. Each cycle projects the current incumbent through P to obtain a structural heatmap, denoises it with D_θ along a noise schedule into a guidance heatmap, and calls S under that guidance to produce the next incumbent; the best solution across cycles is returned.

Backend interface and scope. Guided-LKH replaces α-nearness candidates with heatmap-ranked candidates; Guided-HGS uses model-ranked candidates with geometric backfill. Each cycle passes the incumbent and refreshed priorities forward. Search operators, feasibility rules, stopping criteria, randomization, and parallelization remain unchanged under the stated budget. An adapter maps predicted edge and CVRP node/depot priorities to candidate ordering, warm-starts the backend, and projects its feasible output back to a structural heatmap via P. Backend-agnostic describes this interface, not interchangeability of the sparse diffusion predictor. A new constrained VRP needs

constraint-aware features and candidates, suitable training data, and a feasible backend; solvers without external-priority or warm-start support require additional engineering.

2.3 Guidance Heatmap on a Sparse Candidate Graph

Existing diffusion-style generative NCO methods [32–35] train a predictor to recover a reference solution $\mathbf{x}^\star$ from a perturbed state $\mathbf{x}_t$, where $\mathbf{x}_t$ is itself a perturbed *solution* variable and is therefore pulled toward feasibility throughout the trajectory (Section C surveys the variants). This commits prematurely: the model fixes edge-level decisions before search begins, leaving search only local refinement around that commitment (as shown in Figure 2). The issue is especially severe for CVRP, where capacity and depot constraints aggressively drive $\mathbf{x}_t$ onto the feasibility manifold.

LSSL instead parameterises a guidance heatmap over candidate edges. The model only expresses how likely each local connection is, leaving the assembly of a feasible solution entirely to search. Concretely, the global graph $\mathcal{G}$ is first mapped to a sparse candidate graph

$$\mathcal{G}_K = (\mathcal{V}, \mathcal{E}_K), \qquad \mathcal{E}_K = \{(i,j) \mid j \in \mathcal{N}_K(i),\ |\mathcal{N}_K(i)| = K\}, \tag{1}$$

with per-node neighborhood $\mathcal{N}_K(i)$ which compresses the dominant cost from $O(N^2)$ to $O(NK)$.

On this sparse graph, we define the sparse diffusion Transformer D_θ as a per-step denoiser,

$$\mathcal{H}_{\text{guide}} = D_\theta(\mathcal{G}_K,\ \mathcal{H}_t,\ t), \tag{2}$$

where $\mathcal{H}_t \in [0,1]^{N\times K}$ is a structural prior over candidate edges at noise level $t \in [0,1]$. Each row i assigns a probability mass to each of node i's K candidate neighbors, expressing how likely the edge (i,j) belongs to a high-quality solution. The two endpoints anchor opposite extremes. At $t=0$, $\mathcal{H}_t$ is a clean prior fully informative about high-quality solution regions; at $t=1$, $\mathcal{H}_t$ is a structureless random prior. The precise interpolation between them is given by Eq. (7) in Section 2.5. Crucially, $\mathcal{H}_{\text{guide}}$ is never required to satisfy combinatorial feasibility constraints, leaving full room for downstream search to adjust.

2.4 Sparse diffusion Transformer

The denoiser D_θ consists of a sparse encoder and a time-conditioned sparse decoder, both operating on the same $N \times K$ sparse adjacency. The encoder takes per-node static features $\mathbf{s}_i$ (coordinates for TSP, plus normalized demand for CVRP) and propagates them across $\mathcal{G}_K$ through standard sparse self-attention, producing instance-aware node embeddings. The decoder takes these embeddings as initial state and additionally injects two dynamic signals, i.e., the current heatmap $\mathcal{H}_t$ and the noise level t, through the structural-bias channel below.

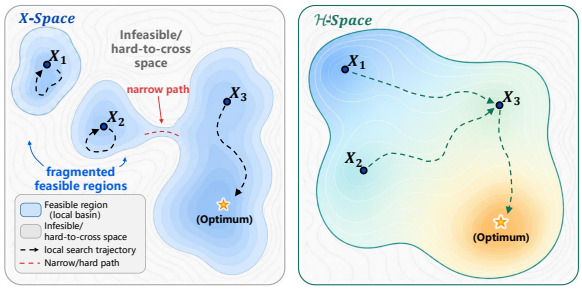


Figure 2: **x-space vs. $\mathcal{H}$-space.** In x-space, feasibility constraints fragment the landscape into disconnected basins, trapping local search. $\mathcal{H}$-space provides a smooth, connected representation that unifies all initialisations with the optimum, enabling global exploration.

Decoder attention. At decoder layer ℓ, head h, the attention score from node i to a candidate neighbor $j \in \mathcal{N}_K(i)$ is

$$e_{ij}^{(h)} = \frac{\langle \mathbf{q}_i^{(h)},\ \mathbf{k}_j^{(h)} \rangle}{\sqrt{d_h}} + b_{ij}, \qquad b_{ij} \ = \ u\big([\mathcal{H}_t]_{ij}\big) + v\big(\phi(t)\big), \tag{3}$$

where $\mathbf{q}_i^{(h)}, \mathbf{k}_j^{(h)}$ are the query and key projections of the current hidden state, d_h is the per-head dimension, $\phi(t)$ is a sinusoidal embedding of t, and u, v are learned scalar projections. The edge-dependent term modulates attention under $\mathcal{H}_t$. The row-constant scalar $v(\phi(t))$ cancels under softmax and does not itself sharpen attention. Time conditioning also enters the decoder hidden state through the time bridge specified in Appendix F. The attention scores then drive a row-wise softmax over $\mathcal{N}_K(i)$, aggregate the value vectors $\mathbf{v}_j^{(h)}$, and feed into a standard FFN-residual block; stacking L such layers yields the final per-node representation $\mathbf{g}_i \in \mathbb{R}^{d_{\text{dec}}}$ at the decoder output.

To compute exact sparse attention without storing all attention weights, we visit each row's K neighbors in blocks, maintaining a running maximum, normalization sum, and weighted-value accumulator. Rescaling these statistics when the maximum changes preserves the standard row-softmax output; the temporary attention workspace depends on block width rather than all K weights. The full sparse graph still occupies $O(NK)$ storage (Appendix F.4).

Edge-scoring head. A lightweight head turns the final node representations $\mathbf{g}_i$ into the guidance heatmap by a row-wise softmax over candidate-neighbor inner products,

$$[\mathcal{H}_{\text{guide}}]_{ij} = \frac{\exp\big(\langle \mathbf{g}_i, \mathbf{g}_j \rangle\big)}{\sum_{j' \in \mathcal{N}_K(i)} \exp\big(\langle \mathbf{g}_i, \mathbf{g}_{j'} \rangle\big)}, \qquad j \in \mathcal{N}_K(i). \tag{4}$$

Two nodes whose final representations align in $\mathbb{R}^{d_{\text{dec}}}$ are scored as a likely edge, so the geometry encoded by the encoder, the structural belief carried by $\mathcal{H}_t$, and the time-dependent sharpening from $\phi(t)$ all flow through a single output channel.

2.5 Search-feedback training

The training procedure of LSSL specifies how search outputs are recycled into supervision on the sparse candidate graph. Unlike GenSCO [35], which supervises the model with a single high-quality tour per instance, LSSL aggregates multiple search-refined route sets per instance into a row-wise structural consensus and trains the denoiser as a direct endpoint predictor of this consensus.

For each training instance, the search backend S produces M high-quality route sets $\{\mathcal{R}^{(m)}\}_{m=1}^{M}$. The search-feedback target $\mathcal{H}^\star$ is the projection of these solutions onto the sparse candidate graph $\mathcal{G}_K$:

$$\mathcal{H}^\star \;=\; P\big(\{\mathcal{R}^{(m)}\}_{m=1}^{M}; \mathcal{G}_K\big). \tag{5}$$

Here P extends the projection of Section 2.2 from a single solution to a set: it counts per-row edge usage across the M route sets and row-normalises. Specifically, for vertex i and candidate slot $j \in \mathcal{N}_K(i)$,

$$\big[P(\{\mathcal{R}^{(m)}\}_{m=1}^{M}; \mathcal{G}_K)\big]_{ij} \;=\; \frac{\sum_{m=1}^{M} \mathbb{1}\big\{(i,j) \in \mathcal{R}^{(m)}\big\}}{\sum_{j' \in \mathcal{N}_K(i)} \sum_{m=1}^{M} \mathbb{1}\big\{(i,j') \in \mathcal{R}^{(m)}\big\}}, \tag{6}$$

where $\mathbb{1}\{(i,j) \in \mathcal{R}^{(m)}\}$ equals 1 iff j is a tour-neighbour of i in $\mathcal{R}^{(m)}$. The result is a sparse structural consensus over multiple search route sets: edges repeatedly selected by good solutions receive higher probability mass, while alternative local connections remain plausible when search has not clearly resolved them.

We next define the source state of the structural transition. Let $\mathcal{R}_\epsilon$ be a randomly initialized route set generated without using the search-feedback target. Its sparse projection $\mathcal{H}_\epsilon = P(\mathcal{R}_\epsilon; \mathcal{G}_K)$ serves as a random route projection. Given the clean target $\mathcal{H}^\star$ and the random route projection $\mathcal{H}_\epsilon$, we construct a continuous interpolation path on the sparse $N \times K$ adjacency:

$$\mathcal{H}_t = (1-t)\mathcal{H}^\star + t\mathcal{H}_\epsilon, \qquad t \sim \mathcal{U}(0,1). \tag{7}$$

Here $t = 0$ corresponds to the clean search-feedback target and $t = 1$ to the randomly initialized structural state. This interpolation should not be interpreted as a feasible VRP solution trajectory; it is instead a relaxed structural-prior trajectory over candidate edges, so neither the clean target $\mathcal{H}^\star$ nor the intermediate states $\mathcal{H}_t$ are required to satisfy TSP or CVRP feasibility constraints. This relaxation lets D_θ focus on learning structural patterns rather than enforcing feasibility, which is left to downstream search at inference time.

Training minimizes a row-wise sparse cross-entropy on $\mathcal{G}_K$:

$$\mathcal{L}_{\text{LSSL}} = -\frac{1}{N} \sum_{i=1}^{N} \sum_{j \in \mathcal{N}_K(i)} \mathcal{H}^\star_{ij} \log\big[D_\theta(\mathcal{G}_K, \mathcal{H}_t, t)\big]_{ij}. \tag{8}$$

CVRP additionally uses a per-customer node-selection loss. The decoder outputs a node-selection heatmap $\mathcal{H}_{\text{node}} \in (0,1)^{N_c}$, supervised by a binary target $\mathcal{H}^\star_{\text{node}} \in \{0,1\}^{N_c}$ that marks customers

starting or ending a sub-route in $\{\mathcal{R}^{(m)}\}_{m=1}^{M}$:

$$\mathcal{L}_{\text{node}} \;=\; -\frac{1}{N_c}\sum_{i=1}^{N_c}\Big[[\mathcal{H}^{\star}_{\text{node}}]_i\,\log[\mathcal{H}_{\text{node}}]_i \;+\; (1-[\mathcal{H}^{\star}_{\text{node}}]_i)\,\log(1-[\mathcal{H}_{\text{node}}]_i)\Big], \tag{9}$$

with N_c the number of customers. The total CVRP objective combines the edge and node losses:

$$\mathcal{L}_{\text{CVRP}} \;=\; \mathcal{L}_{\text{LSSL}} \;+\; \lambda_{\text{node}}\,\mathcal{L}_{\text{node}}. \tag{10}$$

Self-bootstrapping across rounds. The search-feedback mechanism supports iterative self-improvement at training time. In round 0, the search backend S is applied once to each training instance, and its M refined route sets are aggregated via Eq. (5) into the target $\mathcal{H}^{\star}$; D_θ is then trained on the resulting (instance, target) pairs. In each subsequent round, a subset of instances is re-sampled, and the current D_θ generates guidance heatmaps that direct S to produce new route sets. These are re-aggregated into an updated $\mathcal{H}^{\star}$ that replaces the previous target for the same instance, and D_θ is fine-tuned on the updated training set.

This loop is particularly valuable at scales where obtaining external high-quality labels is intractable. By coupling a smaller-scale-trained D_θ with S, self-bootstrapping generates supervision that improves with each round, without relying on a stronger external solver. Our main experiments use a single self-improvement round; details of the training schedule are deferred to Appendix E.

2.6 Inference-time learning–search refinement loop

The denoiser D_θ is trained as a direct endpoint predictor: given $\mathcal{G}_K$, an intermediate state $\mathcal{H}_t$ on the interpolation path of Eq. (7), and the noise level t, it directly estimates the clean target $\mathcal{H}^{\star}$. Unlike velocity-matching parameterizations, D_θ predicts this endpoint at every $t \in [0,1]$, so the same network can later be unrolled along a noise-axis schedule at inference.

Inference runs C learning–search cycles on the same instance, alternating D_θ and the search backend S through the projection P. Algorithm 1 describes one full pass. Starting from a random feasible initialisation $\mathcal{R}_\epsilon$ as $\mathcal{R}^{(0)}$, each cycle (i) projects the current incumbent $\mathcal{R}^{(c)}$ to a structural heatmap $\mathcal{H}^{(c),\star}$ on $\mathcal{G}_K$, (ii) unrolls D_θ along the noise-axis schedule $1{=}t_0{>}\cdots{>}t_T{=}0$ to obtain a guidance heatmap $\mathcal{H}^{(c)}_{\text{guide}}$, and (iii) invokes S under that guidance with budget B to refine the incumbent. The best feasible solution across the C cycles is returned. Backend-specific details are deferred to Appendices H and I.

The inner loop is a deterministic integrator that calls D_θ at every schedule step. Since D_θ is supervised at every $t \in [0,1]$ (Eq. (8)), it can be queried at any t_r to obtain a current estimate of the clean target $\mathcal{H}^{\star}$. The next state is then obtained by sliding along the linear interpolation path of Eq. (7) from t_r to t_{r+1}, with the model's estimate playing the role of the clean endpoint:

$$\begin{aligned}\mathcal{H}_{t_{r+1}} = {}& \tfrac{t_{r+1}}{t_r}\,\mathcal{H}_{t_r} \\ & + \big(1-\tfrac{t_{r+1}}{t_r}\big)\,D_\theta(\mathcal{G}_K,\mathcal{H}_{t_r},t_r),\end{aligned} \tag{11}$$

a convex combination with mixing weight $1 - t_{r+1}/t_r$ on the prediction; the trajectory converges to $\mathcal{H}^{\star}$ as $t_r \to 0$.

Algorithm 1 LSSL inference (one instance).

Require: $\mathcal{G}_K$; D_θ; backend S; cycles C; schedule $\{t_r\}_{r=0}^{T}$, $t_0{=}1$, $t_T{=}0$; budget B
Ensure: best solution $\bar{\mathcal{R}}$

build candidate graph $\mathcal{G}_K$
$\mathcal{R}^{(0)} \leftarrow \mathcal{R}_\epsilon$ ▷ random feasible init
$\bar{\mathcal{R}} \leftarrow \mathcal{R}^{(0)}$
for $c = 0, \dots, C-1$ **do**
  $\mathcal{H}^{(c),\star} \leftarrow P(\mathcal{R}^{(c)}; \mathcal{G}_K)$ ▷ (i)
  $\mathcal{H}_{t_0} \leftarrow \mathcal{H}^{(c),\star}$
  for $r = 0, \dots, T-1$ **do** ▷ (ii)
    update $\mathcal{H}_{t_{r+1}}$ by Eq. (11)
  end for
  $\mathcal{H}^{(c)}_{\text{guide}} \leftarrow \mathcal{H}_{t_T}$
  $\mathcal{R}_S \leftarrow S(\mathcal{R}^{(c)}, \mathcal{H}^{(c)}_{\text{guide}}, B)$ ▷ (iii)
  if $\text{cost}(\mathcal{R}_S) < \text{cost}(\bar{\mathcal{R}})$ **then**
    $\bar{\mathcal{R}} \leftarrow \mathcal{R}_S$
  end if
  $\mathcal{R}^{(c+1)} \leftarrow \bar{\mathcal{R}}$ ▷ best-so-far carries to next cycle
end for
return $\bar{\mathcal{R}}$

3 Experiments

In this section, we systematically evaluate LSSL from four perspectives on large-scale TSP and CVRP. First, we compare it with classical solvers, constructive NCO methods, heatmap-based NCO methods, and two-stage large-scale routing methods on synthetic benchmarks to assess overall solution quality, inference efficiency,

Table 1: Results on synthetic TSP and CVRP instances. OOM: The method triggered an out-of-memory condition. N.T.: Not trained; the original publication did not train or report a model for this problem scale. C: The number of LSSL learning–search cycles; B_{trials}: The per-cycle LKH trial count. B_t: The per-cycle HGS time budget in seconds.

Method	TSP100 Length	TSP100 Gap	TSP100 Time	TSP1K Length	TSP1K Gap	TSP1K Time	TSP10K Length	TSP10K Gap	TSP10K Time
LKH3	7.763	0.000%	3.5m	23.119	0.000%	30.75m	71.778	0.000%	33.2m
Concorde	7.763	0.000%	2.125m	23.119	0.000%	29.25m	72	0.310%	1.4h
DIFUSCO	7.78	0.237%	12.4m	23.56	1.913%	12.1m	73.62	2.58%	14.5m
Fast-T2T	7.76	0.012%	7.1m	23.22	0.422%	15.6m	72.94	1.63%	8.7m
GenSCO (C=10) 2Opt	7.763	0.001%	16.2s	23.145	0.112%	30.2s	N.T.	N.T.	N.T.
GenSCO (C=160) 2Opt	7.763	0.000%	3.4m	23.132	0.057%	7.1m	N.T.	N.T.	N.T.
H-TSP	-	-	-	24.718	6.912%	34s	77.75	8.320%	2.2m
GLOP	7.767	0.048%	5.25m	23.84	3.119%	10.2s	75.04	4.545%	1.9m
UDC-$\mathbf{x}_{50}$ ($\alpha = 50$)	7.788	0.319%	1.38h	23.53	1.782%	6.21m	OOM	OOM	OOM
POMO aug×8	7.774	0.138%	21.3s	32.5	40.577%	2.1m	OOM	OOM	OOM
BQ bs16	7.764	0.010%	1.25m	23.432	1.354%	1.31m	OOM	OOM	OOM
LEHD RRC1000	7.763	0.002%	8.63m	23.288	0.731%	19.5m	80.9	12.709%	13.5m
L2C-Insert Greedy	7.812	0.628%	7.8s	23.963	3.808%	5.4s	77.299	7.727%	40.6s
L2C-Insert (I=1000)	7.692	0.000%	2.87m	23.185	0.438%	9.50m	73.227	2.052%	10.44m
LSSL (C=1, B_{trials}=128)	7.763	0.000%	1.3m	23.122	0.017%	17.1s	71.789	0.015%	32.2s
LSSL (C=2, B_{trials}=256)	-	-	-	23.120	0.007%	38.3s	71.783	0.007%	1.1m
LSSL (C=4, B_{trials}=512)	-	-	-	23.119	0.000%	3.6m	71.773	-0.006%	3.3m

Method	CVRP100 Length	CVRP100 Gap	CVRP100 Time	CVRP1K Length	CVRP1K Gap	CVRP1K Time	CVRP10K Length	CVRP10K Gap	CVRP10K Time
HGS	15.592	0.000%	63m	41.212	0.000%	3.5h	226.177	0.000%	1h
GLOP	-	-	-	45.753	11.019%	1.93m	276.252	22.140%	27.98m
UDC-$\mathbf{x}_{50}$ ($\alpha = 50$)	16.291	4.481%	2.69h	43.583	5.753%	11.96m	OOM	OOM	OOM
POMO aug×8	15.75	1.011%	17.2s	100.990	145.045%	1.76m	OOM	OOM	OOM
BQ bs16	15.81	1.396%	1.13m	43.121	4.632%	1.56m	275.494	21.805%	8.6m
LEHD RRC1000	15.63	0.242%	7.50m	42.408	2.902%	15.56m	248.604	9.916%	30.38m
L2C-Insert Greedy	16.580	4.001%	46s	43.076	7.691%	12.5s	254.990	12.739%	2.35m
L2C-Insert (I=1000)	16.011	0.433%	5.32m	41.704	4.261%	23.95m	245.940	8.738%	26.81m
LSSL (C=1, B_t=4)	15.591	-0.009%	4.2m	41.681	1.138%	25.0s	228.32	0.947%	4.7s
LSSL (C=2, B_t=32)	15.589	-0.021%	66.7m	41.328	0.281%	6.7m	224.69	-0.657%	64.1s
LSSL (C=4, B_t=128)	-	-	-	41.186	-0.063%	53.3m	222.49	-1.630%	8.5m

and scalability. Second, we evaluate cross-distribution generalization on real-world TSPLIB, VLSI, Art-TSP, and CVRPLIB instances. Third, we conduct ablations on the closed-loop mechanism and search-feedback supervision to verify whether the core design truly brings performance gains. Finally, we analyze the behavior of the framework under different inference budgets, cycle counts, candidate widths, and large-scale scaling curves.

3.1 Experimental setup

Datasets. A TSP instance contains N 2-D coordinates and a reference tour; a CVRP instance further specifies vehicle capacity. Following [11], synthetic instances are sampled uniformly from $[0, 1]^2$. Training sets are multi-scale: TSP uses 1.28M / 64K / 6.4K / 640 instances at N=100/1K/10K/100K, and CVRP uses 1.28M / 64K / 6.4K instances at N=100/1K/10K. At small scales (TSP100, TSP1K, CVRP100), targets are obtained directly from the solver (Appendix E.2). At the larger scales TSP10K, TSP100K, CVRP1K, and CVRP10K, directly labelling the full training split with high-quality solver outputs is no longer feasible: the per-instance time required exceeds any uniform labelling pipeline that fits our offline training budget. We therefore use the self-bootstrapping mechanism of Section 2.5: a smaller-scale-trained LSSL model D_θ produces guidance heatmaps on freshly sampled target-scale instances and directs the search backend S to solve them; the resulting route sets are projected and aggregated via Eq. (5) into the search-feedback target $\mathcal{H}^\star$, on which the target-scale LSSL model is then trained as a fixed dataset. Self-improved training has been explored as a precedent for large-scale NCO [19]. The full recipe and per-scale time costs are reported in Appendices E. Test sets follow [11, 32, 24, 21]: TSP100 has 1,280 instances, TSP1K 128, CVRP100 1,280, CVRP1K 100 [24], and TSP/CVRP10K and TSP100K each 16; CVRP capacities are 50/200/300 for N=100/1K/$\geq$10K [24]. For real-world generalization we use TSPLIB [45],

Table 2: Compact reproduction settings. Budgets are per cycle.

Setting	TSP	CVRP
Training scales	100 / 1K / 10K / 100K	100 / 1K / 10K
Direct targets	100 / 1K	100
Bootstrapped targets	10K / 100K	1K / 10K
Candidates	Full for $N \leq$ 1K; $K = 1000$ for $N >$ 1K	
Denoising	Eight steps per learning–search cycle	
Cycles	$C \in \{1, 2, 4\}$	
Per-cycle search	128 / 256 / 512 trials	$B_t = 4/32/128$ s
Evaluation allocation	One RTX 4090 (24 GiB), 16 CPU cores	

Table 3: Results on synthetic TSP100K.

Method	TSP100K Length	TSP100K Gap	TSP100K Time
LKH3	225.997	0.000%	25h
GLOP	237.610	5.139%	3.90m
INViT greedy	242.260	7.196%	5.00h
L2C-Insert Greedy	242.757	7.420%	6.86m
L2C-Insert (I=1000)	237.221	4.971%	16.71m
L2C-Insert (I=10000)	230.132	1.834%	1.77h
LSSL (C=1, B_{trials}=128)	226.219	0.098%	16.9m
LSSL (C=2, B_{trials}=512)	226.028	0.014%	1.92h
LSSL (C=4, B_{trials}=1024)	225.997	0.000%	6.86h

Table 4: Results on real-world TSP benchmarks.

Method	TSPLIB 50–1K	VLSI 50–2K	VLSI 2K–200K	Art-TSP 100K–200K
UDC-$\mathbf{x}_{250}$ ($\alpha = 50$)	3.072%	3.942%	-	-
GLOP	1.153%	3.881%	8.051%	3.238%
GenSCO	0.184%	1.588%	-	-
BQ bs16	1.557%	6.259%	-	-
L2C-Insert (I=1000)	1.499%	2.051%	4.142%	4.240%
NeuroLKH	0.020%	−0.061%	-	-
LSSL (C=1, B_{trials}=128)	0.002%	−0.264%	−0.273%	0.034%
LSSL (C=2, B_{trials}=256)	0.000%	−0.267%	−0.299%	0.026%

VLSI [46], Art-TSP [47], and CVRPLIB Set-X / Set-XL [48], with reference costs computed from the tours published on each benchmark's official site; per-instance results are in the Appendix D.

Model Setting. An LSSL configuration is specified by C (number of learning–search cycles) and the per-cycle search size: the LKH trial count B_{trials} with backend Guided-LKH for TSP, and the HGS time budget B_t with backend Guided-HGS for CVRP. Table 2 summarises the training scales, label sources, candidate widths, and per-cycle budgets. The full baseline list is deferred to Appendix E.5.

Metrics and timing. Length and gap summarize test-set solution quality; negative gaps indicate improvement over the reference, not certified optimality. The original main tables report complete-test-set time with up to 16 concurrent instances when memory permits. Appendix A additionally reports Serial-1 (one active instance) and maximum feasible batching under one RTX 4090 (24 GiB) and 16 CPU cores, without an extra neural batch cap. Both include preprocessing, neural inference, and all backend calls; neither time column is per-instance latency. LSSL is a complete learning-enhanced search system, so comparisons with standalone neural constructors do not isolate the neural module alone.

3.2 Experimental Results

Results on Synthetic Dataset. Tables 1 and 3 report the solving performance comparison on synthetic TSP and CVRP. LSSL matches or improves on the reference solvers at the reported high-budget settings, and this advantage grows with scale. On TSP10K, LSSL (C=4, B_{trials}=512) drives the gap to -0.006% in just 3.3 m, surpassing LKH3 itself at roughly $10\times$ less computation than LKH3 (33.2 m). On TSP100K, LSSL (C=4, B_{trials}=1024) fully closes the gap to 0.000% in 6.86 h, roughly a $3.6\times$ speedup over LKH3 (25 h). On CVRP10K, LSSL (C=4, B_t=128) drives the gap to -1.630% in just 8.5 m, improving on the HGS reference, at roughly $7\times$ less computation than HGS (1 h). By contrast, the other neural baselines either OOM by $\geq$ 10K, including GenSCO, Fast-T2T and UDC-$\mathbf{x}_{50}$, or sit at gaps an order of magnitude wider, for example L2C-Insert (I=10,000) still lands at 1.834% on TSP100K after 1.77 h, and on CVRP10K LEHD RRC1000 and L2C-Insert (I=1000) remain at 9.916% and 8.738% respectively.

Results on Real-world Benchmark. Tables 4 and 5 report cross-distribution generalization on TSPLIB, VLSI, Art-TSP, and CVRPLIB Set-X / Set-XL. LSSL transfers cleanly from synthetic uniform $[0, 1]^2$ to all four distributions, and on the larger benchmarks the average tour is shorter than the published references themselves: at C=2 the gap is -0.267% on VLSI 50–2K and -0.299% on 2K–200K, despite never training on VLSI-style points; on TSPLIB 50–1K it closes to 0.000%, and on Art-TSP 100K–200K to 0.026%. Within each benchmark, raising the search budget tightens the gap monotonically, and the marginal improvement tracks how suboptimal the reference is: $0.002 \to 0.000\%$ (TSPLIB) and $-0.264 \to -0.267\%$ (VLSI 50–2K) where the reference is near-optimal, versus $1.067 \to 0.171\%$ (Set-X) and $4.149 \to 1.838\%$ (Set-XL) where it is far from optimal. Against the strongest neural baselines, the largest reductions appear where post-hoc heatmap and constructive methods struggle most: a $9.5\times$ smaller gap than L2C-Insert on Set-XL and a $124\times$ smaller gap than GLOP on Art-TSP.

Table 5: Results on real-world CVRP benchmarks. †:Several instances are not solvable due to exceeding time limits.

Method	CVRPLIB-setX 100–1K	CVRPLIB-setXL 1K–10K
UDC-$\mathbf{x}_{250}$ ($\alpha = 50$)	7.592%	-
BQ bs16	6.160%	-
L2C-Insert (I=1000)	3.223%	17.494%
NeuroLKH	2.496%	-
LSSL (C=1, B_t=4)	1.067%	4.149%†
LSSL (C=2, B_t=32)	0.353%	2.505%†
LSSL (C=4, B_t=128)	0.171%	1.838%

Takeaway. Two complementary regimes emerge across scales. At small to medium scale, the LSSL heatmap concentrates the backend on near-optimal edges from the first trial, bypassing the geometry-driven startup that vanilla LKH and HGS rely on; the corresponding anytime curves in Fig. 3 make this explicit, with $\mathcal{H}$-space already at 0.28% on CVRP10K at $t \approx 10$ s while Search-Only HGS still sits at 3.44%. At large scale, the complete learning–search system remains competitive with the reference solvers. On CVRP100K, a memory-efficient HGS implementation enables a Serial-1 stress test (Appendix A, Table 15): LSSL ($C = 2$, $B_t = 512$ s) obtains a -1.201% gap in 4.55 h versus 80.6 h for the HGS reference. Both times cover the whole test split, and no model is trained at the CVRP100K scale. Additional experiments compare neural constructors followed by LKH/HGS and learning-enhanced classical solvers under explicit budget rules (Appendix A).

3.3 Ablation studies

Setup. On the same backend (LKH3 for TSP, HGS for CVRP) and under a matched total computation time per panel, we compare four variants tied to the primitives in Section 2. **Search-Only** runs S with no learned prior. **$\mathbf{x}$-space** freezes a single prediction supervised by the hard label $\mathbf{x}^\star$ projected onto $\mathcal{G}_K$, as in NeuroLKH [38]. **$\mathcal{H}$-space** likewise freezes a single prediction but with the LSSL soft target $\mathcal{H}^\star$ from Eq. (5) as supervision, i.e. LSSL at C=1. **LSSL** with $C \in \{2, 4, 8\}$ uses the same supervision but refreshes the prior from the current incumbent at every cycle, splitting the total budget into C equal cycles. Figure 3 reports anytime mean-gap curves on TSP/CVRP $\times$ 1K/10K, with $\mathbf{x}$-space shown on the 1K panels.

$\mathcal{H}$-space vs. $\mathbf{x}$-space. Comparing $\mathbf{x}$-space with $\mathcal{H}$-space at C=1 isolates the supervision form alone: both variants make one prediction before search starts and keep that guidance frozen for the entire backend budget. On TSP1K, $\mathcal{H}$-space reaches a terminal gap of 0.00034% vs. 0.00066% for $\mathbf{x}$-space, roughly halving it; on CVRP1K, it reaches 0.686% vs. 0.974%. Replacing the single-solution hard label with the multi-solution search-feedback target therefore consistently improves one-shot guidance, and refreshing this guidance across cycles yields a further gain, which we isolate below.

Closing the loop vs. freezing the prior. The anytime comparison changes both refresh and cycle structure, so it alone cannot separate guidance refresh from restart effects. We therefore add a 32 s control with $C = 2$ in both arms, matching restarts/reseeding, budget splits, seeds, warm-start handling, and backend settings. Frozen guidance yields 0.00037% / 0.6877% on TSP1K / CVRP1K, while refreshed guidance yields 0.00024% / 0.5942%. The continuous $C = 1$ frozen control gives 0.00034% / 0.6945%. Thus refresh improves quality even at matched cycle structure (Table 17); a target-source $\times$ cycle-count experiment separately tests bootstrapping and multi-solution supervision (Appendix B).

An intermediate refresh rate works best. At fixed total computation time, we sweep the refresh count over $C = 1, 2, 4, 8$. The results show that more feedback is not always better. On CVRP1K,

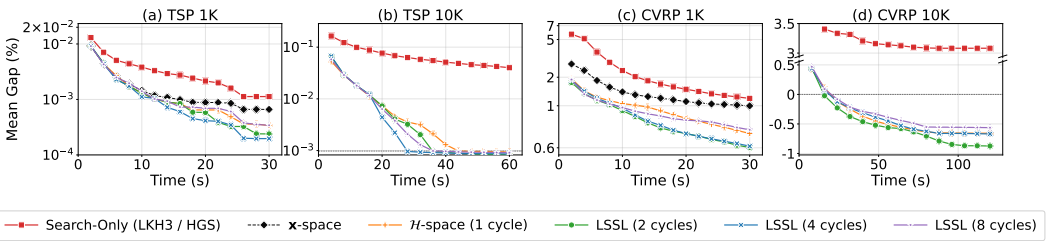


Figure 3: **Performance of LSSL on TSP/CVRP at 1K/10K.** Anytime mean-gap under matched total time per panel (32/64/32/128 s), split into C equal cycles. $\mathbf{x}$-space (shown on the 1K panels) and $\mathcal{H}$-space (1 cycle) are the frozen single-prediction variants from the Setup paragraph; LSSL refreshes the prior every cycle.

the terminal gap changes from 0.686 to 0.590, 0.595, and 0.731% as C increases; on CVRP10K, it changes from -0.657 to -0.878, -0.671, and -0.565%. The best setting is therefore $C{=}2$ on TSP10K, CVRP1K, and CVRP10K alike. This pattern reflects a balance between stale guidance and premature refresh: small C leaves the prior frozen for too long, while large C gives each backend cycle too little time to exploit the current prior. CVRP10K at $C{=}8$ being worse than $C{=}1$ makes this second failure mode especially clear.

Candidate width and CVRP node supervision. A TSP10K sweep over $K = 100, 500, 1000, 2000$ shows quality saturation at $K = 1000$ while neural cost continues to grow. On CVRP1K, we distinguish disabling node guidance at inference from retraining with $\lambda_{\text{node}} = 0$. At 32 s, the full model gives 0.666%; disabling its node guidance gives 0.759%, while the retrained edge-only model gives 0.699%. Unsupervised node guidance from the latter model gives 0.790%. The full jointly trained and deployed model is best, but these results do not establish that node supervision alone improves edge representations (Appendix B).

4 Conclusion

This work presents a closed-loop perspective on neural combinatorial optimization for large-scale VRP. In this perspective, learning and classical search are two endpoints of a single feedback loop, rather than two stages of a one-shot prediction-and-repair pipeline. We instantiate this perspective with LSSL, which couples a globally aware sparse backbone with a heuristic search backend through a shared structural interface. Under this interface, search consumes priors from the model, while the model is in turn supervised by the structures that search produces. Inference iterates this loop within each instance, refreshing the guidance as the search progresses. Extensive experiments on large-scale TSP and CVRP show that LSSL outperforms recent neural solvers, and under matched compute it also improves on a strong pure-search backend. These results suggest that classical search remains an essential component of large-scale VRP solving, and that neural priors are most effective when coupled with such search rather than positioned as its replacement.

Acknowledgments and disclosures

This work received no external funding. The authors declare no relevant competing interests.

References

[1] Bruce L Golden, S Raghavan, and Edward A Wasil. *The Vehicle Routing Problem: Latest Advances and New Challenges*, volume 43. Springer Science & Business Media, 2008.

[2] Paolo Toth and Daniele Vigo. *Vehicle routing: problems, methods, and applications*. SIAM, 2014.

[3] Victor Pillac, Michel Gendreau, Christelle Guéret, and Andrés L Medaglia. A review of dynamic vehicle routing problems. *European Journal of Operational Research*, 225(1):1–11, 2013.

[4] Grigorios D Konstantakopoulos, Sotiris P Gayialis, and Evripidis P Kechagias. Vehicle routing problem and related algorithms for logistics distribution: A literature review and classification. *Operational research*, 22(3):2033–2062, 2022.

[5] Giorgio Ausiello, Pierluigi Crescenzi, Giorgio Gambosi, Viggo Kann, Alberto Marchetti-Spaccamela, and Marco Protasi. *Complexity and approximation: Combinatorial optimization problems and their approximability properties*. Springer Science & Business Media, 2012.

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

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

[8] 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.

[9] Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. Pointer networks. In *Advances in Neural Information Processing Systems*, volume 28, 2015.

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

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

[12] 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, pages 21188–21198, 2020.

[13] 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*, 2023.

[14] 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*, 2023.

[15] Yan Jin, Yuandong Ding, Xuanhao Pan, Kun He, Li Zhao, Tao Qin, Lei Song, and Jiang Bian. Pointerformer: deep reinforced multi-pointer transformer for the traveling salesman problem. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2023.

[16] Jianan Zhou, Yaoxin Wu, Wen Song, Zhiguang Cao, and Jie Zhang. Towards omni-generalizable neural methods for vehicle routing problems. In *International Conference on Machine Learning*, pages 42769–42789. PMLR, 2023.

[17] Yang Wang, Ya-Hui Jia, Wei-Neng Chen, and Yi Mei. Distance-aware attention reshaping: Enhance generalization of neural solver for large-scale vehicle routing problems. *arXiv preprint arXiv:2401.06979*, 2024.

[18] Changliang Zhou, Xi Lin, Zhenkun Wang, Xialiang Tong, Mingxuan Yuan, and Qingfu Zhang. Instance-conditioned adaptation for large-scale generalization of neural combinatorial optimization. *arXiv preprint arXiv:2405.01906*, 2024.

[19] 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. URL https://openreview.net/forum?id=TbTJJNjumY.

[20] Yuanyao Chen, Rongsheng Chen, Fu Luo, and Zhenkun Wang. Improving generalization of neural combinatorial optimization for vehicle routing problems via test-time projection learning. In *Advances in Neural Information Processing Systems*, 2025.

[21] 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 *Advances in Neural Information Processing Systems*, 2025.

[22] 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*, February 2023.

[23] 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.

[24] 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. URL https://openreview.net/forum?id=6ZajpxqTlQ.

[25] 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*, 2024.

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

[27] 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 *International Joint Conference on Artificial Intelligence*, 2024.

[28] 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*, 2024.

[29] 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.

[30] 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.

[31] Ruizhong Qiu, Zhiqing Sun, and Yiming Yang. DIMES: A differentiable meta solver for combinatorial optimization problems. In *Advances in Neural Information Processing Systems*, 2022.

[32] Zhiqing Sun and Yiming Yang. DIFUSCO: Graph-based diffusion solvers for combinatorial optimization. In *Advances in Neural Information Processing Systems*, 2023.

[33] Yang Li, Jinpei Guo, Runzhong Wang, and Junchi Yan. From distribution learning in training to gradient search in testing for combinatorial optimization. In *Advances in Neural Information Processing Systems*, 2023.

[34] 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*, 2024. URL https://openreview.net/forum?id=xDrKZOZEOc.

[35] 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*, 2025. URL https://github.com/Thinklab-SJTU/GenSCO.

[36] Yining Ma, Zhiguang Cao, and Yeow Meng Chee. Learning to search feasible and infeasible regions of routing problems with flexible neural k-opt. In *Advances in Neural Information Processing Systems*, volume 36, 2023.

[37] André Hottung, Paula Wong-Chung, and Kevin Tierney. Neural deconstruction search for vehicle routing problems. *Transactions on Machine Learning Research*, 2025. ISSN 2835-8856. URL https://openreview.net/forum?id=bCmEP1Ltwq.

[38] 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. In *Advances in Neural Information Processing Systems*, volume 34, pages 7472–7483, 2021.

[39] Yifan Xia, Xianliang Yang, Zichuan Liu, Zhihao Liu, Lei Song, and Jiang Bian. Position: Rethinking post-hoc search-based neural approaches for solving large-scale traveling salesman problems. In *International Conference on Machine Learning*, 2024. URL https://openreview.net/forum?id=cEJ9jNJuJP.

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

[41] Fu Luo, Xi Lin, Zhenkun Wang, Tong Xialiang, Mingxuan Yuan, and Qingfu Zhang. Self-improved learning for scalable neural combinatorial optimization. *arXiv preprint arXiv:2403.19561*, 2024.

[42] Jonathan Pirnay and Dominik G. Grimm. Self-improvement for neural combinatorial optimization: Sample without replacement, but improvement. *Transactions on Machine Learning Research*, 2024. ISSN 2835-8856. Featured Certification.

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

[44] Thibaut Vidal, Teodor Gabriel Crainic, Michel Gendreau, Nadia Lahrichi, and Walter Rei. A hybrid genetic algorithm for multidepot and periodic vehicle routing problems. *Operations Research*, 60(3):611–624, 2012.

[45] Gerhard Reinelt. TSPLIB–a traveling salesman problem library. *ORSA Journal on Computing*, 3(4): 376–384, 1991.

[46] André Rohe. VLSI data sets. https://www.math.uwaterloo.ca/tsp/vlsi/index.html, 2014. Forschungsinstitut für Diskrete Mathematik, Universität Bonn.

[47] Robert Bosch and Adrianne Herman. Continuous line drawings via the traveling salesman problem. *Operations Research Letters*, 32(4):302–303, 2004.

[48] 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.

[49] Shipei Zhou, Yuandong Ding, Chi Zhang, Zhiguang Cao, and Yan Jin. DualOpt: A Dual Divide-and-Optimize Algorithm for the Large-scale Traveling Salesman Problem. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2025. URL https://doi.org/10.1609/aaai.v39i25.34926.

[50] Yuxin Pan, Ruohong Liu, Yize Chen, Zhiguang Cao, and Fangzhen Lin. Hierarchical Learning-based Graph Partition for Large-scale Vehicle Routing Problems. In *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems*, 2025. URL https://www.ifaamas.org/Proceedings/aamas2025/pdfs/p1604.pdf.

[51] Sirui Li, Zhongxia Yan, and Cathy Wu. Learning to Delegate for Large-scale Vehicle Routing. In *Advances in Neural Information Processing Systems*, 2021. URL https://arxiv.org/abs/2107.04139.

[52] Wenbin Ouyang, Sirui Li, Yining Ma, and Cathy Wu. Learning to Segment for Vehicle Routing Problems. In *International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=pN261iTKvr.

[53] Shengkai Chen, Zhiguang Cao, Jianan Zhou, Yaoxin Wu, Senthilnath Jayavelu, Zhuoyi Lin, Xiaoli Li, and Shili Xiang. DRAGON: LLM-Driven Decomposition and Reconstruction Agents for Large-Scale Combinatorial Optimization. In *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems*, 2026. URL https://arxiv.org/abs/2601.06502.

[54] Jiongzhi Zheng, Kun He, Jianrong Zhou, Yan Jin, and Chu-Min Li. Combining Reinforcement Learning with Lin-Kernighan-Helsgaun Algorithm for the Traveling Salesman Problem. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2021. URL https://doi.org/10.1609/aaai.v35i14.17476.

[55] Haoran Ye, Jiarui Wang, Zhiguang Cao, Helan Liang, and Yong Li. DeepACO: Neural-enhanced Ant Systems for Combinatorial Optimization. In *Advances in Neural Information Processing Systems*, 2023. URL https://arxiv.org/abs/2309.14032.

[56] Rongjie Zhu, Cong Zhang, and Zhiguang Cao. Refining Hybrid Genetic Search for CVRP via Reinforcement Learning-Finetuned LLM. In *International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=aITKXFeivk.

[57] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. *Advances in neural information processing systems*, 27, 2014.

[58] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

[59] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, volume 30, 2017.

[60] Elias Khalil, Hanjun Dai, Yuyu Zhang, Bistra Dilkina, and Le Song. Learning combinatorial optimization algorithms over graphs. In *Advances in Neural Information Processing Systems*, volume 30, 2017.

[61] Mohammadreza Nazari, Afshin Oroojlooy, Lawrence Snyder, and Martin Takác. Reinforcement learning for solving the vehicle routing problem. In *Advances in Neural Information Processing Systems*, volume 31, 2018.

[62] Michel Deudon, Pierre Cournut, Alexandre Lacoste, Yossiri Adulyasak, and Louis-Martin Rousseau. Learning heuristics for the tsp by policy gradient. In *International conference on the integration of constraint programming, artificial intelligence, and operations research*, pages 170–181. Springer, 2018.

[63] Liang Xin, Wen Song, Zhiguang Cao, and Jie Zhang. Step-wise deep learning models for solving routing problems. *IEEE Transactions on Industrial Informatics*, 17(7):4861–4871, 2020.

[64] Liang Xin, Wen Song, Zhiguang Cao, and Jie Zhang. Multi-decoder attention model with embedding glimpse for solving vehicle routing problems. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 12042–12049, 2021.

[65] Wouter Kool, Herke van Hoof, Joaquim Gromicho, and Max Welling. Deep policy dynamic programming for vehicle routing problems. In *Integration of Constraint Programming, Artificial Intelligence, and Operations Research: 19th International Conference, CPAIOR 2022, Los Angeles, CA, USA, June 20-23, 2022, Proceedings*, pages 190–213. Springer, 2022.

[66] Rui Sun, Zhi Zheng, and Zhenkun Wang. Learning encodings for constructive neural combinatorial optimization needs to regret. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 20803–20811, 2024.

[67] Sahil Manchanda, Sofia Michel, Darko Drakulic, and Jean-Marc Andreoli. On the generalization of neural combinatorial optimization heuristics. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 426–442. Springer, 2022.

[68] Zhang-Hua Fu, Sipeng Sun, Jintong Ren, Tianshu Yu, Haoyu Zhang, Yuanyuan Liu, Lingxiao Huang, Xiang Yan, and Pinyan Lu. A hierarchical destroy and repair approach for solving very large-scale travelling salesman problem. *arXiv preprint arXiv:2308.04639*, 2023.

[69] Yong Liang Goh, Zhiguang Cao, Yining Ma, Jianan Zhou, Mohammed Haroon Dupty, and Wee Sun Lee. Shield: Multi-task multi-distribution vehicle routing solver with sparsity and hierarchy. In *International Conference on Machine Learning*, 2025.

[70] Dian Meng, Zhiguang Cao, Jie Gao, Yaoxin Wu, and Yaqing Hou. Uniteformer: Unifying node and edge modalities in transformers for vehicle routing problems. In *Advances in Neural Information Processing Systems*, 2025. URL https://openreview.net/forum?id=BRklmFlCsD.

[71] Jiwoo Son, Minsu Kim, Hyeonah Kim, and Jinkyoo Park. Meta-sage: Scale meta-learning scheduled adaptation with guided exploration for mitigating scale shift on combinatorial optimization. In *International Conference on Machine Learning*, pages 32194–32210. PMLR, 2023.

[72] Chaitanya K Joshi, Quentin Cappart, Louis-Martin Rousseau, and Thomas Laurent. Learning tsp requires rethinking generalization. In *27th International Conference on Principles and Practice of Constraint Programming (CP 2021)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2021.

[73] Chaitanya K Joshi, Quentin Cappart, Louis-Martin Rousseau, and Thomas Laurent. Learning the travelling salesperson problem requires rethinking generalization. *Constraints*, 27(1-2):70–98, 2022.

[74] 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.

[75] Dongxiang Zhang, Ziyang Xiao, Yuan Wang, Mingli Song, and Gang Chen. Neural tsp solver with progressive distillation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, pages 12147–12154, 2023.

[76] Fu Luo, Yaoxin Wu, Zhi Zheng, and Zhenkun Wang. Rethinking neural combinatorial optimization for vehicle routing problems with different constraint tightness degrees, 2025. URL `https://arxiv.org/abs/2505.24627`.

[77] Shen Lin and Brian W Kernighan. An effective heuristic algorithm for the traveling-salesman problem. *Operations research*, 21(2):498–516, 1973.

[78] Yining Ma, Zhiguang Cao, and Yeow Meng Chee. Learning to search feasible and infeasible regions of routing problems with flexible neural k-opt. In *Advances in Neural Information Processing Systems*, volume 36, 2024.

[79] 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*, 2021.

[80] André Hottung, Bhanu Bhandari, and Kevin Tierney. Learning a latent search space for routing problems using variational autoencoders. In *International Conference on Learning Representations*, 2021.

[81] André Hottung and Kevin Tierney. Neural large neighborhood search for the capacitated vehicle routing problem. In *European Conference on Artificial Intelligence*, 2020.

[82] Arthur Delarue, Ross Anderson, and Christian Tjandraatmadja. Reinforcement learning with combinatorial actions: An application to vehicle routing. *Advances in Neural Information Processing Systems*, 33: 609–620, 2020.

[83] Jinho Choo, Yeong-Dae Kwon, Jihoon Kim, Jeongwoo Jae, André Hottung, Kevin Tierney, and Youngjune Gwon. Simulation-guided beam search for neural combinatorial optimization. In *Advances in Neural Information Processing Systems*, volume 35, pages 8760–8772, 2022.

[84] Changliang Zhou, Xi Lin, Zhenkun Wang, and Qingfu Zhang. L2r: Learning to reduce search space for generalizable neural routing solver, 2025. URL `https://arxiv.org/abs/2503.03137`.

[85] Changliang Zhou, Xi Lin, Zhenkun Wang, and Qingfu Zhang. Learning to reduce search space for generalizable neural routing solver. *arXiv preprint arXiv:2503.03137*, 2025.

[86] Changliang Zhou, Canhong Yu, Shunyu Yao, Xi Lin, Zhenkun Wang, Yu Zhou, and Qingfu Zhang. Urs: A unified neural routing solver for cross-problem zero-shot generalization. *arXiv preprint arXiv:2509.23413*, 2025.

[87] Kai Li, Fei Liu, Zhenkun Wang, Xialiang Tong, Xiongwei Han, Mingxuan Yuan, and Qingfu Zhang. Ars: Automatic routing solver with large language models. *arXiv preprint arXiv:2502.15359*, 2025.

[88] Quentin Cappart, Thierry Moisan, Louis-Martin Rousseau, Isabeau Prémont-Schwarz, and Andre A Cire. Combining reinforcement learning and constraint programming for combinatorial optimization. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2021.

[89] David Applegate, Robert Bixby, Vašek Chvátal, and William Cook. Concorde tsp solver, 2006.

[90] Éric D Taillard and Keld Helsgaun. Popmusic for the travelling salesman problem. *European Journal of Operational Research*, 272(2):420–429, 2019.

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 state the closed-loop LSSL framework, its sparse scaling motivation, and its empirical scope, and these claims are matched by the method and experiment sections(Section 2 and 3).

   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: The paper discusses the limitations of the work in Appendix K.

   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 and model settings are in Section 3 and Appendix E, F, H and I. We will make our source code publicly available upon acceptance.

   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: We will release our code and data once the paper is accepted.

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 and model settings are in Section 3 and Appendix E, F, H and I. We will make our source code publicly available upon acceptance.

   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: In neural combinatorial optimization, we usually adopt the optimality gap and inference time as metrics to measure performance. Both of them are deterministic.

   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: We provide the resource requirements for training and dataset self-bootstrapping in the Appendix. E.

   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: Our research conducted in the paper conforms 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 broader impacts in Appendix K.

    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: [N/A]

    Justification: This paper has 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: We have properly cited the assets (e.g., code, data, models) in section L.

    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. Upon acceptance, we will make the source code and LSSL-bootstrapped datasets publicly available, accompanied by comprehensive documentation.

    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 methodological development of this work does not rely on LLMs as essential, 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.

A Additional controlled evaluation

We supplement the original main-table protocols with the following evaluations reported during the review discussion. The tables retain the reported values and protocol boundaries; differences across experiments should not be interpreted as reruns of a single universal setting.

A.1 Serial and batched end-to-end evaluation

Serial-1 processes one active instance at a time. Batched evaluation uses the largest feasible batching under 24 GiB GPU memory and 16 CPU cores, without an additional cap on neural batch size. Both columns measure the wall-clock time to solve the complete test split, including preprocessing, neural inference, and all backend calls. GLOP uses batch size one for CVRP. Mean length and mean gap in Tables 6–12 are the arithmetic means of the Serial-1 and batched quality results, as reported in the review discussion, rather than per-protocol quality statistics. Rounded aggregate lengths do not suffice to reconstruct a mean per-instance relative gap. A dash denotes an unavailable measurement.

Table 6: End-to-end TSP100 evaluation. Quality averages the two protocols; times cover the full split. B_t is seconds per cycle.

Method	Mean length	Mean gap (%)	Serial-1	Batched
LKH	7.763	0.000	56m	3.5m
LSSL $C = 1$, $B_{\text{trials}} = 128$	7.763	0.000	13.2m	1.3m
GenSCO $C = 10$	7.763	0.0006	1.3m	50.3s
GenSCO $C = 160$	7.763	0.0003	8m	3.1m
GLOP	7.767	0.050	35.6m	30.8m
L2C-Insert greedy	7.799	0.454	52.2s	42s
L2C-Insert $I = 1000$	7.763	0.0003	8.7h	6.3h
SIL-PRC1000	7.770	0.090	–	14.4m

Table 7: End-to-end TSP1K evaluation. Quality averages the two protocols; times cover the full split. B_t is seconds per cycle.

Method	Mean length	Mean gap (%)	Serial-1	Batched
LKH	23.119	0.000	8.2h	30.8m
LSSL $C = 1$, $B_{\text{trials}} = 128$	23.122	0.017	2.9m	17.1s
LSSL $C = 2$, $B_{\text{trials}} = 256$	23.120	0.007	6.5m	38.3s
LSSL $C = 4$, $B_{\text{trials}} = 512$	23.119	0.000	36.7m	3.6m
GenSCO $C = 10$	23.143	0.107	3m	2.1m
GenSCO $C = 160$	23.132	0.058	26.4m	22.2m
GLOP	23.997	3.798	4.5m	2.4m
L2C-Insert greedy	24.289	5.061	23.2s	17.9s
L2C-Insert $I = 1000$	23.231	0.484	52.3m	5.8h
DualOpt	23.295	0.765	5.8m	2.2m
SIL-PRC1000	23.210	0.399	2.9h	1.8h

Table 8: End-to-end TSP10K evaluation. Quality averages the two protocols; times cover the full split. B_t is seconds per cycle.

Method	Mean length	Mean gap (%)	Serial-1	Batched
LKH	71.778	0.000	8.9h	33.2m
LSSL $C = 1$, $B_{\text{trials}} = 128$	71.789	0.015	4.9m	32.2s
LSSL $C = 2$, $B_{\text{trials}} = 256$	71.783	0.007	10.1m	1.1m
LSSL $C = 4$, $B_{\text{trials}} = 512$	71.773	-0.006	30.3m	3.3m
GLOP	75.792	5.609	1.6m	1.6m
L2C-Insert greedy	78.131	8.852	1.3m	56.4s
L2C-Insert $I = 1000$	73.347	2.185	17.3m	1.8h
DualOpt	72.318	0.784	5.1m	5m
SIL-PRC1000	73.091	1.862	3.7h	2.3h

Table 9: End-to-end TSP100K evaluation. Quality averages the two protocols; times cover the full split. B_t is seconds per cycle.

Method	Mean length	Mean gap (%)	Serial-1	Batched
LKH	225.997	0.000	400h	25h
LSSL $C = 1$, $B_{\text{trials}} = 128$	226.219	0.098	2.7h	16.9m
LSSL $C = 2$, $B_{\text{trials}} = 512$	226.028	0.014	18.1h	1.9h
LSSL $C = 4$, $B_{\text{trials}} = 1024$	225.997	0.000	64.7h	6.9h
DualOpt	229.784	1.680	2.4h	2.5h
SIL-PRC1000	232.054	2.680	9.8h	5.8h

Table 10: End-to-end CVRP100 evaluation. Quality averages the two protocols; times cover the full split. B_t is seconds per cycle.

Method	Mean length	Mean gap (%)	Serial-1	Batched
HGS	15.592	0.000	16.8h	1.1h
LSSL $C = 1$, $B_t = 4$	15.591	-0.009	1.1h	4.2m
LSSL $C = 2$, $B_t = 32$	15.589	-0.021	17.8h	1.1h
GLOP	19.341	24.257	39.6m	39.6m
L2C-Insert greedy	16.182	3.785	1.5m	1.2m
L2C-Insert $I = 1000$	15.630	0.434	11.8h	9h
HLGP	15.902	1.919	3.3h	8.7m
Learning to Delegate	15.705	0.656	8.4h	51.5m
SIL-PRC1000	20.459	31.128	52.7m	35.1m

Table 11: End-to-end CVRP1K evaluation. Quality averages the two protocols; times cover the full split. B_t is seconds per cycle.

Method	Mean length	Mean gap (%)	Serial-1	Batched
HGS	41.212	0.000	56h	3.5h
LSSL $C = 1$, $B_t = 4$	41.681	1.138	6.7m	25s
LSSL $C = 2$, $B_t = 32$	41.328	0.281	1.8h	6.7m
LSSL $C = 4$, $B_t = 128$	41.186	-0.063	14.2h	53.3m
GLOP	47.208	14.695	3.2m	3.2m
L2C-Insert greedy	44.696	8.454	1.4m	1m
L2C-Insert $I = 1000$	43.413	5.470	2.3h	1.9h
HLGP	42.081	2.233	26.3m	10.2m
Learning to Delegate	41.724	1.367	8.5h	25.3m
SIL-PRC1000	42.638	3.588	3.4h	2.1h

Table 12: End-to-end CVRP10K evaluation. Quality averages the two protocols; times cover the full split. B_t is seconds per cycle.

Method	Mean length	Mean gap (%)	Serial-1	Batched
HGS	226.177	0.000	16h	1h
LSSL $C = 1$, $B_t = 4$	228.320	0.947	1.3m	4.7s
LSSL $C = 2$, $B_t = 32$	224.690	-0.657	17.1m	1.1m
LSSL $C = 4$, $B_t = 128$	222.490	-1.630	2.3h	8.5m
GLOP	263.970	16.687	1.9m	1.9m
L2C-Insert greedy	256.261	13.301	3.3m	2.6m
L2C-Insert $I = 1000$	247.179	9.286	36.9m	30.8m
HLGP	224.552	-0.714	50.6m	31.8m
Learning to Delegate	227.662	0.662	4.8h	41.1m
SIL-PRC1000	228.848	1.181	4.1h	2.5h

A.2 Backend-access and learning-enhanced search controls

To distinguish the benefit of the closed loop from access to a strong backend, neural constructors receive LKH/HGS post-processing with the corresponding LSSL backend budget (Table 13). Arrows denote quality before and after post-processing; sequential runtime includes both components. This control differs from the arithmetic-mean protocol table above. We also compare learning-enhanced classical solvers (Table 14), retaining each method's native search schedule because trials and candidate widths are not equivalent across backends. † explicitly marks a run extended to complete the test split; such a row is not a matched-time comparison.

Table 13: Neural construction followed by LKH/HGS. Gaps are percentages; time is neural cost plus backend cost for the full split.

Task	Neural method	Before	After	Time
TSP100	L2C-Insert $I = 1000$	0.0003	0.0003	8.7h + 13.2m
TSP100	Fast-T2T	0.032	0.013	23.4m + 13.2m
TSP100	GenSCO $C = 10$	0.0006	0.0003	1.3m + 13.2m
TSP100	GenSCO $C = 160$	0.0003	0.0002	8m + 13.2m
TSP1K	L2C-Insert $I = 1000$	0.476	0.127	52.3m + 6.5m
TSP1K	Fast-T2T	0.589	0.104	20.5m + 6.5m
TSP1K	GenSCO $C = 10$	0.106	0.046	3m + 6.5m
TSP1K	GenSCO $C = 160$	0.056	0.028	26.4m + 6.5m
TSP10K	L2C-Insert $I = 1000$	2.052	0.234	17.3m + 10.1m
CVRP100	L2C-Insert $I = 1000$	0.370	-0.014	13.3h + 17.8h
CVRP1K	L2C-Insert $I = 1000$	4.739	0.982	4.5h + 1.8h
CVRP10K	L2C-Insert $I = 1000$	8.716	2.855	50.3m + 17.1m

Table 14: Learning-enhanced classical search. Reported mean gaps and complete-split times; † denotes an extended, unmatched runtime.

Task	Method	Gap (%)	Time
TSP100	LSSL $C = 1$, trials128	0.0000	13.16m
TSP100	NeuroLKH	0.0000	13.16m
TSP100	VSR-LKH	0.002	13.02m
TSP100	DeepACO-NLS	0.271	201.42m†
TSP1K	LSSL $C = 2$, trials256	0.007	6.50m
TSP1K	NeuroLKH	0.043	6.50m
TSP1K	VSR-LKH	0.013	6.92m†
TSP1K	DeepACO-NLS	3.903	363.51m†
TSP10K	LSSL $C = 2$, trials256	0.007	10.09m
TSP10K	NeuroLKH	0.164	10.09m
TSP10K	VSR-LKH	0.041	137.76m†
CVRP100	LSSL $C = 1$, $B_t = 4$	-0.009	1.1h
CVRP100	NeuroLKH	1.075	1.1h
CVRP100	RFTHGS	1.407	1.1h
CVRP100	DeepACO-NLS	3.622	50.21m
CVRP1K	LSSL $C = 2$, $B_t = 32$	0.281	1.8h
CVRP1K	NeuroLKH	3.373	1.8h
CVRP1K	RFTHGS	5.621	1.8h
CVRP1K	Learning to Delegate	9.858	1.8h
CVRP1K	DeepACO-NLS	9.817	215.79m†
CVRP10K	LSSL $C = 2$, $B_t = 32$	-0.657	17.1m
CVRP10K	NeuroLKH	49.280	105.66m†
CVRP10K	RFTHGS	1.416	17.1m
CVRP10K	Learning to Delegate	3.419	17.1m

A.3 CVRP100K scalability stress test

Dense HGS distance and auxiliary-state storage originally prevented CVRP100K evaluation. For this stress test, distances are computed on demand with a bounded cache, and SWAP*/Split states are

stored or recomputed sparsely. These are memory-only changes: the objective, feasibility rules, search operators, stopping criterion, budget, randomization, and parallelization are unchanged. Table 15 uses Serial-1 and reports complete-split time. It evaluates end-to-end scalability, without claiming a CVRP100K training dataset or a certified optimum.

Table 15: CVRP100K under Serial-1. B_t is the per-cycle time budget in seconds.

Method	Length	Gap (%)	Time
HGS	2088.243	0.000	80.6h
LSSL $C = 1$, $B_t = 256$	2100.922	0.607	1.1h
LSSL $C = 2$, $B_t = 512$	2063.166	-1.201	4.55h
L2C-Insert greedy	2273.172	8.856	38.3m
L2C-Insert $I = 1000$	2260.411	8.245	1.7h

A.4 Decomposition as a complementary component

Decomposition can preserve useful structure and is complementary to LSSL. The L2Seg-SYN-L2D row in Table 16 is paper-reported, because the L2Seg code was unavailable at the time of the rebuttal. It uses different hardware and is not a matched reproduction; its runtime must not be used to claim a controlled speedup. The UDC-style decomposition plus Guided-HGS rows are a proof of concept. On the reported CVRP1K comparison, L2Seg provides better quality than plain LSSL at approximately 150 s, while the decomposition–LSSL hybrid yields the lowest reported gap.

Table 16: CVRP1K decomposition comparison. This table alone reports seconds per instance; the L2Seg row uses external hardware.

Method	Length	Gap (%)	s/instance
L2Seg-SYN-L2D (paper-reported)	41.230	0.0437	150
LSSL $C = 2$, $B_t = 32$	41.328	0.2815	64.8
LSSL $C = 2$, $B_t = 75$	41.278	0.1590	150.2
LSSL $C = 4$, $B_t = 128$	41.186	-0.0631	511.2
LSSL + UDC-style $C = 1$, $B_t = 90$	41.203	-0.0213	90
LSSL + UDC-style $C = 2$, $B_t = 75$	41.152	-0.1460	150

B Additional controlled ablations

All entries below are reported mean gaps, without confidence intervals or a statistical-significance claim. Each experiment preserves its stated budget, data, and reference protocol. Table 17 isolates per-cycle prior refresh at matched cycle count, Table 18 crosses target construction with cycle count, Table 19 sweeps the candidate width K on TSP10K, and Table 20 ablates the CVRP node loss and node guidance.

Table 17: Matched-cycle refresh control at 32 s. The two $C = 2$ arms match restart/reseed behavior, budget splits, seeds, warm starts, and backend settings.

Guidance	Cycles	TSP1K gap (%)	CVRP1K gap (%)
Frozen, continuous	1	0.00034	0.6945
Frozen, matched restarts	2	0.00037	0.6877
Refreshed	2	0.00024	0.5942

Table 18: Target construction $\times$ cycle count at 32 s. Each cell is TSP1K / CVRP1K mean gap (%). The two bootstrapped targets use the same data, isolating single-solution versus multi-solution supervision.

Target	$C = 1$	$C = 2$	$C = 4$	$C = 8$
Direct solver $\mathcal{H}$	0.00074 / 0.98	0.00052 / 0.84	0.00041 / 0.85	0.00067 / 1.04
Bootstrap single $\mathbf{x}$	0.00066 / 0.97	0.00047 / 0.83	0.00037 / 0.84	0.00060 / 1.03
Bootstrap multi $\mathcal{H}$	0.00034 / 0.69	0.00024 / 0.59	0.00019 / 0.60	0.00031 / 0.73

Table 19: TSP10K candidate-width sweep with eight denoising steps. Times are the reported neural inference and training measurements in milliseconds, not backend wall-clock times. Gaps retain this experiment's reference convention.

K	Length	Gap (%)	Inference (ms)	Training (ms)
100	71.771	0.0231	91.830	103.840
500	71.768	0.0194	268.415	361.685
1000	71.767	0.0171	506.975	726.695
2000	71.767	0.0171	959.710	1391.425

Table 20: CVRP1K node-loss versus inference-guidance ablation on the same 100 instances. Entries are mean gaps (%). Joint trains edge and node losses; zero retrains with $\lambda_{\text{node}} = 0$. The zero-loss node head is unsupervised.

HGS budget	Joint: edge+node	Joint: edge only	Zero: edge+node	Zero: edge only
8s	1.160	1.298	1.340	1.207
32s	0.666	0.759	0.790	0.699
64s	0.510	0.583	0.610	0.535
128s	0.387	0.444	0.480	0.406

The full joint model with both guidance channels is best at every checkpoint. Removing learned node guidance from that checkpoint worsens quality, particularly early in search. Retraining with zero node loss improves the edge-only variant relative to disabling the node head of a jointly trained checkpoint, while deploying its unsupervised node head worsens quality. Consequently, the deployment benefit of trained node guidance is distinct from a claim that auxiliary supervision necessarily improves the shared edge representation.

C Related work

Neural combinatorial optimization (NCO) for VRPs differs along three axes that matter for LSSL: how solutions or scores are produced, how computation scales to large N, and how the learned model couples with classical search. We organise prior work along these axes; Table 21 contrasts LSSL with the most closely related families.

Recent complementary approaches include DualOpt[49], HLGP[50], and Learning to Delegate[51], which optimize decomposed subproblems or their selection. L2Seg[52] preserves stable solution segments to avoid redundant search; DRAGON[53] uses LLM-driven decomposition and reconstruction with feedback. Their mechanisms differ from LSSL's instance-level sparse heatmap reconditioning and multi-solution training target. Learning-enhanced classical baselines include VSR-LKH[54], DeepACO[55], and RFTHGS[56]; Appendix A reports explicit runtime protocols rather than treating access to classical search as unique to LSSL.

C.1 Scalable neural construction

Constructive NCO, building on the sequence-to-sequence [57] and attention [58, 59] lineage adapted to combinatorial outputs, autoregressively appends or inserts nodes to produce a feasible route. Pointer Networks [9], REINFORCE-trained policies [10, 60–62], the Attention Model [11], stepwise constructive variants [63, 64], POMO [12], and policy dynamic-programming hybrids [65] establish the family; BQ-NCO [14], LEHD [13], Pointerformer [15], regret-aware encodings [66], Boosting NCO [19] and L2C-Insert [21] strengthen decoding or scale supervised training, while Manchanda et al. [67] characterise the out-of-distribution failure modes that motivate scale-aware training. Three strategies extend this regime to ultra-large instances. Decomposition splits an instance into subproblems with a learned partition or revision stage [22, 25, 26, 23, 24, 68]. Cross-scale generalisation keeps a single end-to-end model and strengthens the inductive bias needed for scale transfer [16, 17, 27, 28, 69–71]. Linear-complexity construction amortises a constructive solver to 100K nodes [19, 21]. Across this family the model parameterises the policy or its decoded route directly; classical search, when present, runs once downstream, so search-discovered global structure has no shared interface to feed back into the predictor, and the supervisory signal collapses any plurality of high-quality solutions into a single argmax target.

C.2 Heatmap, diffusion, and generative NCO

Non-constructive NCO predicts edge heatmaps or tour distributions and decodes a feasible solution post hoc. Early supervised GNN heatmaps decoded by beam search or 2-opt [29, 72, 73], learned 2-opt operators [74], Att-GCN+MCTS [30], and DIMES [31] initiated the line, with progressive-distillation [75] variants extending the recipe to harder instances. Diffusion- and flow-based variants are now dominant: DIFUSCO [32] casts edge selection as graph diffusion, T2T [33] adds objective-guided test-time gradient search, Fast-T2T [34] compresses the trajectory via optimisation consistency, and GenSCO [35] reframes generation as a search operator alternating disruption with rectified-flow regeneration. Constraint-tightness analyses argue that single-shot heatmap decoders break when feasibility manifolds are stiff [76], and Xia et al. [39] question whether post-hoc search alone scales to very large TSPs. This is LSSL's closest neural precursor (scoring and feasibility recovery are already decoupled), but the time-conditioned state is a perturbed solution variable trained to converge to a single reference $\mathbf{x}^\star$. Downstream search refines a sample without being looped back as a persistent state, and training never aggregates multiple search outputs into a single supervisory target.

C.3 Neural-guided search and one-sided learning–search loops

A third line couples learning with classical search but closes only one side of the loop. The strongest non-neural baselines remain Lin–Kernighan [77, 6] and HGS [7, 43]. NeuroLKH [38] is the most direct precedent: a sparse GNN predicts candidate edges and node penalties consumed once by LKH. Improvement-operator methods learn a policy that edits an existing tour [36, 78, 79, 37] or, in latent-action variants, sample edits from a learned search space [80]; large-neighbourhood search is similarly amenable to learning the destroy-and-repair operator [81, 82]. Search amplifiers such as EAS [40] and SGBS [83] operate inside a constructive policy. On the training side, self-improvement pipelines feed iteratively improved solutions back as relabelled data [42, 41, 19]. On the inference

side, Test-Time Projection Learning [20], instance-conditioned adaptation [18], and search-space-reduction methods [84–87] adapt the predictor or prune the search space at test time. Generic learning–search frameworks [8, 88] provide broader context. Each of these mechanisms closes at most one side of the learning–search loop (training-time relabelling or inference-time adaptation), but not both, and the interface is typically tied to a specific backend rather than the backend-agnostic $N \times K$ heatmap that LSSL exchanges.

C.4 Positioning summary

LSSL differs from the families above on the axes summarised in Table 21. The exchanged object is a sparse, not-necessarily-feasible structural prior $\mathcal{H}_t$ on an $O(NK)$ candidate graph rather than a route, a perturbed solution variable, or a one-shot LKH candidate set. The same heatmap interface is invoked in multiple inference cycles and aggregates row-wise soft targets from multiple backend-refined route sets at training, closing both sides of the loop with a backend-agnostic design instantiated here with LKH for TSP and HGS for CVRP.

Table 21: Comparison of LSSL with closely related method families along the main axes of learning–search coupling and scalability. partial indicates the property holds at moderate scale but not at the family's largest tractable N under a 24 GB GPU budget.

Family (representatives)	Learned object	Search backend	Backend cycles	Search→training	Sparse $O(NK)$	Max N on 24 GB GPU
NeuroLKH [38]	candidate edges / π for LKH	LKH	×	×	partial	5K
Diffusion / generative NCO [32, 34, 35]	perturbed solution $\mathbf{x}_t$	2-opt / MCTS (post hoc)	×	×	partial	1K
Self-improvement [19, 42, 41]	constructed routes	none	×	✓ (relabel)	×	100K
Test-time / search-space reduction [20, 84, 18]	adapted projection / pruning	none	partial	×	partial	100K
Improvement operators [36, 37, 40, 83]	local repair policy	own learned	×	×	×	1K
LSSL (ours)	**infeasible structural prior $\mathcal{H}_t$**	**backend-agnostic (LKH, HGS used)**	✓	✓ (multi-soln soft)	✓	**100K**

D Supplementary experiments

D.1 Real-world TSP results on TSPLIB

Table 22 reports supplementary real-world TSPLIB TSP results. The DIFUSCO and Fast T2T baseline columns are transcribed from Tables 7 and 8 of [35]; the added L2C-Insert column comes from our local evaluation results, while the NeuroLKH column is transcribed from the supplementary TSPLIB tables of NeuroLKH [38].

Table 22: Results on real-world TSPLIB TSP instances with 50–1K nodes.

Instances	DIFUSCO	Fast T2T	GenSCO	L2C-Insert	NeuroLKH	LSSL Gap (Time) (C=1, B_{trials}=128)	LSSL Gap (Time) (C=2, B_{trials}=256)
eil51	2.82%	0.00%	0.000%	0.000%	0.000%	0.000% (0.0s)	0.000% (0.2s)
berlin52	0.00%	0.00%	0.000%	1.584%	0.000%	0.000% (0.0s)	0.000% (0.0s)
st70	0.00%	0.00%	0.000%	0.000%	0.000%	0.000% (0.0s)	0.000% (0.2s)
eil76	0.34%	0.00%	0.000%	0.004%	0.000%	0.000% (0.1s)	0.000% (0.2s)
pr76	1.12%	0.00%	0.000%	0.000%	0.000%	0.000% (0.1s)	0.000% (0.4s)
rat99	0.09%	0.00%	0.000%	12.416%	0.000%	0.000% (0.1s)	0.000% (0.2s)
kroA100	0.10%	0.00%	0.000%	5.013%	0.000%	0.000% (0.1s)	0.000% (0.3s)
kroB100	2.29%	0.65%	0.000%	5.300%	0.000%	0.000% (0.2s)	0.000% (0.6s)
kroC100	0.00%	0.00%	0.000%	6.076%	0.000%	0.000% (0.1s)	0.000% (0.3s)
kroD100	0.07%	0.00%	0.000%	4.815%	0.000%	0.000% (0.1s)	0.000% (0.3s)
kroE100	3.83%	0.00%	0.000%	4.050%	0.000%	0.000% (0.1s)	0.000% (0.5s)
rd100	0.08%	0.00%	0.000%	0.000%	0.000%	0.000% (0.1s)	0.000% (0.3s)
eil101	0.03%	0.00%	0.000%	0.000%	0.000%	0.000% (0.0s)	0.000% (0.1s)
lin105	0.00%	0.00%	0.000%	4.337%	0.000%	0.000% (0.1s)	0.000% (0.2s)
pr107	0.91%	0.62%	0.000%	0.000%	0.000%	0.000% (0.2s)	0.000% (0.7s)
pr124	1.02%	0.08%	0.000%	0.000%	0.000%	0.000% (0.3s)	0.000% (1.0s)
bier127	0.94%	1.50%	0.000%	0.000%	0.016%	0.000% (0.1s)	0.000% (0.3s)
ch130	0.29%	0.00%	0.000%	0.000%	0.000%	0.000% (0.2s)	0.000% (0.6s)
pr136	0.19%	0.01%	0.000%	0.000%	0.000%	0.000% (0.5s)	0.000% (1.9s)
pr144	0.80%	0.39%	0.000%	0.000%	0.081%	0.089% (0.7s)	0.000% (2.0s)
ch150	0.57%	0.00%	0.000%	0.000%	0.000%	0.000% (0.4s)	0.000% (1.5s)
kroA150	0.34%	0.00%	0.000%	3.220%	0.000%	0.000% (0.3s)	0.000% (1.1s)
kroB150	0.30%	0.07%	0.340%	3.846%	0.000%	0.000% (0.5s)	0.000% (1.8s)
pr152	1.69%	0.19%	0.187%	0.005%	0.037%	0.000% (1.0s)	0.000% (3.4s)
u159	0.82%	0.00%	0.000%	0.025%	0.000%	0.000% (0.1s)	0.000% (0.2s)
rat195	1.48%	0.79%	0.194%	4.828%	0.000%	0.000% (0.4s)	0.000% (1.4s)
d198	3.32%	0.86%	0.751%	1.835%	0.285%	0.000% (3.6s)	0.000% (13.2s)
kroA200	2.28%	0.49%	0.160%	1.025%	0.000%	0.000% (0.3s)	0.000% (1.2s)
kroB200	2.35%	2.50%	0.098%	0.219%	0.000%	0.000% (0.2s)	0.000% (0.8s)
ts225	4.95%	1.37%	0.517%	0.000%	0.000%	0.000% (0.4s)	0.000% (1.7s)
tsp225	3.25%	0.81%	0.000%	2.860%	0.000%	0.000% (0.2s)	0.000% (0.6s)
pr226	4.22%	0.34%	0.211%	0.290%	0.016%	0.000% (0.4s)	0.000% (1.8s)
gil262	2.18%	0.18%	0.000%	0.089%	0.000%	0.000% (0.6s)	0.000% (2.0s)
pr264	0.92%	0.73%	0.000%	0.000%	0.000%	0.000% (0.6s)	0.000% (2.0s)
a280	1.39%	0.10%	0.000%	0.109%	0.000%	0.000% (0.4s)	0.000% (1.4s)
pr299	1.46%	1.40%	0.011%	2.390%	0.000%	0.000% (1.6s)	0.000% (4.6s)
lin318	2.95%	1.21%	0.025%	0.038%	0.000%	0.000% (1.1s)	0.000% (4.1s)
rd400	1.18%	0.08%	0.030%	0.162%	0.000%	0.000% (0.8s)	0.000% (2.6s)
fl417	3.30%	2.01%	0.596%	0.408%	0.056%	0.000% (11.8s)	0.000% (44.1s)
pr439	2.73%	0.50%	0.106%	0.387%	0.047%	0.000% (1.4s)	0.000% (6.0s)
pcb442	2.59%	0.61%	0.015%	0.598%	0.000%	0.000% (0.7s)	0.000% (2.7s)
d493	1.81%	1.43%	3.872%	0.469%	0.086%	0.000% (2.8s)	0.000% (10.3s)
u574	2.50%	0.94%	0.109%	0.630%	0.000%	0.000% (1.2s)	0.000% (4.4s)
rat575	2.32%	1.43%	0.000%	0.825%	0.001%	0.000% (1.1s)	0.000% (3.9s)
p654	7.49%	1.67%	1.023%	1.815%	0.354%	0.000% (19.6s)	0.000% (1.23m)
d657	4.86%	0.64%	0.457%	0.751%	0.001%	0.000% (1.5s)	0.000% (5.7s)
u724	2.05%	1.41%	0.040%	0.555%	0.000%	0.000% (1.4s)	0.000% (7.9s)
rat783	3.04%	1.03%	0.068%	0.995%	0.000%	0.000% (1.2s)	0.000% (3.5s)
Mean	1.735%	0.542%	0.184%	1.499%	0.020%	0.002% (1.2s)	0.000% (4.5s)

D.2 Real-world TSP results on VLSI

Tables 23 and 24 report supplementary real-world VLSI TSP results, both transcribed from the corresponding real-world result summary file. The former covers instances with fewer than 2K nodes and reports GenSCO, L2C-Insert, NeuroLKH, BQ bs16, UDC-$\mathbf{x}_{250}$ (α=50), GLOP, LSSL Gap, and LSSL Time. The latter covers instances with 2K–200K nodes and reports L2C-Insert, LSSL. Figures 6 and 7 visualize LSSL tours on `djb2036` and `bck2217`.

Table 23: Results on real-world VLSI TSP instances with fewer than 2K nodes.

Instance	GenSCO	L2C-Insert	NeuroLKH	BQ bs16	UDC-$\mathbf{x}_{250}$	GLOP	LSSL Gap (Time) (C=1, B_{trials}=128)	LSSL Gap (Time) (C=2, B_{trials}=256)
xqf131	-0.138%	4.765%	−0.061%	-0.138%	-0.138%	1.494%	−0.138% (0.3s)	−0.138% (1.1s)
xqg237	0.043%	1.289%	−0.029%	-0.129%	-0.013%	1.783%	−0.183% (0.9s)	−0.183% (2.8s)
pma343	0.611%	15.229%	0.045%	2.934%	5.424%	2.167%	−0.175% (6.4s)	−0.175% (23.3s)
pka379	1.771%	19.875%	−0.039%	4.344%	5.434%	2.703%	−0.191% (5.8s)	−0.191% (21.2s)
bcl380	-0.193%	-0.134%	−0.006%	1.148%	3.840%	2.289%	−0.193% (1.6s)	−0.193% (2.5s)
pbl395	-0.364%	-0.356%	−0.141%	1.166%	2.286%	3.100%	−0.364% (0.7s)	−0.364% (3.0s)
pbk411	-0.460%	-0.376%	−0.365%	0.155%	0.754%	1.792%	−0.469% (1.5s)	−0.469% (5.4s)
pbn423	-0.383%	-0.340%	−0.030%	-0.023%	1.433%	2.977%	−0.394% (0.9s)	−0.394% (4.8s)
pbm436	-0.364%	0.198%	−0.045%	1.629%	3.754%	2.019%	−0.364% (2.9s)	−0.364% (6.9s)
xql662	-0.113%	0.028%	−0.097%	2.216%	2.471%	5.447%	−0.185% (4.2s)	−0.185% (15.6s)
rbx711	-0.129%	0.948%	−0.041%	2.346%	2.758%	5.247%	−0.174% (1.7s)	−0.174% (5.3s)
rbu737	0.223%	0.763%	0.034%	2.673%	1.466%	3.149%	−0.131% (1.9s)	−0.131% (6.7s)
dkg813	0.819%	0.666%	−0.086%	4.328%	6.442%	3.185%	−0.286% (4.0s)	−0.286% (12.8s)
lim963	-0.055%	0.785%	−0.016%	2.068%	2.251%	3.981%	−0.262% (2.7s)	−0.269% (12.6s)
pbd984	-0.225%	0.378%	−0.133%	2.413%	3.202%	4.463%	−0.309% (3.3s)	−0.340% (38.2s)
xit1083	-0.226%	0.674%	−0.005%	20.926%	1.618%	4.469%	−0.250% (5.5s)	−0.250% (19.9s)
dka1376	-0.123%	1.695%	−0.040%	3.455%	5.575%	6.088%	−0.170% (2.3s)	−0.170% (6.8s)
dca1389	0.411%	0.539%	−0.084%	8.239%	6.715%	4.553%	−0.258% (3.7s)	−0.195% (32.5s)
dja1436	0.437%	1.008%	0.045%	9.670%	5.111%	5.448%	−0.147% (5.4s)	−0.153% (15.5s)
icw1483	0.226%	1.245%	−0.077%	7.366%	8.725%	6.291%	−0.396% (3.0s)	−0.417% (9.9s)
fra1488	3.397%	0.637%	−0.161%	9.941%	7.098%	2.633%	−0.404% (3.8s)	−0.404% (12.7s)
rbv1583	0.598%	1.168%	0.013%	6.495%	3.695%	4.109%	−0.205% (11.0s)	−0.205% (25.4s)
rby1599	0.660%	0.967%	−0.094%	7.018%	4.133%	5.691%	−0.284% (5.4s)	−0.314% (14.7s)
fnb1615	10.120%	1.039%	−0.015%	11.194%	4.767%	4.808%	−0.278% (14.9s)	−0.278% (44.6s)
djc1785	2.269%	0.932%	−0.106%	5.945%	3.412%	6.520%	−0.253% (8.1s)	−0.264% (24.2s)
dcc1911	0.761%	1.102%	−0.044%	8.892%	7.471%	5.441%	−0.200% (8.1s)	−0.226% (21.0s)
dkd1973	23.299%	0.653%	−0.078%	42.724%	6.744%	2.949%	−0.479% (15.1s)	−0.479% (45.3s)
Mean	1.588%	2.051%	−0.061%	6.259%	3.942%	3.881%	−0.264% (4.6s)	−0.267% (16.1s)

Table 24: Results on real-world VLSI TSP instances with 2K–200K nodes.

Instance	L2C-Insert	GLOP	LSSL Gap (Time) (C=1, B_{trials}=128)	LSSL Gap (Time) (C=2, B_{trials}=256)	Instance	L2C-Insert	GLOP	LSSL Gap (Time) (C=1, B_{trials}=128)	LSSL Gap (Time) (C=2, B_{trials}=256)
djb2036	0.491%	6.445%	−0.383% (5.7s)	−0.387% (14.0s)	ido21215	3.486%	7.834%	−0.234% (3.75m)	−0.275% (10.20m)
dcb2086	1.069%	7.290%	−0.309% (3.7s)	−0.315% (14.8s)	fma21553	3.786%	8.634%	−0.162% (4.09m)	−0.230% (9.87m)
bva2144	0.854%	6.945%	−0.271% (3.9s)	−0.271% (10.7s)	lsb22777	4.204%	8.580%	−0.279% (4.42m)	−0.313% (12.44m)
xqc2175	1.589%	5.864%	−0.283% (4.9s)	−0.403% (14.7s)	xrh24104	5.330%	8.566%	−0.237% (6.23m)	−0.252% (13.37m)
bck2217	0.745%	6.051%	−0.400% (11.8s)	−0.407% (28.1s)	bbz25234	4.908%	7.387%	−0.314% (6.57m)	−0.320% (13.62m)
xpr2308	1.213%	8.387%	−0.222% (5.7s)	−0.295% (18.5s)	irx28268	4.258%	7.847%	−0.355% (9.64m)	−0.367% (18.09m)
ley2323	1.163%	8.366%	−0.210% (5.6s)	−0.246% (22.0s)	fyg28534	7.719%	10.375%	−0.274% (8.66m)	−0.288% (17.93m)
irw2802	1.547%	8.162%	−0.259% (6.9s)	−0.263% (20.0s)	icx28698	6.638%	9.307%	−0.271% (9.49m)	−0.295% (18.50m)
dbj2924	1.359%	7.232%	−0.125% (7.3s)	−0.125% (21.3s)	boa28924	6.409%	8.829%	−0.306% (9.73m)	−0.313% (20.25m)
pia3056	2.217%	6.826%	−0.378% (13.4s)	−0.383% (34.2s)	ird29514	5.461%	9.085%	−0.249% (8.64m)	−0.259% (21.18m)
xqe3891	1.246%	7.697%	−0.097% (9.6s)	−0.136% (30.5s)	pbh30440	4.929%	8.667%	−0.231% (9.62m)	−0.259% (20.66m)
bgb4355	1.453%	7.045%	−0.279% (13.4s)	−0.299% (40.7s)	xib32892	5.454%	8.935%	−0.231% (10.82m)	−0.246% (24.55m)
xsc6880	2.041%	8.126%	−0.317% (28.0s)	−0.361% (1.25m)	fry33203	7.195%	9.281%	−0.168% (14.87m)	−0.197% (27.18m)
lap7454	3.013%	8.450%	−0.344% (32.8s)	−0.354% (1.39m)	bby34656	6.315%	8.080%	−0.281% (14.18m)	−0.317% (34.10m)
ida8197	2.111%	8.127%	−0.262% (39.2s)	−0.297% (1.71m)	pba38478	6.531%	9.031%	−0.254% (17.54m)	−0.265% (37.21m)
xmc10150	2.449%	8.732%	−0.279% (52.7s)	−0.293% (3.61m)	ics39603	9.478%	9.699%	−0.262% (19.71m)	−0.319% (37.93m)
xvb13584	2.344%	6.654%	−0.300% (1.60m)	−0.307% (4.58m)	rbz43748	7.483%	8.412%	−0.282% (21.69m)	−0.323% (49.80m)
xrb14233	3.248%	9.089%	−0.184% (1.83m)	−0.198% (5.85m)	fht47608	8.644%	8.226%	−0.233% (27.85m)	−0.266% (54.20m)
xia16928	3.752%	7.383%	−0.244% (2.48m)	−0.262% (7.03m)	fna52057	7.334%	8.245%	−0.245% (35.21m)	−0.283% (1.13h)
pjh17845	3.007%	6.974%	−0.307% (2.69m)	−0.330% (7.70m)	bna56769	6.761%	7.740%	−0.298% (37.84m)	−0.320% (1.32h)
frh19289	3.965%	8.007%	−0.233% (3.07m)	−0.259% (10.91m)	dan59296	9.102%	8.465%	−0.229% (41.57m)	−0.263% (1.36h)
fnc19402	3.816%	9.094%	−0.211% (3.17m)	−0.260% (8.29m)	sra104815	6.114%	6.089%	−0.718% (1.18h)	−0.733% (1.67h)
Mean	4.142%	8.051%	−0.273% (9.36m)	−0.299% (18.78m)					

D.3 Real-world TSP results on Art-TSP

Table 25 reports results on the 6 large-scale real-world TSP instances from Art-TSP; Figure 5 visualizes the LSSL tour on `mona-lisa100K`.

Table 25: Results on large-scale real-world TSP instances from Art-TSP.

Instance	Random Insertion	L2C-Insert	GLOP	LSSL Gap (Time) (C=1, B_{trials}=128)	LSSL Gap (Time) (C=2, B_{trials}=256)
mona-lisa100K	6.706%	3.764%	3.224%	0.034% (0.33h)	0.026% (1.37h)
vangogh120K	6.812%	4.022%	3.156%	0.038% (0.49h)	0.028% (1.91h)
venus140K	6.581%	4.102%	3.067%	0.027% (0.62h)	0.023% (2.54h)
pareja160K	6.830%	4.259%	3.158%	0.032% (0.87h)	0.024% (3.50h)
courbet180K	7.145%	4.777%	3.294%	0.035% (1.04h)	0.025% (4.00h)
earring200K	7.605%	4.514%	3.528%	0.039% (1.32h)	0.029% (5.18h)
Mean	6.947%	4.240%	3.238%	0.034% (0.78h)	0.026% (3.08h)

D.4 Real-world CVRP results on CVRPLIB

Tables 26 and 27 report supplementary real-world CVRPLIB results on the `set-X` (100 instances) and `set-XL` (100 instances) collections. The L2C-Insert numbers use I=1000 runs over all instances. Figures 8 and 9 visualize LSSL routes on `X-n1001-k43` and `XL-n9571-k55`.

Table 26: Results on real-world CVRPLIB `set-X` CVRP instances.

Instance	L2C-Insert	Neuro LKH	BQ bs16	UDC-$\mathbf{x}_{250}$	LSSL Gap (C=1, B_t=4)	LSSL Gap (C=2, B_t=32)	LSSL Gap (C=4, B_t=128)	Instance	L2C-Insert	Neuro LKH	BQ bs16	UDC-$\mathbf{x}_{250}$	LSSL Gap (C=1, B_t=4)	LSSL Gap (C=2, B_t=32)	LSSL Gap (C=4, B_t=128)
X-n101-k25	2.555%	0.722%	11.240%	20.472%	0.000%	0.000%	0.000%	X-n336-k84	3.184%	3.119%	10.430%	9.355%	1.565%	0.656%	0.308%
X-n106-k14	0.751%	0.632%	2.121%	5.174%	0.099%	0.000%	0.000%	X-n344-k43	2.694%	2.305%	4.040%	7.426%	1.384%	0.150%	0.109%
X-n110-k13	0.048%	0.689%	0.849%	6.153%	0.000%	0.000%	0.000%	X-n351-k40	3.906%	4.026%	5.826%	7.132%	1.877%	0.768%	0.417%
X-n115-k10	0.629%	0.289%	11.065%	14.397%	0.000%	0.000%	0.000%	X-n359-k29	2.682%	3.654%	2.572%	3.575%	1.654%	0.666%	0.285%
X-n120-k6	0.038%	0.433%	0.077%	9.361%	0.000%	0.000%	0.000%	X-n367-k17	4.209%	2.375%	5.353%	6.709%	1.302%	0.053%	0.000%
X-n125-k30	2.901%	0.797%	12.173%	7.907%	0.187%	0.000%	0.000%	X-n376-k94	0.965%	0.560%	0.810%	5.659%	0.185%	0.056%	0.014%
X-n129-k18	0.926%	0.982%	1.833%	4.828%	0.000%	0.000%	0.000%	X-n384-k52	3.715%	4.352%	2.989%	7.946%	1.333%	0.588%	0.182%
X-n134-k13	0.708%	1.178%	5.152%	6.082%	0.192%	0.000%	0.000%	X-n393-k38	3.466%	2.182%	4.451%	7.754%	1.291%	0.434%	0.120%
X-n139-k10	0.127%	0.829%	1.398%	4.994%	0.000%	0.000%	0.000%	X-n401-k29	2.013%	1.365%	5.900%	4.539%	0.981%	0.274%	0.162%
X-n143-k7	1.913%	1.184%	1.860%	6.668%	0.000%	0.000%	0.000%	X-n411-k19	4.926%	2.126%	8.834%	8.333%	1.618%	0.122%	0.081%
X-n148-k46	1.786%	0.852%	13.526%	8.938%	0.000%	0.000%	0.000%	X-n420-k130	5.893%	3.682%	16.931%	7.965%	1.336%	0.579%	0.189%
X-n153-k22	3.811%	0.429%	25.813%	13.921%	0.603%	0.047%	0.024%	X-n429-k61	3.520%	2.463%	3.117%	8.011%	1.335%	0.324%	0.272%
X-n157-k13	0.588%	0.171%	2.829%	3.847%	0.000%	0.000%	0.000%	X-n439-k37	1.459%	2.165%	1.769%	9.204%	0.742%	0.058%	0.033%
X-n162-k11	0.721%	0.759%	1.633%	6.509%	0.000%	0.000%	0.000%	X-n449-k29	4.074%	6.118%	2.700%	4.968%	2.104%	0.677%	0.244%
X-n167-k10	1.693%	1.463%	1.811%	9.328%	0.185%	0.000%	0.000%	X-n459-k26	5.629%	3.745%	5.558%	5.285%	1.471%	0.555%	0.406%
X-n172-k51	2.119%	0.850%	11.831%	9.825%	0.410%	0.007%	0.000%	X-n469-k138	5.822%	4.694%	8.496%	9.305%	1.049%	0.597%	0.295%
X-n176-k26	4.533%	0.978%	20.335%	10.967%	0.379%	0.098%	0.000%	X-n480-k70	2.976%	1.650%	3.226%	6.069%	1.377%	0.505%	0.206%
X-n181-k23	0.088%	0.457%	1.170%	5.649%	0.149%	0.000%	0.000%	X-n491-k59	3.679%	3.372%	4.857%	9.387%	2.119%	1.009%	0.516%
X-n186-k15	0.702%	1.566%	2.546%	7.575%	0.041%	0.000%	0.000%	X-n502-k39	1.551%	0.578%	1.887%	4.190%	0.397%	0.185%	0.048%
X-n190-k8	1.374%	1.648%	3.097%	2.108%	0.483%	0.124%	0.000%	X-n513-k21	3.522%	2.690%	4.325%	4.343%	1.545%	0.293%	0.145%
X-n195-k51	1.718%	1.047%	16.678%	14.228%	0.497%	0.106%	0.036%	X-n524-k153	5.997%	2.594%	19.326%	9.441%	1.858%	0.361%	0.182%
X-n200-k36	2.963%	0.899%	6.738%	6.132%	0.159%	0.000%	0.000%	X-n536-k96	4.883%	4.944%	6.333%	7.700%	1.360%	0.490%	0.254%
X-n204-k19	1.473%	1.496%	3.916%	8.995%	0.199%	0.026%	0.000%	X-n548-k50	3.086%	1.208%	0.665%	5.936%	0.708%	0.288%	0.073%
X-n209-k16	1.679%	1.579%	1.728%	7.135%	0.150%	0.085%	0.000%	X-n561-k42	3.673%	2.800%	3.268%	4.267%	1.145%	0.487%	0.276%
X-n214-k11	2.405%	5.916%	4.394%	5.749%	0.590%	0.166%	0.046%	X-n573-k30	3.327%	1.920%	8.491%	4.870%	1.269%	0.620%	0.458%
X-n219-k73	0.715%	0.118%	5.239%	5.902%	0.013%	0.000%	0.000%	X-n586-k159	4.877%	3.268%	7.126%	9.208%	1.144%	0.324%	0.193%
X-n223-k34	1.932%	1.449%	3.367%	6.045%	0.381%	0.084%	0.020%	X-n599-k92	4.814%	12.395%	3.140%	8.897%	1.759%	0.579%	0.244%
X-n228-k23	4.172%	1.121%	22.614%	8.700%	0.618%	0.256%	0.004%	X-n613-k62	4.565%	4.900%	4.744%	8.450%	2.395%	0.761%	0.366%
X-n233-k16	1.748%	1.911%	2.670%	10.573%	0.593%	0.021%	0.000%	X-n627-k43	4.867%	3.303%	6.620%	7.103%	1.802%	0.857%	0.512%
X-n237-k14	1.730%	1.334%	1.120%	9.834%	0.255%	0.000%	0.000%	X-n641-k35	6.308%	3.081%	2.684%	8.723%	1.812%	0.616%	0.380%
X-n242-k48	2.013%	1.200%	2.976%	5.774%	0.859%	0.315%	0.070%	X-n655-k131	2.755%	0.906%	3.801%	4.280%	0.291%	0.111%	0.050%
X-n247-k50	5.312%	0.979%	23.822%	11.497%	0.829%	0.054%	0.011%	X-n670-k130	8.781%	6.485%	25.411%	11.567%	3.491%	0.679%	0.517%
X-n251-k28	1.867%	1.488%	4.115%	6.474%	0.879%	0.305%	0.000%	X-n685-k75	5.100%	3.846%	5.379%	9.499%	1.984%	0.529%	0.213%
X-n256-k16	1.233%	3.158%	4.254%	7.759%	0.218%	0.218%	0.218%	X-n701-k44	4.425%	2.850%	3.272%	3.566%	2.080%	0.926%	0.599%
X-n261-k13	1.747%	2.825%	2.430%	4.783%	0.249%	0.038%	0.000%	X-n716-k35	5.550%	3.968%	7.018%	7.058%	2.799%	0.747%	0.380%
X-n266-k58	2.449%	1.464%	3.282%	8.724%	1.096%	0.448%	0.129%	X-n733-k159	3.859%	2.236%	10.624%	7.389%	1.781%	0.708%	0.296%
X-n270-k35	1.610%	1.492%	3.494%	6.961%	0.496%	0.091%	0.045%	X-n749-k98	5.211%	4.334%	6.981%	7.860%	2.566%	1.173%	0.844%
X-n275-k28	0.953%	1.257%	1.958%	6.474%	0.522%	0.000%	0.000%	X-n766-k71	7.543%	3.376%	12.310%	8.194%	2.397%	0.797%	0.545%
X-n280-k17	3.180%	2.016%	7.707%	6.153%	0.654%	0.090%	0.033%	X-n783-k48	5.469%	4.276%	3.018%	7.504%	2.610%	0.942%	0.442%
X-n284-k15	2.698%	3.084%	3.708%	3.156%	1.271%	0.415%	0.237%	X-n801-k40	5.258%	3.080%	2.019%	12.415%	1.930%	0.604%	0.250%
X-n289-k60	2.996%	1.942%	7.480%	9.594%	1.051%	0.427%	0.131%	X-n819-k171	6.224%	4.429%	7.579%	7.426%	1.645%	0.770%	0.434%
X-n294-k50	2.343%	1.598%	6.054%	9.851%	1.126%	0.174%	0.117%	X-n837-k142	4.272%	2.266%	2.797%	7.453%	1.544%	0.618%	0.247%
X-n298-k31	1.878%	2.578%	1.794%	5.816%	0.640%	0.009%	0.000%	X-n856-k95	3.428%	2.177%	4.429%	6.336%	0.905%	0.238%	0.129%
X-n303-k21	2.408%	2.068%	3.735%	7.892%	0.732%	0.230%	0.046%	X-n876-k59	4.789%	3.603%	5.382%	4.482%	2.206%	1.069%	0.480%
X-n308-k13	1.851%	1.358%	4.046%	6.712%	1.036%	0.162%	0.012%	X-n895-k37	7.105%	9.621%	5.113%	9.000%	3.390%	1.175%	0.618%
X-n313-k71	3.754%	2.794%	7.849%	7.409%	1.185%	0.336%	0.046%	X-n916-k207	4.599%	3.091%	5.222%	7.502%	1.205%	0.609%	0.265%
X-n317-k53	0.784%	0.149%	2.662%	3.595%	0.170%	0.027%	0.005%	X-n936-k151	10.396%	9.459%	25.758%	11.797%	3.711%	1.873%	0.824%
X-n322-k28	2.886%	2.672%	3.370%	9.205%	0.697%	0.359%	0.047%	X-n957-k87	4.176%	2.242%	3.788%	7.782%	1.182%	0.541%	0.301%
X-n327-k20	3.329%	3.107%	3.436%	9.048%	1.010%	0.352%	0.015%	X-n979-k58	5.213%	2.619%	4.560%	5.981%	2.824%	0.804%	0.334%
X-n331-k15	1.996%	1.484%	1.909%	10.577%	0.839%	0.071%	0.003%	X-n1001-k43	5.996%	4.583%	4.209%	4.941%	2.503%	1.328%	1.097%
Mean	3.223%	2.496%	6.160%	7.592%	1.067%	0.353%	0.171%								

Table 27: Results on real-world CVRPLIB `set-XL` CVRP instances.

Instance	L2C-Insert Gap	L2C-Insert time	LSSL Gap (C=1, B_t=4)	LSSL Gap (C=2, B_t=32)	LSSL Gap (C=4, B_t=128)	Instance	L2C-Insert Gap	L2C-Insert time	LSSL Gap (C=1, B_t=4)	LSSL Gap (C=2, B_t=32)	LSSL Gap (C=4, B_t=128)
XL-n1048-k237	12.464%	657.17s	-	3.248%	1.914%	XL-n3408-k524	8.497%	2451.82s	2.237%	1.633%	1.415%
XL-n1094-k157	5.244%	680.48s	0.630%	0.369%	0.276%	XL-n3484-k436	4.041%	2420.72s	0.430%	0.262%	0.218%
XL-n1141-k112	11.177%	664.97s	1.494%	1.034%	0.636%	XL-n3561-k229	15.092%	2372.08s	4.240%	3.161%	2.171%
XL-n1188-k96	9.826%	668.40s	2.200%	1.183%	0.936%	XL-n3640-k211	18.468%	2385.89s	4.090%	2.823%	2.007%
XL-n1234-k55	9.397%	693.84s	2.871%	1.997%	1.486%	XL-n3721-k77	22.581%	2483.03s	5.279%	3.754%	2.076%
XL-n1281-k29	17.010%	698.57s	3.646%	1.990%	1.621%	XL-n3804-k29	36.146%	2593.52s	8.042%	5.228%	2.749%
XL-n1328-k19	22.963%	749.11s	4.633%	1.924%	0.899%	XL-n3888-k1010	11.575%	3326.79s	-	-	2.733%
XL-n1374-k278	10.076%	766.16s	1.928%	1.277%	0.841%	XL-n3975-k687	8.484%	3435.63s	2.271%	1.887%	1.506%
XL-n1421-k232	5.262%	790.87s	1.431%	1.253%	0.903%	XL-n4063-k347	10.341%	3243.75s	2.638%	1.935%	1.562%
XL-n1468-k151	7.438%	791.09s	1.490%	0.961%	0.684%	XL-n4153-k291	11.321%	3296.78s	2.396%	1.996%	1.415%
XL-n1514-k106	12.103%	797.49s	2.354%	1.180%	0.732%	XL-n4245-k203	23.355%	3320.98s	2.517%	1.435%	1.153%
XL-n1561-k75	13.303%	814.39s	1.649%	1.032%	0.795%	XL-n4340-k148	22.302%	3330.50s	6.047%	3.095%	2.453%
XL-n1608-k39	21.031%	803.25s	3.357%	1.718%	1.037%	XL-n4436-k48	39.119%	3896.42s	6.651%	4.263%	2.941%
XL-n1654-k11	19.225%	824.63s	7.646%	2.682%	1.333%	XL-n4535-k1134	6.746%	4288.59s	0.229%	0.167%	0.128%
XL-n1701-k562	14.417%	988.37s	1.515%	1.107%	1.095%	XL-n4635-k790	9.644%	4505.06s	2.777%	2.266%	2.122%
XL-n1748-k271	11.071%	1000.83s	2.142%	1.629%	1.307%	XL-n4738-k487	9.018%	4499.19s	2.194%	1.804%	1.327%
XL-n1794-k163	8.338%	918.05s	1.340%	0.777%	0.617%	XL-n4844-k321	11.555%	4467.05s	4.448%	3.097%	2.557%
XL-n1841-k126	9.424%	909.82s	2.903%	2.064%	1.633%	XL-n4951-k203	20.809%	4576.57s	5.696%	3.212%	2.221%
XL-n1888-k82	12.208%	954.52s	3.094%	2.175%	1.539%	XL-n5061-k184	30.736%	4812.85s	5.630%	3.480%	2.323%
XL-n1934-k46	18.939%	970.06s	4.029%	2.845%	2.194%	XL-n5174-k55	49.057%	5003.24s	11.200%	7.484%	5.526%
XL-n1981-k13	33.021%	1081.15s	6.154%	3.072%	2.020%	XL-n5288-k1246	10.254%	6199.65s	1.870%	1.053%	0.965%
XL-n2028-k617	12.630%	1127.05s	1.927%	1.185%	1.027%	XL-n5406-k783	5.849%	6475.50s	2.059%	1.507%	1.238%
XL-n2074-k264	6.360%	1150.32s	2.575%	2.178%	1.910%	XL-n5526-k553	12.169%	6109.14s	1.544%	0.998%	0.668%
XL-n2121-k186	7.456%	1144.95s	3.397%	2.149%	1.636%	XL-n5649-k401	12.063%	6107.04s	8.870%	3.495%	2.631%
XL-n2168-k138	13.529%	1170.37s	3.470%	2.328%	1.626%	XL-n5774-k290	22.527%	6181.67s	6.601%	4.240%	3.115%
XL-n2214-k131	11.267%	1186.25s	1.546%	0.836%	0.561%	XL-n5902-k122	29.987%	6373.03s	8.686%	5.191%	3.602%
XL-n2261-k54	14.323%	1186.05s	4.990%	3.532%	2.430%	XL-n6034-k61	51.171%	7074.99s	10.486%	7.228%	5.681%
XL-n2307-k34	21.714%	1203.58s	5.978%	3.119%	2.129%	XL-n6168-k1922	12.197%	8910.71s	-	2.717%	1.686%
XL-n2354-k631	9.937%	1451.36s	1.116%	0.609%	0.469%	XL-n6305-k1042	8.921%	9297.80s	2.475%	1.679%	1.409%
XL-n2401-k408	9.309%	1484.36s	2.667%	2.192%	1.546%	XL-n6445-k628	8.995%	9019.53s	2.195%	1.388%	1.168%
XL-n2447-k290	14.727%	1428.11s	4.416%	1.994%	1.414%	XL-n6588-k473	18.703%	8934.79s	5.019%	3.351%	2.772%
XL-n2494-k194	7.813%	1416.22s	1.981%	1.247%	1.002%	XL-n6734-k330	19.550%	8968.28s	4.083%	2.503%	1.693%
XL-n2541-k121	18.555%	1393.17s	2.547%	1.401%	1.125%	XL-n6884-k148	38.515%	9268.05s	6.788%	4.688%	3.545%
XL-n2587-k66	15.535%	1398.07s	5.122%	3.138%	2.124%	XL-n7037-k38	65.663%	10034.09s	10.397%	8.444%	5.015%
XL-n2634-k17	43.241%	1518.16s	10.063%	6.618%	5.683%	XL-n7193-k1683	9.048%	12958.66s	2.262%	0.677%	0.555%
XL-n2681-k540	6.156%	1605.13s	2.096%	1.493%	1.349%	XL-n7353-k1471	5.214%	13772.80s	0.515%	0.330%	0.266%
XL-n2727-k546	6.721%	1749.10s	0.611%	0.411%	0.315%	XL-n7516-k859	10.655%	13359.74s	3.397%	2.278%	1.998%
XL-n2774-k286	7.854%	1735.59s	1.861%	1.117%	0.896%	XL-n7683-k602	14.891%	13321.26s	2.549%	1.719%	1.432%
XL-n2821-k208	11.590%	1613.14s	4.007%	2.489%	2.103%	XL-n7854-k365	12.304%	13259.23s	-	1.360%	1.044%
XL-n2867-k120	13.502%	1631.60s	3.645%	1.976%	1.469%	XL-n8028-k294	28.905%	13917.90s	6.700%	4.599%	3.520%
XL-n2914-k95	16.882%	1704.10s	5.334%	3.402%	2.132%	XL-n8207-k108	67.731%	15057.08s	8.474%	5.488%	4.539%
XL-n2961-k55	19.949%	1734.12s	7.598%	4.440%	3.150%	XL-n8389-k2028	6.546%	18839.21s	-	-	1.887%
XL-n3007-k658	8.106%	2078.38s	2.109%	1.658%	1.107%	XL-n8575-k1297	8.681%	19603.71s	2.971%	1.906%	1.751%
XL-n3054-k461	7.083%	2132.77s	1.506%	1.107%	0.808%	XL-n8766-k1032	10.689%	19641.99s	4.882%	1.683%	1.405%
XL-n3101-k311	10.965%	2046.84s	3.688%	1.964%	1.338%	XL-n8960-k634	12.830%	19361.49s	2.921%	1.651%	1.329%
XL-n3147-k232	14.902%	2032.63s	2.583%	1.899%	1.450%	XL-n9160-k379	29.890%	19941.54s	7.983%	5.090%	3.738%
XL-n3194-k161	16.659%	1997.66s	3.874%	2.819%	2.292%	XL-n9363-k209	43.011%	20924.73s	6.987%	4.667%	2.985%
XL-n3241-k115	20.739%	2032.68s	2.812%	1.617%	1.114%	XL-n9571-k55	83.047%	22431.75s	29.430%	10.432%	7.478%
XL-n3287-k30	42.752%	2236.79s	7.440%	4.502%	2.936%	XL-n9784-k2774	7.775%	29594.32s	-	0.904%	0.771%
XL-n3334-k934	7.518%	2384.53s	1.357%	1.157%	0.892%	XL-n10001-k1570	11.069%	23881.49s	2.007%	1.133%	0.804%
Mean	17.494%	5337.3s	4.149%	2.505%	1.838%						

D.5 Scaling of inference and training cost with instance size

We profile inference (4-step Euler flow-matching) and training (single-step forward+backward) wall-clock and GPU peak memory for the LSSL backbone D_θ, validating the near-linear-in-N claim of Section 2. The architecture matches the main experiments; we compare sparse $N \times K$ (K=1000) against a dense $N \times N$ baseline at batch size 1, with 1000 instances per scale. Results are reported in Table 28.

Table 28: Inference (INF) and training (TR) wall-clock and GPU peak memory at batch size 1. Sparse uses K=1000; dense uses full $N \times N$ attention with the same backbone. OOM means out-of-memory on a 24 GB GPU.

N	K	Mode	INF total (s)	INF/iter (ms)	INF peak (MB)	TR total (s)	TR/step (ms)	TR peak (MB)	Notes
1,000	999	sparse	22.84	22.84	159	57.25	57.25	323	LSSL baseline
1,000	—	dense	4.36	4.36	143	14.22	14.22	1,105	dense fits at small N
10,000	1000	sparse	209.80	209.80	510	517.00	517.00	870	LSSL baseline
10,000	—	dense	337.76	337.76	4,100	—	—	OOM	TR needs ∼37 GB
100,000	1000	sparse	2,414.17	2,414.17	3,338	6,224.82	6,224.82	6,051	LSSL baseline
100,000	—	dense	—	—	—	—	—	OOM	infeasible (bias ≥40 GB)

D.6 Allocation of inference budget between refinement cycles and per-cycle search

We probe how a fixed wall-budget B should be split between the number of LSSL refinement cycles C and the per-cycle LKH time $\tau{=}B/C$. All runs use the canonical Guided-LKH configuration (Appendix H.2) on the 16-instance TSP10000 test set, $K{=}1000$, 8-way LKH parallelism, identical seed. Mean gap% vs. ground-truth tours is reported in Table 29; negative values mean LSSL beat the reference. Figure 4 visualizes an LSSL tour on one instance of this test set.

Table 29: Mean gap% on TSP10000 (16 instances) for total wall-budget $B = C \cdot \tau$. Columns: refinement cycles C (per-cycle time $\tau{=}B/C$).

B (s)	$C{=}1$	$C{=}2$	$C{=}4$	$C{=}8$	$C{=}16$
8	+**0.0329**	–	–	–	–
16	+0.0152	+**0.0059**	–	–	–
32	+0.0065	−**0.0029**	−0.0021	–	–
64	−0.0016	−**0.0067**	−0.0066	−0.0048	–
128	−0.0054	−0.0083	−0.0094	−**0.0101**	−0.0097

E Experimental Details

E.1 Hyperparameters

Let L^{enc} and L^{dec} denote the number of encoder and decoder layers, d_{enc} and d_{dec} their hidden widths, H the number of attention heads (shared by encoder and decoder), $d_h{=}d_\bullet/H$ the per-head dimension, and $d_{\mathrm{ffn}}{=}3d_\bullet$ the gated feed-forward hidden width. For TSP we use $L^{\mathrm{enc}}{=}16$, $L^{\mathrm{dec}}{=}6$, $d_{\mathrm{enc}}{=}d_{\mathrm{dec}}{=}256$, and $H{=}8$. For CVRP, which carries additional depot, demand, and capacity features, we widen the backbone to $d_{\mathrm{enc}}{=}d_{\mathrm{dec}}{=}384$ with $H{=}12$ attention heads, keeping $L^{\mathrm{enc}}{=}16$ and $L^{\mathrm{dec}}{=}6$. The same backbone is shared across all instance sizes within each task. For instances with $N > 1000$, the sparse neighborhood size is fixed to $K = 1000$. The main evaluation uses full candidates for $N \leq 1000$ and eight denoising steps per cycle. The separate four-step profiling experiment below is not the main evaluation schedule.

Optimization uses Muon for all backbones and all instance sizes, with weight decay $1{\times}10^{-4}$, global-norm gradient clipping at 1.0, peak learning rate $1{\times}10^{-3}$, linear warmup over the first $\sim 1\%$ of update steps, and cosine decay to $1{\times}10^{-5}$. The closed-loop schedule uses $M = 3$ search-refined route sets per instance to construct the row-wise soft target $\mathcal{H}^\star$, matching the parallelism of S at training time.

E.2 Training-label construction

Training labels are constructed in two regimes. At the small scales (TSP100, TSP1K, CVRP100), targets come from direct solver calls, with the per-scale sources listed in Table 30. At the larger scales (TSP10K, TSP100K, CVRP1K, CVRP10K), where uniform direct labelling exceeds our offline budget, labels are constructed through the self-bootstrapping mechanism of Section 2.5: the smaller-scale-trained LSSL checkpoint $\theta^{(s')}$ produces guidance heatmaps on freshly sampled target-scale instances, the search backend S solves each instance under its standard parameter file, and the resulting M route sets are projected and aggregated via Eq. (5) into the search-feedback target $\mathcal{H}^\star$, which is stored as the fixed training label. Table 30 summarises the per-scale recipe.

Self-improved training has been used to extend supervision for large-scale NCO. The SIT framework of Luo et al. [19] casts training as a multi-round iteration: at round r the current model $\theta^{(r)}$ reconstructs a tour on each training instance, those tours are recorded as the pseudo-labels for round $r{+}1$, and $\theta^{(r+1)}$ is trained on this updated supervision; by repeatedly weaving the model's own outputs back into the supervision signal, a single architecture is pushed past the level of its initial fixed-label set on TSP and CVRP. The label-construction recipe of this section inherits the same high-level idea (letting the model participate in constructing its own supervision) and instantiates it as a cross-scale bootstrap: the guidance signal $\theta^{(s')}$ comes from an LSSL checkpoint trained at a smaller scale, and every target-scale label is the actual solver output produced by the search backend

Table 30: Per-scale training-label recipe. Direct denotes a single solver call; self-bootstrap uses the smaller-scale checkpoint $\theta^{(s')}$ as guidance for the search backend S. The aggregation count M matches the search parallelism used at training time (cf. Section 2.5). Rounds r is the number of multi-round iterations of label refresh (Section 2.5); direct rows do not iterate.

Scale	Label source	Guidance $\theta^{(s')}$	Search backend S	M	Rounds r	# instances
TSP100	direct	–	Concorde	–	–	1.28M
TSP1K	direct	–	LKH3	3	–	64K
TSP10K	self-bootstrap	$\theta^{(\text{TSP1K})}$	LKH3	3	2	6.4K
TSP100K	self-bootstrap	$\theta^{(\text{TSP10K})}$	LKH3	3	2	640
CVRP100	direct	–	HGS	3	–	1.28M
CVRP1K	self-bootstrap	$\theta^{(\text{CVRP100})}$	HGS	3	2	64K
CVRP10K	self-bootstrap	$\theta^{(\text{CVRP1K})}$	HGS	3	2	6.4K

S under that guidance. The wall-clock cost of label construction is folded into the per-scale training budget reported in Appendix E.

E.3 Hardware

All training and inference results reported in the main paper were carried out on a single workstation: one NVIDIA GeForce RTX 4090 GPU (24 GB), two AMD EPYC 7K62 CPU (48 cores), and 128 GB of DDR4 system memory. The additional Serial-1/batched comparison allocates at most 16 CPU cores and 24 GiB GPU memory per method.

E.4 Computational Resources

All evaluations are run on a single NVIDIA RTX 4090 GPU paired with two AMD EPYC 7K62 CPU. On the same RTX 4090, end-to-end LSSL training takes ≤ 2 days / ≤ 1 day / ≤ 1 day for TSP-100 / TSP-1K / TSP-10K respectively; the TSP-10K/TSP-100K self-bootstrapping take ≤ 2 days and ≤ 1 days respectively.

For CVRP, training takes ≤ 3 days / ≤ 2 days / ≤ 2 days for CVRP-100 / CVRP-1K / CVRP-10K, and CVRP-1K / CVRP-10K self-bootstrapping take ≤ 3 days and ≤ 3 days respectively.

For comparison, prior heatmap baselines already require this range or more for training alone on stronger hardware: DIFUSCO [32] reports 8.6 and 5.1 days for TSP-100 and TSP-1K training on a single A100 GPU, whereas the budgets above already cover both training and self-bootstrapped label construction on a weaker consumer-grade RTX 4090, placing LSSL's total time in a comparable range.

E.5 Implementation details of baselines

This appendix details only those baselines that appear in Tables 1, 3, 4, and 5. We organise them into five families. **(1) Classical solvers:** Concorde [89], LKH3 [6], and HGS [43]. **(2) Constructive NCO methods:** POMO [12], BQ [14], LEHD [13], INViT [28], and the three L2C-Insert variants (Greedy, I=1000, I=10000) [21]. **(3) Heatmap-based NCO methods:** Att-GCN+MCTS [30], DIFUSCO [32], T2T [33], Fast-T2T [34], and GenSCO [35]. **(4) Two-stage methods:** H-TSP [22], GLOP [25], and UDC [26]; Tables 1 and 3 use UDC-$\mathbf{x}_{50}$, while Tables 4 and 5 use UDC-$\mathbf{x}_{250}$. **(5) Neural-guided classical search:** NeuroLKH [38], reported in Tables 4 and 5. Methods that never appear in these tables are not discussed here.

- **Concorde** [89]: we use the PyConcorde Python wrapper at its default settings on TSP instances of every size.
- **LKH3** [6]: we use the runner shipped with AM [11]. On TSP100K we follow Fu et al. [68] and switch the preprocessing stage to POPMUSIC [90] (CANDIDATE_SET_TYPE = POPMUSIC, INITIAL_PERIOD = 1000) to keep the per-instance cost manageable; all other parameters are kept at their defaults.
- **HGS** [43]: we use the PyHygese Python wrapper of the HGS-CVRP solver.

- **LEHD** [13]: we cap the RRC maximum destruction size at 1000. We adopted this cap after verifying that lifting it leaves the solution quality essentially unchanged while substantially increasing inference time.
- **L2C-Insert** [21]: we report the greedy decoder together with the local-reconstruction variants at I=1000 and I=10000; the corresponding rows are labelled L2C-Insert Greedy, L2C-Insert (I=1000), and L2C-Insert (I=10000). The I=10000 setting is reported only on TSP100K (Table 3).
- **UDC** [26]: we keep the variant naming of the original paper. The synthetic-instance main tables use UDC-$\mathbf{x}_{50}$ (α=50), and the real-world main tables use UDC-$\mathbf{x}_{250}$ (α=50).
- **DIFUSCO** [32]: we re-run the official categorical-diffusion checkpoint with T_s=50 denoising sampling steps, decode each instance by greedy heatmap decoding, and refine with 2-opt local search.
- **Fast-T2T** [34]: we re-run the official consistency-trained checkpoint with T_s=5 denoising sampling steps coupled with T_g=5 objective-guided rewrite cycles, generate candidate tours by stochastic heatmap decoding, and refine with 2-opt local search.
- **GenSCO** [35]: on the synthetic TSP main table (Table 1) we re-run the official implementation under C=10 and C=160, both with the bundled 2-opt post-processing, and label the corresponding rows GenSCO (C=10) 2Opt and GenSCO (C=160) 2Opt; both configurations run out of memory at TSP10K.
- **NeuroLKH** [38]: the TSPLIB column is transcribed from the supplementary TSPLIB tables of Xin et al. [38], and the VLSI column follows the real-world TSP summary used for Table 23. For CVRPLIB Set-X, we combine the available supplementary NeuroLKH entries for the smaller instances with our local rerun on the larger Set-X instances. The rerun uses the released NeuroLKH checkpoints and 1024 LKH trials, a 7200-second per-instance time limit.

F Network architecture details

Our backbone is inspired by the use of Transformer-based generative models for combinatorial optimization in [35], but differs in being tailored to LSSL's closed-loop search interface: all intermediate representations are backend-facing heatmaps on the sparse K-NN graph rather than dense pairwise node logits. This appendix gives the full architecture of the sparse diffusion Transformer D_θ (cf. Section 2.4). All operators map $N \times K$ tensors on the K-NN subgraph $\mathcal{G}_K$ to $N \times K$ tensors of the same shape; no $N \times N$ object is ever materialised, so per-layer compute and memory stay at $O(NK)$ for any candidate width $K \ll N$.

F.1 Per-node features and slot indexing

TSP and CVRP share the same sparse candidate graph $\mathcal{G}_K$. We treat $\mathcal{N}_K(i)$ as an ordered K-tuple of slots with the slot-to-node alias

$$j_{ik} \;:=\; \mathcal{N}_K(i)[\,k\,] \;\in\; \mathcal{V}, \qquad k \in \{1, \dots, K\}. \tag{12}$$

Per-node features are $\boldsymbol{s}_i = [\hat{x}_i, \hat{y}_i]$ for TSP and $[\hat{x}_i, \hat{y}_i, \rho_i]$ for CVRP, with $\rho_i = q_i/Q$ on customers (using the same q_i for customer demand as in Appendix I) and $\rho_0 = -1$ on the depot; coordinates $(\hat{x}_i, \hat{y}_i)$ are centred and isotropically rescaled by the mean node-wise ℓ_2 norm.

F.2 Heatmap construction at training time

The decoder consumes two conditioning channels, each obtained by projecting two endpoints onto $\mathcal{G}_K$ via the operator $P(\cdot; \mathcal{G}_K)$ of Section 2.2. The *edge channel* uses the multi-solution search-feedback target $\mathcal{H}^\star = P(\{\mathcal{R}^{(m)}\}_{m=1}^{M}; \mathcal{G}_K)$ of Eq. (5) (the row-normalised edge-usage frequency over the M retained route sets) together with an independent random route set $\mathcal{R}_\epsilon$ on the same instance, written as $\mathcal{H}_\epsilon = P(\mathcal{R}_\epsilon; \mathcal{G}_K)$ (no perturbation of any $\mathcal{R}^{(m)}$ is involved). The *node-selection channel* (CVRP only) uses the route-boundary frequency $\mathcal{H}^\star_{\text{node}}$ and a matched random boundary indicator $\mathcal{H}_{\text{node},\epsilon}$. Both channels are interpolated independently by the same noise level t:

$$\mathcal{H}_t \;=\; (1-t)\,\mathcal{H}^\star \;+\; t\,\mathcal{H}_\epsilon, \qquad \mathcal{H}_{\text{node},t} \;=\; (1-t)\,\mathcal{H}^\star_{\text{node}} \;+\; t\,\mathcal{H}_{\text{node},\epsilon}, \qquad t \sim \mathcal{U}(0,1). \tag{13}$$

The two channels are kept separate: $\mathcal{H}^\star$ supervises the row-stochastic edge head F_{out} via the row-wise sparse cross-entropy of Eq. (8), while $\mathcal{H}^\star_{\text{node}}$ supervises the per-customer node-selection head via the binary cross-entropy of Eq. (9); the two losses are different functionals on different decoder heads and are not collapsed into a single tensor at the input either. For TSP, $\mathcal{H}^\star_{\text{node}} = \mathcal{H}_{\text{node},\epsilon} = 0$ so $\mathcal{H}_{\text{node},t} \equiv 0$, only the edge channel is active, and Eq. (13) reduces to a pure linear interpolation between $\mathcal{H}^\star$ and $\mathcal{H}_\epsilon$, recovering Eq. (7).

CVRP boundary structure. The node-selection heatmaps $\mathcal{H}^\star_{\text{node}}, \mathcal{H}_{\text{node},\epsilon} \in [0,1]^{N_c}$ are per-customer scalars and live on a separate channel from the edge tensor $\mathcal{H}^\star$: $\mathcal{H}^\star_{\text{node}}$ is the route-boundary frequency over the same M route sets $\{\mathcal{R}^{(m)}\}_{m=1}^{M}$ used for $\mathcal{H}^\star$ (matching the supervision target of Eq. (9)), and $\mathcal{H}_{\text{node},\epsilon}$ is sampled by drawing customers from a small pool of those closest to the depot. The decoder pairs the edge head F_{out} with a separate per-customer head producing $\mathcal{H}_{\text{node}} \in [0,1]^{N_c}$.

F.3 Sparse diffusion Transformer backbone

Macro structure. $D_\theta(\mathcal{G}_K, \mathcal{H}_t, t)$ is a stack of five $N \times K$ operators:

$$\underbrace{F_{\text{in}}}_{\text{input projection}} \longrightarrow \underbrace{\big(F^{(\ell)}_{\text{enc}}\big)_{\ell=1}^{L^{\text{enc}}}}_{\text{sparse encoder}} \longrightarrow \underbrace{F_{\text{mid}} \oplus \boldsymbol{\tau}(t)}_{\text{time bridge}} \longrightarrow \underbrace{\big(F^{(\ell)}_{\text{dec}}\big)_{\ell=1}^{L^{\text{dec}}}}_{\text{time-conditioned decoder}} \longrightarrow \underbrace{F_{\text{out}}}_{\text{edge head}} . \tag{14}$$

$F_{\text{in}} : \mathbb{R}^{d_s} \to \mathbb{R}^{d_{\text{enc}}}$ is a per-node affine map ($d_s = 2$ TSP, $d_s = 3$ CVRP). The encoder propagates $\{\boldsymbol{s}_i\}$ on $\mathcal{G}_K$ to instance-aware node states; the decoder additionally consumes $\mathcal{H}_t$ and t to inject which candidate edges are currently believed to belong to a high-quality solution and how informative that belief is. The edge head F_{out} produces row-stochastic $\mathcal{H}_{\text{guide}} \in \mathbb{R}_+^{N \times K}$ in the same shape as $\mathcal{H}_t$.

Time embedding. The noise level t is encoded by a sinusoidal basis followed by a two-layer MLP, with frequency width d_{freq}, max period $T_{\max} = 10000$, and timestep scale T_{scale}:

$$\begin{aligned} \boldsymbol{\tau}(t) \;&=\; W_2^\top \,\text{SiLU}\big(W_1^\top\,[\cos(\boldsymbol{\omega}\bar{t}), \sin(\boldsymbol{\omega}\bar{t})]\big) \;\in\; \mathbb{R}^{d_{\text{dec}}}, \\ \omega_k \;&=\; \exp\!\Big(-\tfrac{2k}{d_{\text{freq}}} \log T_{\max}\Big), \qquad \bar{t} \;=\; T_{\text{scale}}\, t. \end{aligned} \tag{15}$$

$\boldsymbol{\tau}(t)$ is injected additively at the decoder entrance: $\boldsymbol{h}_i^{(0,\text{dec})} = F_{\text{mid}} \boldsymbol{h}_i^{(L^{\text{enc}},\text{enc})} + \boldsymbol{\tau}(t)$, where $F_{\text{mid}} : \mathbb{R}^{d_{\text{enc}}} \to \mathbb{R}^{d_{\text{dec}}}$ aligns encoder/decoder widths.

Pre-norm Transformer block. Each encoder/decoder layer is a pre-norm Transformer block on the $N \times K$ tensor:

$$\widetilde{\boldsymbol{h}}_i^{(\ell)} \;=\; \boldsymbol{h}_i^{(\ell-1)} + \text{SparseMHA}^{(\ell)}\big(\,\text{LN}\big(\boldsymbol{h}_\bullet^{(\ell-1)}\big);\, \mathcal{G}_K,\, b_\bullet^{(\ell)}\big)_i, \tag{16}$$

$$\boldsymbol{h}_i^{(\ell)} \;=\; \widetilde{\boldsymbol{h}}_i^{(\ell)} + \text{SwiGLU}^{(\ell)}\big(\,\text{LN}\big(\widetilde{\boldsymbol{h}}_i^{(\ell)}\big)\big), \tag{17}$$

with bias-free LayerNorm, the SparseMHA operator detailed in Section F.4, and the position-wise gated FFN

$$\text{SwiGLU}^{(\ell)}(\boldsymbol{u}) = \big(\,\text{SiLU}(W_1^{(\ell)}\boldsymbol{u} + \mathbf{b}_1^{(\ell)}) \odot (W_2^{(\ell)}\boldsymbol{u} + \mathbf{b}_2^{(\ell)})\,\big)\, W_3^{(\ell)} + \mathbf{b}_3^{(\ell)}, \tag{18}$$

of hidden width $d_{\text{ffn}} = 3d$ acting independently on each row i. Optionally, query and key activations are normalised by a parameter-free RMSNorm on the head dimension before the dot product.

F.4 Sparse multi-head attention forward operator

For H heads of width d_h ($d = H d_h$), with $\bar{\boldsymbol{h}}_\bullet^{(\ell)} = \text{LN}(\boldsymbol{h}_\bullet^{(\ell-1)})$, the per-head projections at layer ℓ, head h, row i, candidate $j \in \mathcal{N}_K(i)$ are

$$\boldsymbol{q}_i^{(\ell,h)} = W_Q^{(\ell,h)} \bar{\boldsymbol{h}}_i^{(\ell)}, \quad \boldsymbol{k}_j^{(\ell,h)} = W_K^{(\ell,h)} \bar{\boldsymbol{h}}_j^{(\ell)}, \quad \boldsymbol{v}_j^{(\ell,h)} = W_V^{(\ell,h)} \bar{\boldsymbol{h}}_j^{(\ell)}. \tag{19}$$

The softmax-input score combines the bilinear term with a per-row bias:

$$e_{ij}^{(\ell,h)} \;=\; \tfrac{\langle \boldsymbol{q}_i^{(\ell,h)}, \boldsymbol{k}_j^{(\ell,h)} \rangle}{\sqrt{d_h}} \;+\; b_{ij}^{(\ell)}, \tag{20}$$

with $b^{(\ell)} \in \mathbb{R}^{N\times K}$ per instance (broadcast across batch and heads). Consistent with the main-text decoder bias of Eq. (3), in the decoder $b_{ij}^{(\ell)}$ fuses the per-edge heatmap entry with a sinusoidal time embedding through learned scalar projections $u^{(\ell)}, v^{(\ell)}$, while in the encoder it vanishes:

$$b_{ij}^{(\ell)} \;=\; u^{(\ell)}\big([\mathcal{H}_t]_{ij}\big) \;+\; v^{(\ell)}\big(\phi(t)\big) \quad \text{(decoder)}; \qquad b_{ij}^{(\ell)} \equiv 0 \quad \text{(encoder)}. \tag{21}$$

Here $\phi(t)$ is a sinusoidal embedding of t, and $u^{(\ell)}, v^{(\ell)}$ are layer-wise learned scalar linear maps; the second term is broadcast across (i, j) within a layer, so $b^{(\ell)}$ remains $N\times K$ per instance. Self-loops are excluded by construction since $\mathcal{N}_K(i)$ is a strict K-NN list. CVRP-specific structure (route boundaries) enters attention through both channels of Eq. (13): the edge tensor $\mathcal{H}_t$ contributes $u^{(\ell)}([\mathcal{H}_t]_{ij})$ on every (i, j) slot, while the per-customer node-selection scalar $[\mathcal{H}_{\mathrm{node},t}]_i$ of Section F.2 contributes an additional learned scalar projection broadcast across all K slots of customer row i (the depot row receives no node-selection bias), so the per-edge edge channel and the per-row node channel act on different axes without being summed at the input.

Block-streaming forward. We never materialize the row-wise softmax over $\mathcal{N}_K(i)$. Instead, the operator visits each row's neighborhood in a sequence of disjoint blocks $\mathcal{B}_1, \ldots, \mathcal{B}_{N_b}$ that together cover $\mathcal{N}_K(i)$, and maintains an exact online-softmax statistic. After block b the maximum and partition terms are

$$m_i^{(\ell,h,b)} \;=\; \max\Big(m_i^{(\ell,h,b-1)},\, \max_{j\in\mathcal{B}_b} e_{ij}^{(\ell,h)}\Big), \qquad m_i^{(\ell,h,0)} = -\infty, \tag{22}$$

$$\lambda_i^{(\ell,h,b)} \;=\; \lambda_i^{(\ell,h,b-1)}\, \exp\big(m_i^{(\ell,h,b-1)} - m_i^{(\ell,h,b)}\big) \;+\; \sum_{j\in\mathcal{B}_b} \exp\big(e_{ij}^{(\ell,h)} - m_i^{(\ell,h,b)}\big), \qquad \lambda_i^{(\ell,h,0)} = 0, \tag{23}$$

and the running value aggregation $\boldsymbol{o}_i^{(\ell,h,b)}$ obeys the same rebasing rule,

$$\begin{aligned} \boldsymbol{o}_i^{(\ell,h,b)} \;=\; & \frac{\lambda_i^{(\ell,h,b-1)}\, \exp\big(m_i^{(\ell,h,b-1)} - m_i^{(\ell,h,b)}\big)}{\lambda_i^{(\ell,h,b)}}\, \boldsymbol{o}_i^{(\ell,h,b-1)} \\ & +\; \frac{1}{\lambda_i^{(\ell,h,b)}} \sum_{j\in\mathcal{B}_b} \exp\big(e_{ij}^{(\ell,h)} - m_i^{(\ell,h,b)}\big)\, \boldsymbol{v}_j^{(\ell,h)}. \end{aligned} \tag{24}$$

After the final block N_b, the head output is $\boldsymbol{o}_i^{(\ell,h)} = \boldsymbol{o}_i^{(\ell,h,N_b)}$ and the row-wise normalized attention weight is

$$\alpha_{ij}^{(\ell,h)} \;=\; \frac{\exp\big(e_{ij}^{(\ell,h)} - m_i^{(\ell,h,N_b)}\big)}{\lambda_i^{(\ell,h,N_b)}}, \qquad \sum_{j\in\mathcal{N}_K(i)} \alpha_{ij}^{(\ell,h)} = 1. \tag{25}$$

The heads are concatenated and projected by $W_O^{(\ell)} \in \mathbb{R}^{d\times d}$ to obtain the layer's attention output. By construction the operator only ever accesses the gathered tensors $\boldsymbol{k}_{\mathcal{N}_K(i)}^{(\ell,h)}, \boldsymbol{v}_{\mathcal{N}_K(i)}^{(\ell,h)}, b_{i,\mathcal{N}_K(i)}^{(\ell)}$ for a single row i at a time, so its working set scales as $O(K\, d_h\, H)$ (independent of $|\mathcal{V}|$) and its compute as $O(|\mathcal{V}|\, K\, d)$ per layer, well below the dense $O(|\mathcal{V}|^2\, d)$.

F.5 Edge-scoring head and row-wise sparse cross-entropy

Let $\boldsymbol{g}_i$ be the projected decoder state at row i after the final norm. The edge head scores only the candidate edges of $\mathcal{G}_K$ (logit scale γ):

$$[\mathcal{H}_{\mathrm{guide}}]_{ij} \;=\; \frac{\exp\big(\gamma\, \langle \boldsymbol{g}_i, \boldsymbol{g}_j\rangle\big)}{\sum_{j'\in\mathcal{N}_K(i)} \exp\big(\gamma\, \langle \boldsymbol{g}_i, \boldsymbol{g}_{j'}\rangle\big)}, \qquad j \in \mathcal{N}_K(i), \tag{26}$$

with padding slots masked before normalisation; the scoring reuses the decoder's $N \times K$ sparse index, never materialising a dense $|\mathcal{V}|\times|\mathcal{V}|$ matrix.

Consistent with the main-text definition (Eq. (5)–(6)), the row-wise soft target is the multi-solution edge-usage frequency over the M retained route sets $\{\mathcal{R}^{(m)}\}_{m=1}^{M}$ projected onto $\mathcal{G}_K$:

$$\mathcal{H}_{ij}^{\star} \;=\; \frac{\sum_{m=1}^{M} \mathbb{1}\big\{(i,j) \in \mathcal{R}^{(m)}\big\}}{\sum_{j'\in\mathcal{N}_K(i)} \sum_{m=1}^{M} \mathbb{1}\big\{(i,j') \in \mathcal{R}^{(m)}\big\}}, \qquad j \in \mathcal{N}_K(i), \tag{27}$$

so $\mathcal{H}^{\star}_{ij} > 0$ iff j is a tour-neighbour of i in at least one $\mathcal{R}^{(m)}$, and edges hit by more route sets receive proportionally larger mass; for CVRP this also covers $j = 0$ on route-boundary customers, since their depot adjacency is counted by the same indicator. The LSSL training objective is the row-weighted sparse cross-entropy

$$\mathcal{L}_{\mathrm{LSSL}} \;=\; -\frac{1}{N}\sum_{i=1}^{N}\sum_{j\in\mathcal{N}_K(i)} \mathcal{H}^{\star}_{ij}\,\log[\mathcal{H}_{\mathrm{guide}}]_{ij}, \tag{28}$$

G Analysis of the computational complexity of LSSL

The complexity of LSSL has three sources: the sparse neural backbone, the projection/heatmap interface, and the heuristic search backend. All three are restricted to the same $N \times K$ sparse adjacency, where $K \ll N$ is the candidate width per node, so scaling along N stays linear rather than $O(N^2)$. Throughout this section, N is the number of nodes, K the candidate width per row, d the hidden dimension, L the number of sparse attention layers, S_{denoise} the number of denoising calls along the noise schedule, C the number of learning–search cycles, M the number of refined route sets retained per training instance, and B the per-cycle search budget (instantiated as B_{trials}, the LKH trial count, for TSP, and B_t, the HGS time budget, for CVRP).

Sparse backbone and supervision cost For a single network call, sparse attention, structural-bias injection, candidate-edge scoring, and the row-wise softmax are all performed only over $\mathcal{N}_K(i)$, while the per-node FFN and projections have fixed per-node width d. The dominant time complexity is therefore

$$T_{D_\theta} = O\big(L\,N\,(Kd + d^2)\big)\,,$$

with sparse storage $O(NK)$; the blockwise online-softmax used in this paper processes attention block by block without ever materializing the full per-layer, per-head weight matrix. Full neural inference across multiple cycles costs $O(C\,S_{\mathrm{denoise}}\,L\,N(Kd + d^2))$, so the only factor scaling with problem size is N. The projection P from a tour or a route set back to the same $N \times K$ sparse adjacency is approximately linear in the number of solution edges (each TSP node contributes a predecessor and a successor; CVRP boundaries add depot slots), with $O(NK)$ as the sparse-storage upper bound. At training time, building the search-feedback target $\mathcal{H}^{\star}$ from M refined route sets requires $O(MN)$ edge scanning plus $O(NK)$ storage, and the row-wise cross-entropy in Eq. (8) is computed only over the NK candidate slots, never over a dense $N \times N$ label matrix.

H LKH algorithm for TSP

This appendix specifies how the Lin–Kernighan–Helsgaun heuristic [6] is integrated into the LSSL learning–search loop. Section H.1 formalises vanilla LKH around a single per-node candidate measure, the sole channel through which structural prior information enters k-opt search. Section H.2 introduces Guided-LKH, which retains the LKH search core but replaces this static, geometry-based measure with a trajectory-conditioned one refreshed at every LSSL cycle. Algorithm 2 provides a line-level diff between the two main loops.

H.1 Vanilla LKH

Let an instance be a complete graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ on $N = |\mathcal{V}|$ cities with non-negative edge weight $w : \mathcal{E} \to \mathbb{R}_{\geq 0}$, and let $\mathrm{cost}(\mathcal{R}) = \sum_{e \in \mathcal{R}} w(e)$ denote the length of a Hamiltonian tour $\mathcal{R} \subset \mathcal{E}$. Vanilla LKH [77, 6] attacks $\min_{\mathcal{R}} \mathrm{cost}(\mathcal{R})$ by sequential k-opt local search. Its efficiency rests on two static, instance-only objects fixed once at the start: a node-potential vector $\pi^\star \in \mathbb{R}^N$ used to lift the cost matrix into reduced costs, and a per-vertex candidate measure $\mathcal{C}^{\mathrm{geo}}(i)$ that lists, for each city i, the K neighbours that the k-opt recursion is allowed to consider when extending a partial move from i. We introduce the four ingredients in turn (k-opt, the minimum 1-tree, α-nearness, and the dual $\pi^\star$) and then state the trial loop summarised in Algorithm 2(a).

k-opt move and sequential search. A k-opt move replaces k tour edges $\{x_1, \ldots, x_k\} \subset \mathcal{R}$ with k non-tour edges $\{y_1, \ldots, y_k\}$ such that the resulting edge set is again a Hamiltonian tour $\mathcal{R}'$ with $\mathrm{cost}(\mathcal{R}') < \mathrm{cost}(\mathcal{R})$. LKH restricts attention to sequential k-opt moves whose removed and added edges chain into a single alternating city sequence $(p_1, p_2, \ldots, p_{2k}, p_{2k+1}{=}p_1)$, with $x_i = (p_{2i-1}, p_{2i})$ and $y_i = (p_{2i}, p_{2i+1})$, and explores them by a depth-first recursion of maximum depth $k_{\max}{=}5$ (the LKH default). At each removed edge x_i the next vertex p_{2i} is one of the two tour neighbours of p_{2i-1}; at each added edge y_i the next vertex p_{2i+1} is drawn from the candidate set $\mathcal{C}^{\mathrm{geo}}(p_{2i})$ defined below. The recursion accepts the first y-sequence whose partial gain $\sum_{j \leq i}\big(w(x_j) - w(y_j)\big)$ remains positive at every depth and whose closing edge (p_{2i}, p_1) yields a strictly improving full tour. Once accepted, the move is applied and search continues from the new tour; the trial terminates when one full sweep over $\mathcal{V}$ uncovers no further improving move.

Minimum 1-tree and α-nearness. LKH constructs the candidate set from a sensitivity analysis of the Held–Karp lower bound. Fix an arbitrary special vertex $v_0 \in \mathcal{V}$. A 1-tree of $\mathcal{G}$ is a spanning tree of $\mathcal{V} \setminus \{v_0\}$ together with two distinct edges of $\mathcal{E}$ incident to v_0; a minimum 1-tree T is a 1-tree of minimum total cost. Its length $L(T)$ is a valid lower bound on $\min_{\mathcal{R}} \mathrm{cost}(\mathcal{R})$, and equals it whenever every vertex has degree exactly 2 in T (in which case T is the optimal tour). The α-value of an edge $(i, j) \in \mathcal{E}$ measures by how much $L(T)$ would have to grow to forcibly contain (i, j):

$$\alpha(i,j) \;=\; L\big(T^+(i,j)\big) \;-\; L(T), \qquad T^+(i,j) \;=\; \arg\min_{T' \ni (i,j)} L(T'). \tag{29}$$

Edges with small α are precisely those the lower bound is unwilling to pay much to include, and are therefore strong candidates for an optimal tour.

Held–Karp dual and subgradient ascent. The raw statistic in (29) is sharp only when $L(T)$ already approaches the optimum. To tighten it, LKH lifts the cost matrix by node potentials $\pi = (\pi_1, \ldots, \pi_N)$ via the reduced cost

$$\tilde{w}_{ij}(\pi) \;=\; w_{ij} + \pi_i + \pi_j. \tag{30}$$

This reduction is invariant on the optimal tour (every tour visits each vertex exactly twice in incident-edge degree, so its total cost merely shifts by the constant $2\sum_i \pi_i$) yet shifts the minimum 1-tree. Writing $L(T_\pi)$ for the minimum 1-tree length under $\tilde{w}(\pi)$, the Held–Karp lower bound becomes

$$w(\pi) \;=\; L\big(T_\pi\big) \;-\; 2\sum_{i \in \mathcal{V}} \pi_i, \tag{31}$$

and any maximiser $\pi^\star \in \arg\max_\pi w(\pi)$ pushes every vertex's degree in T_π towards 2. LKH approximates $\pi^\star$ by subgradient ascent (the LKH ASCENT routine): starting from $\pi^0 = 0$ it iterates

$$\pi^{\tau+1} \;=\; \pi^\tau \;+\; t^\tau\big(d^\tau - 2\,\mathbf{1}\big), \qquad d_i^\tau \;=\; \deg_{T_{\pi^\tau}}(i), \tag{32}$$

with Polyak-style step size $t^\tau > 0$, until d^τ stabilises near $2\,\mathbf{1}$. ASCENT is invoked exactly once per instance and dominates pre-search compute on large N. The post-ascent statistic

$$\alpha^{\pi^\star}(i,j) \;=\; L\big(T^{+}_{\pi^\star}(i,j)\big) \;-\; L(T_{\pi^\star}) \tag{33}$$

is what LKH actually uses for candidate construction, and $\tilde{w}(\pi^\star)$ is the cost matrix carried inside k-opt.

Candidate measure and trial loop. The geometric candidate measure of vertex i is the top-K slice of $-\alpha^{\pi^\star}$:

$$\mathcal{C}^{\mathrm{geo}}(i) \;=\; \operatorname*{argtop}_{K}\big\{-\alpha^{\pi^\star}(i,\cdot)\big\}, \tag{34}$$

sorted in ascending $\alpha^{\pi^\star}$. This set serves a dual role inside k-opt: it is the only family from which y-edges may be drawn (filter), and its ordering is the priority along which the depth-first recursion expands children (search direction). Both $\pi^\star$ and $\mathcal{C}^{\mathrm{geo}}$ are deterministic functions of $\mathcal{G}$ and remain fixed for the rest of the run.

The outer driver launches B_{trials} trials. Trial $r{=}1$ starts from a tour $\mathcal{R}$ generated greedily under $\mathcal{C}^{\mathrm{geo}}$, runs sequential k-opt to a local optimum, and updates the global incumbent $\mathcal{R}$. Each subsequent trial perturbs the current incumbent by a non-sequential 4-opt-like segment swap (the LKH kick) and re-optimises. Vanilla LKH terminates once the trial budget is exhausted; the procedure is summarised in Algorithm 2(a). Crucially, $\mathcal{C}^{\mathrm{geo}}$ is the only channel through which any prior knowledge of likely-optimal edges enters search, and it never reacts to what search has discovered, an opening that LSSL exploits next.

H.2 Guided-LKH

LSSL retains every component of Algorithm 2(a), namely the Held–Karp dual $\pi^\star$, the reduced cost $\tilde{w}(\pi^\star)$, sequential k-opt with depth $k_{\max}{=}5$, the non-sequential kick, and the multi-trial loop, and modifies a single object: the static geometric candidate measure $\mathcal{C}^{\mathrm{geo}}$ of Eq. (34) is replaced by a trajectory-conditioned measure $\mathcal{C}^{(c)}$, $c = 0, 1, \ldots, C-1$, that the sparse diffusion Transformer D_θ of Section 2 re-emits at every LSSL cycle in response to the running incumbent. The slots inside k-opt and their priority semantics are unchanged; only the contents of those slots are refreshed.

At cycle c, LSSL projects the cycle-c incumbent $\mathcal{R}^{(c)}$ (the same object Algorithm 1 carries between D_θ and the search backend S) onto the K-NN sparse graph $\mathcal{G}_K$ via the binary mask $\mathcal{H}^{(c)} = P(\mathcal{R}^{(c)}; \mathcal{G}_K) \in \{0,1\}^{N\times K}$ of Section 2.2, where $\mathcal{H}^{(c)}_{ik}{=}1$ iff slot k at vertex i is one of the two tour neighbours of i in $\mathcal{R}^{(c)}$. D_θ ingests $(\mathcal{G}_K, \mathcal{H}^{(c)})$ along with the noise-axis schedule and emits a sparse, row-normalised edge-prior heatmap

$$\mathcal{H}^{(c)}_{\mathrm{guide}} \in [0,1]^{N\times K}, \qquad [\mathcal{H}^{(c)}_{\mathrm{guide}}]_{ik} \;\approx\; \Pr\big[\,(i, j_{ik}) \in \mathcal{R}^\star \,\big|\, \mathcal{R}^{(c)}\,\big], \tag{35}$$

where $j_{ik} = \mathcal{N}_K(i)[k]$ is the slot-to-node alias of Eq. (12) and $\mathcal{R}^\star$ denotes the optimal tour. The cycle-c candidate measure consumed by the inner k-opt recursion is the ordered concatenation

$$\mathcal{C}^{(c)}(i) \;=\; \mathcal{C}^{\mathrm{rsv}}(i) \,\circ\, \Big(\big\{j_{ik} \,:\, k \in \mathrm{argtop}_K[\mathcal{H}^{(c)}_{\mathrm{guide}}]_{i,\cdot}\big\} \setminus \mathcal{C}^{\mathrm{rsv}}(i)\Big), \tag{36}$$

of total width K. The argtop_K ordering inherits the priority role that $\alpha^{\pi^\star}$ played for $\mathcal{C}^{\mathrm{geo}}$, so k-opt visits the model's highest-confidence neighbours first. The reserved prefix $\mathcal{C}^{\mathrm{rsv}}(i) \subseteq \mathcal{V}\setminus\{i\}$ collects slots that must remain in $\mathcal{C}^{(c)}(i)$ regardless of $\mathcal{H}^{(c)}_{\mathrm{guide}}$; by default we populate it with the top heatmap slots from both the per-cycle and the cycle-averaged predictions, so that consensus edges across symmetry views (e.g. rotation and reflection augmentations of $\mathcal{G}$) are not displaced by within-cycle stochasticity. Geometric variants with $\mathcal{C}^{\mathrm{rsv}} \subseteq \mathcal{C}^{\mathrm{geo}}$ are equally admissible. The remaining slots are filled by the highest-scoring model neighbours.

The replacement of $\mathcal{C}^{\mathrm{geo}}$ by $\mathcal{C}^{(c)}$ is more than a candidate-set swap.

(i) Time variance: $\mathcal{C}^{(c)}$ depends on c and, through $\mathcal{R}^{(c)}$, on the entire search prefix; setting $\mathcal{C}^{(c)} \equiv \mathcal{C}^{\mathrm{geo}}$ for all c collapses Algorithm 2(b) back to (a).

(ii) Cumulative warm-start: cycle c is launched from $\mathcal{R}^{(c)}$, so earlier improvements both bias the next heatmap $\mathcal{H}^{(c+1)}_{\mathrm{guide}}$ and seed k-opt inside the basin already discovered.

(iii) Amortised dual: $\pi^\star$ is computed by ASCENT at cycle 0 and then reused via a persistent LKH session that keeps the reduced cost $\tilde{w}(\pi^\star)$ resident across cycles; only $\mathcal{C}^{(c)}$ is refreshed (concretely, after cycle 0 writes $\pi^\star$ to a `PI_FILE`, subsequent cycles consume it together with the new `CANDIDATE_FILE` and bypass ASCENT entirely). Guided-LKH thus preserves the Held–Karp dual setup and the sequential k-opt admissibility of vanilla LKH; the closed loop only changes which edges k-opt examines, through the dataflow $\mathcal{R}^{(c)} \to \mathcal{H}^{(c)}_{\text{guide}} \to \mathcal{C}^{(c)} \to \mathcal{R}^{(c+1)}$ shown in Algorithm 2(b).

Algorithm 2 Vanilla LKH (top) vs. Guided-LKH (bottom).
REUSED: dual $\pi^\star$, reduced cost $\tilde{w}(\pi^\star)$, k-opt admissibility, kick, multi-trial loop.
REPLACED: static geometric $\mathcal{C}^{\text{geo}}$ becomes trajectory-conditioned $\mathcal{C}^{(c)}$.
CLOSED LOOP: $\mathcal{R}^{(c)} \to \mathcal{H}^{(c)}_{\text{guide}} \to \mathcal{C}^{(c)} \to \mathcal{R}^{(c+1)}$.

(a) Vanilla LKH.

Require: graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, w)$, candidate width K, trial budget B_{trials}
Ensure: best tour $\bar{\mathcal{R}}$ found across all trials

compute dual $\pi^\star \leftarrow \arg\max_\pi w(\pi)$ via ASCENT ▷ Eqs. (31),(32)
build static candidate measure $\mathcal{C}^{\text{geo}} = \{\mathcal{C}^{\text{geo}}(i)\}_{i \in \mathcal{V}}$ ▷ Eqs. (33),(34)
$\bar{\mathcal{R}} \leftarrow \perp$
for $r = 1, \ldots, B_{\text{trials}}$ **do**
  if $r{=}1$ **then** initialise $\mathcal{R}$ greedily under $\mathcal{C}^{\text{geo}}$; **else** kick: $\mathcal{R} \leftarrow \bar{\mathcal{R}}$ and apply a non-sequential segment swap
  repeat
    perform one k-opt sweep over $\mathcal{V}$ under $\tilde{w}(\pi^\star)$ with candidate measure $\mathcal{C}^{\text{geo}}$, accepting the first improving move
  until no improving move is found in a full sweep
  if $\bar{\mathcal{R}} = \perp$ **or** $\text{cost}(\mathcal{R}) < \text{cost}(\bar{\mathcal{R}})$ **then** $\bar{\mathcal{R}} \leftarrow \mathcal{R}$
  end if
end for
return $\bar{\mathcal{R}}$

(b) Guided-LKH (LSSL). blue block = line differs from (a); ● = structurally replaced symbol.

Require: graph $\mathcal{G}$, sparse diffusion Transformer D_θ, candidate width K, number of cycles C, trial budget per cycle B_{trials}, denoising schedule $\{t_r\}_{r=0}^{T}$ with $t_0 = 1$, $t_T = 0$
Ensure: best tour $\bar{\mathcal{R}}$ found across all cycles

compute dual $\pi^\star \leftarrow \arg\max_\pi w(\pi)$ via ASCENT ▷ Eq. (31); amortised: solved once at cycle 0, then reused
build static candidate measure $\mathcal{C}^{\text{geo}}$ ▷ geometric reference, retained as fallback
$\bar{\mathcal{R}} \leftarrow \perp$, $\mathcal{R}^{(0)} \leftarrow \perp$
for $c = 0, 1, \ldots, C-1$ **do**
  $\mathcal{H}^{(c)} \leftarrow P(\mathcal{R}^{(c)}; \mathcal{G}_K)$ if $\mathcal{R}^{(c)} \neq \perp$ else random $N \times K$ init ▷ condition on incumbent prefix
  $\mathcal{H}^{(c)}_{\text{guide}} \leftarrow \text{DENOISESCHEDULE}(D_\theta, \mathcal{G}_K, \mathcal{H}^{(c)}; \{t_r\}_{r=0}^{T})$ ▷ Alg. 1 lines 6–10; Eq. (35)
  form trajectory-conditioned candidate measure $\mathcal{C}^{(c)}$ from $\mathcal{H}^{(c)}_{\text{guide}}$ ▷ Eq. (36)
    ▷ only $\mathcal{C}^{(c)}$ refreshes per cycle
  for $r = 1, \ldots, B_{\text{trials}}$ **do**
    if $r = 1$ **and** $\bar{\mathcal{R}} \neq \perp$ **then**
      warm-start $\mathcal{R} \leftarrow \bar{\mathcal{R}}$ ▷ cumulative incumbent passes through cycle boundary
    else
      kick: perturb $\mathcal{R}$ by a non-sequential segment swap
    end if
    repeat
      perform one k-opt sweep over $\mathcal{V}$ under $\tilde{w}(\pi^\star)$ with candidate measure $\mathcal{C}^{(c)}$, accepting the first improving move
    until no improving move is found in a full sweep
    if $\bar{\mathcal{R}} = \perp$ **or** $\text{cost}(\mathcal{R}) < \text{cost}(\bar{\mathcal{R}})$ **then** $\bar{\mathcal{R}} \leftarrow \mathcal{R}$
    end if
  end for
  $\mathcal{R}^{(c+1)} \leftarrow \bar{\mathcal{R}}$ ▷ snapshot incumbent for next cycle
end for
return $\bar{\mathcal{R}}$

I HGS algorithm for CVRP

This appendix specifies how the Hybrid Genetic Search heuristic [44, 43] is integrated into the LSSL learning–search loop. Section I.1 formalises vanilla HGS around its granular neighbourhood measure, the sole channel through which structural prior information enters education and the SWAP* neighbourhood. Section I.2 introduces Guided-HGS, which retains the HGS search core but replaces this static, geometry-based measure with a trajectory-conditioned counterpart refreshed at every LSSL cycle. Algorithm 3 provides a line-level diff between the two main loops.

I.1 Vanilla HGS

Let a CVRP instance specify a customer set $\mathcal{V}_c$, a depot 0, distances c_{ij}, customer demands q_i, and a vehicle capacity Q. A solution (route set) $\mathcal{R}$ is encoded by a giant tour $\tau \in \mathrm{Sym}(\mathcal{V}_c)$ on which the linear-time SPLIT dynamic program [44]

$$f^{\phi}(\tau) \;=\; \min_{\sigma \in \Sigma(\tau)} \sum_{r \in \sigma} \Big(c(r) \;+\; \lambda_Q (Q(r) - Q)^{+} \Big) \tag{37}$$

recovers the unique cost-optimal capacity-feasible route segmentation, where $\Sigma(\tau)$ is the set of all segmentations of τ, $Q(r)$ is the load of route r, and λ_Q is a current capacity-penalty coefficient. HGS evolves a population $\mathcal{P} = \mathcal{P}_{\mathrm{feas}} \cup \mathcal{P}_{\mathrm{inf}}$ of giant tours under a biased fitness combining a cost rank with a diversity rank,

$$\Phi_{\mathcal{P}}(\mathcal{R}) \;=\; f^{\phi}_{\mathcal{P}}(\mathcal{R}) \;+\; \Big(1 - \tfrac{n_{\mathrm{elite}}}{|\mathcal{P}|}\Big) f^{\mathrm{div}}_{\mathcal{P}}(\mathcal{R}), \tag{38}$$

where $f^{\phi}_{\mathcal{P}}$ is the cost rank and $f^{\mathrm{div}}_{\mathcal{P}}$ the broken-pairs distance to the n_{closest} most similar individuals in $\mathcal{P}$; the $1 - n_{\mathrm{elite}}/|\mathcal{P}|$ weight preserves the top n_{elite} on quality alone. Capacity and duration violations V^Q, V^T are softly penalised by $\tilde{f} = f^{\phi} + \lambda_Q V^Q + \lambda_T V^T$, with λ_Q, λ_T periodically retuned toward a target feasible-ratio ξ^{ref} (controlled infeasibility). A single sparse object then dominates HGS, the granular neighbourhood measure

$$\Gamma^{\mathrm{geo}}(i) \;=\; \operatorname*{argtop}_{\gamma} \big\{ -c_{ij} : j \in \mathcal{V}_c \setminus \{i\} \big\}, \tag{39}$$

i.e. the γ spatial nearest neighbours of customer i, which cap each move neighbourhood at $\mathcal{O}(\gamma n)$ instead of $\mathcal{O}(n^2)$ and form the single channel through which prior structural information enters search: every classical move (RELOCATE, SWAP, 2-OPT, 2-OPT*) and the SWAP* neighbourhood of Vidal [43] restrict their partner customers to Γ^{geo}, and the 4μ random initial giant tours are educated by first-improvement local search under the same restriction. Each generation then draws two binary-tournament parents under $\Phi_{\mathcal{P}}$, recombines them by Ordered Crossover on the depot-free permutation, recovers feasibility via SPLIT, and re-educates the offspring; offspring still infeasible are REPAIRed at $10\times$ penalty with probability $1/2$.

I.2 Guided-HGS

LSSL leaves the HGS search core untouched (SPLIT (37), biased fitness (38), controlled infeasibility, and the SWAP* structural theorem of Vidal [43]), and replaces only the static geometric prior Γ^{geo} with a trajectory-conditioned counterpart refreshed at every cycle $c = 0, 1, \ldots, C-1$ by the sparse diffusion Transformer D_θ of Section 2, with three structural consequences.

(i) Dynamic neighbourhood prior. At cycle c, D_θ ingests the $N \times K$ projection $\mathcal{H}^{(c)} = P(\mathcal{R}^{(c)}; \mathcal{G}_K)$ of the cumulative-best route set $\mathcal{R}^{(c)}$ (the same carrier as in Algorithm 1, set to the best feasible solution found through cycle $c-1$) and emits a customer edge-prior heatmap $\mathcal{H}^{(c)}_{\mathrm{guide}} \in [0,1]^{|\mathcal{V}_c| \times K}$ with $[\mathcal{H}^{(c)}_{\mathrm{guide}}]_{ik} \approx \Pr[(i, j_{ik}) \in \mathcal{R}^{\star} \mid \mathcal{R}^{(c)}]$, where $\mathcal{R}^{\star}$ denotes the optimal route set. The granular measure driving every classical move and SWAP* is then

$$\widetilde{\Gamma}^{(c)}(i) \;=\; \Gamma^{(c)}_D(i) \circ \big(\Gamma^{\mathrm{geo}}(i) \setminus \Gamma^{(c)}_D(i) \big), \qquad \Gamma^{(c)}_D(i) \;=\; \big\{ j_{ik} : k \in \arg\operatorname{top}_{\gamma} [\mathcal{H}^{(c)}_{\mathrm{guide}}]_{i,\cdot} \big\}, \tag{40}$$

with model-ranked neighbours leading and Γ^{geo} as backfill. The substitution preserves every HGS invariant, including the $\Theta(N^3) \to o(N^2)$ SWAP* compression.

(ii) Dynamic boundary bias. At cycle c, the decoder's node-selection head additionally outputs a per-customer heatmap $\mathcal{H}^{(c)}_{\text{node}} \in [0,1]^{N_c}$ (Section F.2), with $[\mathcal{H}^{(c)}_{\text{node}}]_i$ modelling the probability that customer i anchors a route boundary. Its rank induces a node-boundary bonus

$$\delta^{(c)}_{\text{node}}(i) \;=\; 1 - r^{(c)}_{\text{node}}(i) \,/\, |\mathcal{V}_c| \;\in\; [0,1), \tag{41}$$

used as a tiebreaker on depot-attachment moves while leaving move evaluations unchanged.

(iii) Incumbent-carrying restart. The cumulative best $\mathcal{R}^{(c)}$ crosses cycle boundaries: it conditions $\mathcal{H}^{(c)}_{\text{guide}}$, enters $\mathcal{P}_{\text{feas}}$ as an elite when available, and seeds the rest of the population along $\widetilde{\Gamma}^{(c)}$, so each cycle resumes inside the basin already discovered. Setting $C=1$, $\widetilde{\Gamma}^{(0)} \equiv \Gamma^{\text{geo}}$, $\delta^{(0)}_{\text{node}} \equiv 0$ with random seeding recovers vanilla HGS.

Algorithm 3 Vanilla HGS (top) vs. Guided-HGS (bottom).
REUSED: SPLIT, biased fitness, OX, controlled infeasibility, survivor selection, SWAP*.
REPLACED: $\Gamma^{\text{geo}} \to \widetilde{\Gamma}^{(c)}$ plus node-boundary bonus $\delta^{(c)}_{\text{node}}$.
CLOSED LOOP: $\mathcal{R}^{(c)} \to \mathcal{H}^{(c)}_{\text{guide}} \to (\widetilde{\Gamma}^{(c)}, \mathcal{H}^{(c)}_{\text{node}}, \delta^{(c)}_{\text{node}}) \to \mathcal{R}^{(c+1)}$.

(a) Vanilla HGS.

Require: instance $\mathcal{G}$, capacity Q, granularity γ, total time budget
Ensure: incumbent feasible route set $\bar{\mathcal{R}}$

build static granular measure Γ^{geo} ▷ Eq. (39)
seed $\mathcal{P}$ with random giant tours, SPLIT (37), educate on Γ^{geo}
$\bar{\mathcal{R}} \leftarrow \arg\min_{\mathcal{R} \in \mathcal{P}_{\text{feas}}} f^{\phi}(\mathcal{R})$
while time budget not exhausted **do**
  binary-tournament parents P_1, P_2 by $\Phi_{\mathcal{P}}$ ▷ Eq. (38)
  $O \leftarrow \text{OX}(P_1, P_2)$; SPLIT; educate on Γ^{geo}; w.p. $1/2$ REPAIR at $10\times$ penalty if infeasible
  insert O into $\mathcal{P}_{\text{feas}}/\mathcal{P}_{\text{inf}}$; survivor selection on overflow (clones first, then by $\Phi_{\mathcal{P}}$)
  periodically adapt λ_Q, λ_T toward feasible-ratio target ξ^{ref} ▷ controlled infeasibility
  $\bar{\mathcal{R}} \leftarrow \arg\min_{\mathcal{R} \in \mathcal{P}_{\text{feas}} \cup \{\bar{\mathcal{R}}\}} f^{\phi}(\mathcal{R})$ ▷ update incumbent
end while
return $\bar{\mathcal{R}}$

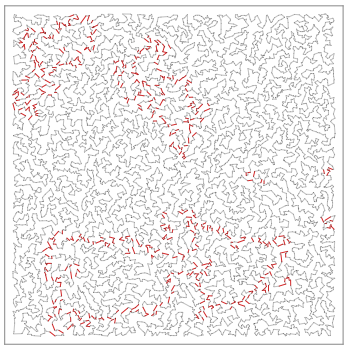
(b) Guided-HGS (LSSL). blue block = line differs from (a); ● = structurally replaced symbol.

Require: instance $\mathcal{G}$, capacity Q, sparse diffusion Transformer D_θ, granularity γ, cycles C, denoising schedule $\{t_r\}_{r=0}^{T}$ with $t_0 = 1$, $t_T = 0$
Ensure: incumbent feasible route set $\bar{\mathcal{R}}$

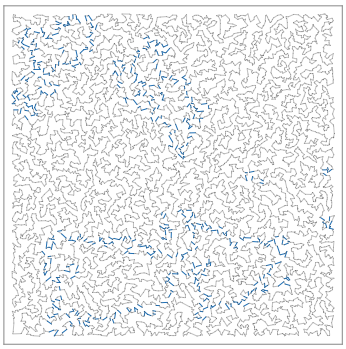
build static granular measure Γ^{geo} ▷ geometric backfill only
$\bar{\mathcal{R}} \leftarrow \perp$, $\mathcal{R}^{(0)} \leftarrow \perp$
for $c = 0, 1, \ldots, C-1$ **do**
  $\mathcal{H}^{(c)} \leftarrow P(\mathcal{R}^{(c)}; \mathcal{G}_K)$ if $\mathcal{R}^{(c)} \neq \perp$ else random $N \times K$ init
  $\mathcal{H}^{(c)}_{\text{guide}} \leftarrow \text{DENOISESCHEDULE}(D_\theta, \mathcal{G}_K, \mathcal{H}^{(c)}; \{t_r\}_{r=0}^{T})$ ▷ Alg. 1 lines 6–10
  form $\widetilde{\Gamma}^{(c)}, \mathcal{H}^{(c)}_{\text{node}}, \delta^{(c)}_{\text{node}}$ from $\mathcal{H}^{(c)}_{\text{guide}}$ ▷ Eqs. (40)–(41)
  re-seed $\mathcal{P}$ along $\widetilde{\Gamma}^{(c)}$, with $\mathcal{R}^{(c)}$ as elite if available
  while cycle-c time budget not exhausted **do**
    binary-tournament parents P_1, P_2 by $\Phi_{\mathcal{P}}$ ▷ Eq. (38)
    $O \leftarrow \text{OX}(P_1, P_2)$; SPLIT; educate on $\widetilde{\Gamma}^{(c)}$ ($\delta^{(c)}_{\text{node}}$ tiebreaks at depot); w.p. $1/2$ REPAIR at $10\times$ penalty if infeasible
    insert O into $\mathcal{P}_{\text{feas}}/\mathcal{P}_{\text{inf}}$; survivor selection on overflow (clones first, then by $\Phi_{\mathcal{P}}$)
    periodically adapt λ_Q, λ_T toward feasible-ratio target ξ^{ref} ▷ controlled infeasibility
    $\bar{\mathcal{R}} \leftarrow \arg\min_{\mathcal{R} \in \mathcal{P}_{\text{feas}} \cup \{\bar{\mathcal{R}}\}} f^{\phi}(\mathcal{R})$ ▷ update incumbent
  end while
  $\mathcal{R}^{(c+1)} \leftarrow \bar{\mathcal{R}}$ ▷ best-so-far carries to next cycle
end for
return $\bar{\mathcal{R}}$

J Solution Visualization

J.1 Solution Visualization on Synthetic TSP Instances



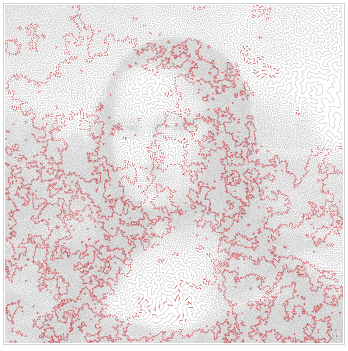
(a) Reference tour
(BKS, length 71.7031).



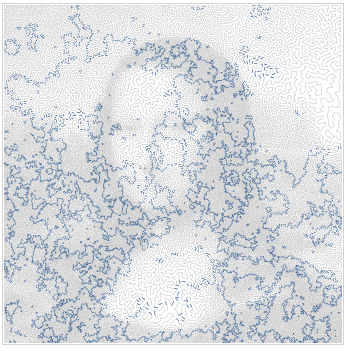
(b) LSSL tour
(C=2, B_{trials}=256, gap -0.0135%).

Figure 4: Solution visualization on a synthetic TSP10000 instance (sample #12, N=10000).

J.2 Solution Visualization on Art-TSP Instances



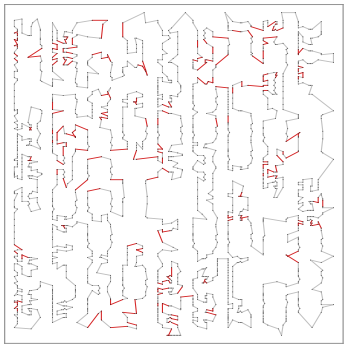
(a) Reference tour
(BKS, length 5,757,191).



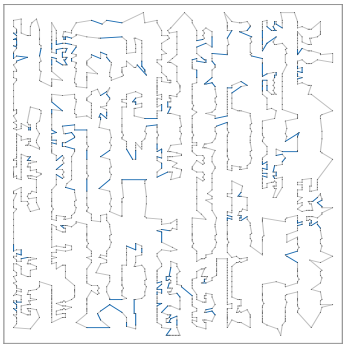
(b) LSSL tour
(C=4, B_{trials}=512, gap 0.026%).

Figure 5: Solution visualization on Art-TSP instance `mona-lisa100K` (N=100000).

J.3 Solution Visualization on VLSI Instances

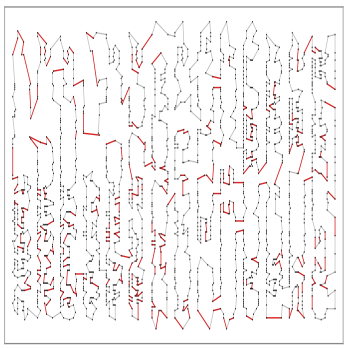


(a) Reference tour
(BKS, length 6,197).

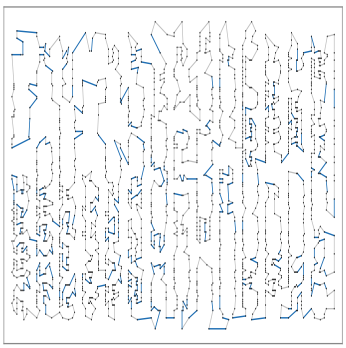


(b) LSSL tour
(C=2, B_{trials}=256, gap -0.387%).

Figure 6: Solution visualization on VLSI instance `djb2036` (N=2036).



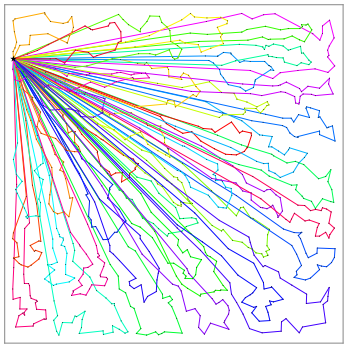
(a) Reference tour
(BKS, length 6,764).



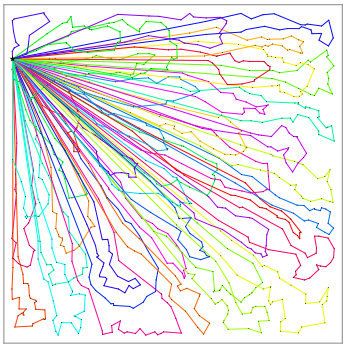
(b) LSSL tour
(C=2, B_{trials}=256, gap -0.407%).

Figure 7: Solution visualization on VLSI instance `bck2217` (N=2217).

J.4 Solution Visualization on CVRPLIB Instances

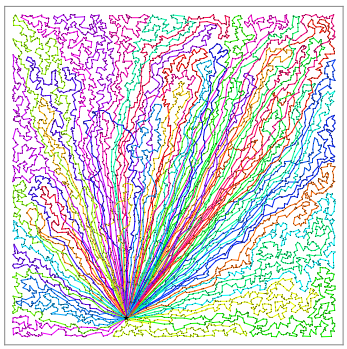


(a) Reference tour
(BKS, length 72,355).

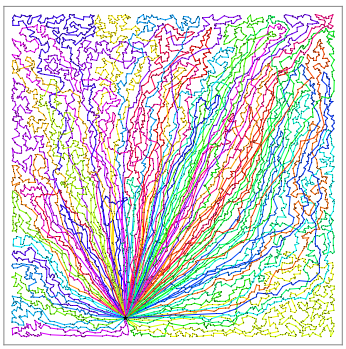


(b) LSSL tour
(C=4, B_t=128, gap 1.093%).

Figure 8: Solution visualization on CVRPLIB Set-X instance `X-n1001-k43` (N=1001, 43 routes).



(a) HGS tour
(B_t=8h, length 111,115).



(b) LSSL tour
(C=4, B_t=128, length 114,633, gap 3.166%).

Figure 9: Solution visualization on CVRPLIB Set-XL instance `XL-n9571-k55` (N=9,571, 55 routes). CVRPLIB only publishes the optimal cost (106,791) for this instance and not the corresponding tour, so we substitute a long-time HGS run (B_t=8h, no LSSL candidates) as the reference tour against which the LSSL gap is computed. The gap shown is therefore measured against this HGS-8h reference rather than the unattainable BKS tour.

K Broader impacts, Limitations and Future Work

Broader impacts. This research advances neural combinatorial optimization by introducing LSSL, a closed-loop learning–search framework for large-scale Vehicle Routing Problems (VRPs). We hope this framework will provide useful insight for future work on scalable, search-aware neural optimization. As a general methodology for solving VRPs, LSSL is not anticipated to create specific negative societal impacts beyond those already associated with optimization systems in logistics and routing.

Limitations and future work. The validated backends are LKH and HGS for TSP and CVRP; interface modularity does not demonstrate arbitrary solver or constraint generalization. Sparse candidate restriction can omit useful edges. Decomposition is complementary: a UDC-style hybrid and the paper-reported L2Seg comparison in Appendix A.4 indicate potential gains, but the latter uses different hardware and is not a matched reproduction. CVRP100K is a scalability stress test with memory-efficient HGS, not evidence of training at that scale. Reported aggregate gaps are not accompanied by repeated-run confidence intervals. Tighter model–backend co-design, broader constraint coverage, and controlled decomposition studies remain future work.

L Licenses

The licenses for the codes and the datasets used in this work are listed in Table 31.

Table 31: List of licenses for the codes and datasets we used in this work

Resource	Type	Link	License
LKH3 [6]	Code	http://webhotel4.ruc.dk/ keld/research/LKH-3/	Available for academic research use
HGS [43]	Code	https://github.com/chkwon/PyHygese	MIT License
Concorde [89]	Code	https://github.com/jvkersch/pyconcorde	BSD 3-Clause License
POMO [12]	Code	https://github.com/yd-kwon/POMO	MIT License
BQ [14]	Code	https://github.com/naver/bq-nco	CC BY-NC-SA 4.0
LEHD [13]	Code	https://github.com/CIAM-Group/NCO_code/tree/main/single_objective/LEHD	MIT License
INViT [28]	Code	https://github.com/Kasumigaoka-Utaha/INViT	Available for academic research use
L2C-Insert [21]	Code	https://github.com/CIAM-Group/NCO_code/tree/main/single_objective/L2C-Insert	MIT License
Att-GCN+MCTS [30]	Code	https://github.com/SaneLYX/TSP_Att-GCRN-MCTS	MIT License
DIFUSCO [32]	Code	https://github.com/Edward-Sun/DIFUSCO	MIT License
T2T [33]	Code	https://github.com/Thinklab-SJTU/T2TCO	MIT License
Fast-T2T [34]	Code	https://github.com/Thinklab-SJTU/Fast-T2T	MIT License
GenSCO [35]	Code	https://github.com/Thinklab-SJTU/GenSCO	MIT License
H-TSP [22]	Code	https://github.com/Learning4Optimization-HUST/H-TSP	MIT License
GLOP [25]	Code	https://github.com/henry-yeh/GLOP	MIT License
UDC [26]	Code	https://github.com/CIAM-Group/NCO_code/tree/main/single_objective/UDC-Large-scale-CO-master	MIT License
NeuroLKH [38]	Code	https://github.com/liangxinedu/NeuroLKH	Available for academic research use
TSPLib [45]	Dataset	http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/	Available for any non-commercial use
VLSI [46]	Dataset	http://www.math.uwaterloo.ca/tsp/vlsi/	Available for academic research use
Art-TSP [47]	Dataset	http://www.math.uwaterloo.ca/tsp/data/art/	Available for academic research use
CVRPLib [48]	Dataset	http://vrp.galgos.inf.puc-rio.br/index.php/en/	Available for academic research use