COMPUTE TIME SCALING WITH RECURSIVE MODELS FOR COMBINATORIAL OPTIMIZATION

Anonymous authors
Paper under double-blind review

ABSTRACT

We propose Tiny Recursive Models for Combinatorial Optimization (TRM4CO), a general neural method for combinatorial optimization that scales both depth (how often we recursively invoke our network) and width (how much we sample in parallel). Both are fundamental for combinatorial optimization: hard instances demand a large amount of compute, while a small network is essential to avoid overfitting and capture the algorithmic essence of optimization. In particular, our method consists of a graph-aware tiny recursive model that iterates on a latent state with adaptive halting and needs only a lightweight problem-specific decoder. Compared with previous heatmap-based general neural solvers, it achieves a better balance between solution quality and inference speed on both the Traveling Salesman Problem (TSP) and the Maximum Independent Set (MIS) problem, and remains competitive with hybrid methods that combine neural components with heuristics specific to each problem. With the same backbone architecture for both tasks, TRM4CO outperforms every diffusion-based solver on TSP from 500 to 10,000 cities at a lower inference cost, and on the standard Erdős–Rényi-[700-800] MIS benchmark it surpasses all neural solvers except those that only work well on MIS. We then explore self-relabeling for self-supervised training. We periodically replace the current set of training labels with the model's own better solutions, as an alternative training signal. Self-relabeling can, while forgoing supervision from near-optimal solutions, still result in on-par quality.

1 INTRODUCTION

Deep learning for combinatorial optimization faces a mismatch. Most neural networks have large numbers of parameters, yet do relatively little computation per parameter. Combinatorial optimization algorithms, on the other hand, are comparatively short programs that perform a lot of computation. It is an open problem to design architectures with large and deep computational graphs, which give the representational capacity that optimization demands, from relatively few parameters, which keep them fast and prevent overfitting. One answer to this is compute time scaling. To this end, we bring current advances in recursive deep learning architectures to neural combinatorial optimization.

Specifically, we propose TRM4CO, Tiny Recursive Model for Combinatorial Optimization. Our backbone is the Tiny Recursive Model (TRM, Jolicoeur-Martineau, 2025), a two-block Transformer with a few million parameters that is applied repeatedly to a latent state and an evolving answer. It is trained with deep supervision to improve the answer at every step. TRM4CO scales the test-time compute of the TRM recursion along two axes (Figure 1): along depth, *i.e.* the number of times the network is invoked, and along width, *i.e.* the number of solutions it samples in parallel. The network trains either on solver labels or without any, in a self-supervised loop that samples solutions, keeps those that improve on the stored ones, trains on them, and repeats this hill-climbing.

Existing work on neural combinatorial optimization scales compute differently. Autoregressive constructive policies (Vinyals et al., 2015; Kool et al., 2019; Kwon et al., 2020; Luo et al., 2023; Drakulić et al., 2023) grow a solution one node at a time, which is accurate but sequential, so at thousands of nodes they rely in practice on decomposition (Ye et al., 2024; Zheng et al., 2024; Li et al., 2025). Heatmap-based solvers instead predict all decision variables at once and hand the rest to a search. Among them diffusion solvers (Sun & Yang, 2023; Li et al., 2023; 2024; Zhao et al., 2025; Feng et al., 2026b) also iterate, but along a fixed noise schedule. The variants that scale test-time compute

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

do so in heatmap space (Li et al., 2023; 2024), where a search is still required to obtain a solution. Other heatmap solvers spend the extra compute around the network rather than in it, by fine-tuning a meta-learned policy on each instance and sampling massively in parallel (Qiu et al., 2022), or by learning an optimizer whose iterations update the heatmap (Dernedde et al., 2025). Earlier approaches train a graph neural network (GNN) as a supervised edge classifier (Joshi et al., 2019; Fu et al., 2021) and are no longer competitive on their own.

We benchmark on the Maximum Independent Set (MIS) problem and the Traveling Salesman Problem (TSP), two prototypical combinatorial optimization problems in the literature. To make recursive models work on both, we augment them with (i) an embedding of the problem structure (adjacency matrices modulating attention via a learnable per-head bias for MIS, embedded coordinates for TSP), (ii) lightweight problem-specific decoders (peeling for MIS, edge insertion with 2-opt for TSP), and (iii) halting and selection driven by the decoded solutions. The network size stays constant by design, so additional compute goes into iterations rather than parameters.

Empirically, TRM4CO reaches the lowest optimality gap of all learned solvers on TSP-500 and TSP-1000, *e.g.* 0.11% on TSP-500 in 2.1 seconds per instance, and matches the best diffusion solver on TSP-10000. On MIS, solution quality grows with test-time compute, and with 200 rollouts TRM4CO surpasses every general-purpose neural solver and the KaMIS reference on Erdős–Rényi (ER) graphs with 700–800 nodes. Finally, self-relabeling raises 16% of the training labels above those of KaMIS and yields our best MIS checkpoint. Without any solver labels, the same loop trains an MIS model from scratch whose own labels match or beat KaMIS on 42% of the training graphs and whose test gap comes close to that of our best checkpoint (Section 3.6).

In summary, our contributions are as follows:

- We propose a deep recursive architecture for combinatorial optimization that can efficiently scale along depth and width.
- Our method can be trained in a supervised way from solver labels or in a self-supervised manner by self-relabeling, *i.e.* generating multiple labels and hill-climbing on the best ones.
- We show experimentally on MIS and TSP that we match or outperform state-of-the-art neural combinatorial optimization approaches.

2 METHOD

At a high level, our method works as follows (Figure 1). The encoder turns a problem instance, which comes with a set of feasible solutions and an objective, into tokens. A problem-agnostic recursive backbone then updates a latent state and an answer state iteratively. We scale the compute of the backbone along depth, the number of steps, and along width, the number of rollouts. We also inject noise into the latents for diversity. Periodically, a decoder reads the answer state and turns it into a solution. At inference, we keep the best solution over all readings across depths and rollouts. Optionally, during training, we replace the label of an instance whenever we sample a better solution.

We explain all components in detail below.

2.1 BACKGROUND: TINY RECURSIVE MODEL (TRM)

We use the Tiny Recursive Model (Jolicoeur-Martineau, 2025) as the backbone. It is a simplification of the Hierarchical Reasoning Model (HRM, Wang et al., 2025). TRM keeps two states of the same shape as the input tokens e. The latent z carries the reasoning and the answer y carries the current prediction. As shown in Figure 4 (Appendix A), one weight-tied network f_θ of two Transformer blocks updates both states in cycles:

$$\begin{aligned} z &\leftarrow f_\theta(z + y + e) && (L \text{ times}), \\ y &\leftarrow f_\theta(z + y) && (\text{once}), \end{aligned} \tag{1}$$

and one recursion step runs H such cycles, of which only the last carries gradient. Deep supervision chains up to D recursion steps, carrying the states (y, z) detached from one step to the next and applying the loss to the output head $g_\theta(y)$ after each step. Training memory is thus constrained to one cycle, while the effective depth is $2H(L+1)D$ blocks. A learned halting head on y decides during training when an instance stops recursing (Graves, 2016). This acts as hard mining, because

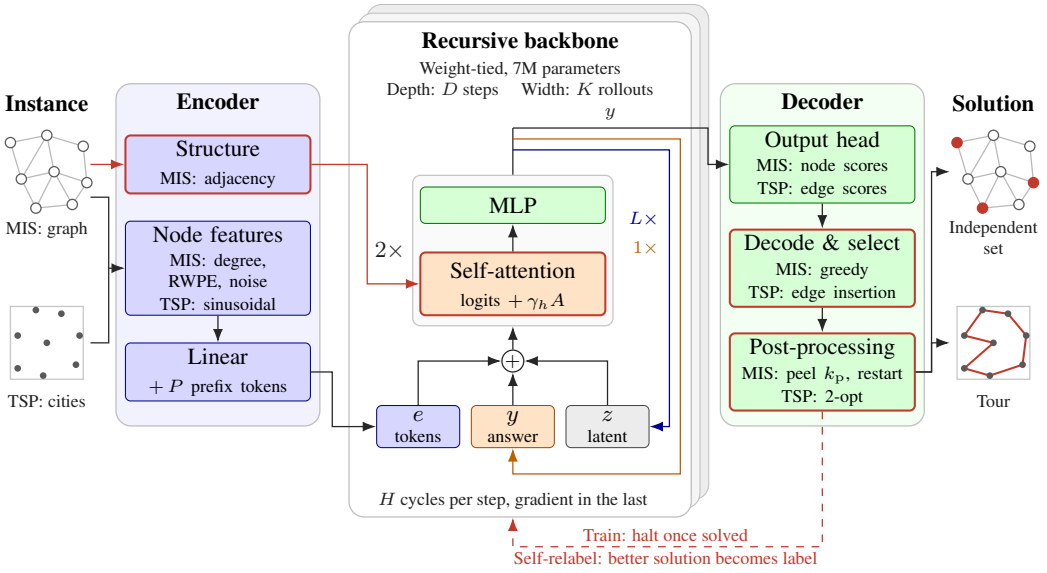


Figure 1: TRM4CO overview. An encoder turns an instance into node tokens plus P prefix tokens and, for MIS, the adjacency matrix A that biases every attention call of the shared recursive backbone (Eq. 2). Each step runs one TRM recursion, and test-time compute scales along depth D, the number of steps, and width K, the number of parallel rollouts. A problem-specific decoder turns the answer state of every step into a feasible solution, with sequential peeling for MIS and edge insertion with 2-opt for TSP, and the objective keeps the best candidate. In training, a solved check against the label halts an instance, and better solutions can replace the labels. Red marks what we add to TRM.

easy instances leave early and the remaining steps go to hard ones. Adding noise to z at test time and running several copies in parallel yields extra samples (Sghaier et al., 2026).

2.2 The TRM4CO framework

Input embedding and encoder. The encoder maps an instance x to n node tokens in $\mathbb{R}^d$ and, for MIS, passes on the adjacency matrix $A \in \{0,1\}^{n\times n}$ (Table 4). P learnable prefix tokens are prepended and serve as global registers, *i.e.* tokens attached to no node that every node can attend to, and we write e for the resulting $P+n$ tokens. A is zero-padded on the prefix rows and columns, so those registers receive no bias. No global positional encoding over the token order is used for either problem, so the model is permutation equivariant by construction. For MIS, the node tokens are a linear projection of the degree, a random-walk positional encoding (RWPE, Dwivedi et al., 2022), and a uniform random scalar that breaks the symmetry between structurally identical nodes. The edges enter the recursion through A. For TSP, the geometry is entirely in the coordinates X. The TSP node tokens are a linear projection of X lifted with sinusoidal features at L_{pe} octaves per axis (Mildenhall et al., 2020), and no attention bias is needed. This asymmetry is deliberate. The graph structure of MIS has to be injected into attention, whereas for TSP on the unit square, the coordinates already determine locality and the model can recover it via standard self-attention.

Recursive core with edge-biased attention. The core of our model is the two-block network of Eq. (1) with one change. Every attention head h of every call computes

$$\text{softmax}\left(\frac{\mathbf{Q}_h\mathbf{K}_h^\top}{\sqrt{d_h}} + \gamma_h A\right), \tag{2}$$

where $\mathbf{Q}_h$ and $\mathbf{K}_h$ are the queries and keys of head h with dimension d_h, and γ_h is a learnable scalar per head and block, initialized at zero. A is the adjacency matrix of the instance, with $A_{ij} = 1$ when nodes i and j are neighbors and 0 otherwise, so the bias raises or lowers the attention between adjacent nodes. With $\gamma_h = 0$ the head is standard self-attention, as $\gamma_h \to \infty$ it attends only to neighbors, and with $\gamma_h < 0$ it is biased toward non-neighbors. Each head therefore learns its own

mixture of local and global attention (Ying et al., 2021) at the cost of one scalar per head. For MIS, this stands in for a separate message-passing GNN encoder (Table 7). The recursive core has about 7M parameters, and its size does not change between problems.

Depth and width. Depth D is the number of recursion steps. Since the output head g_θ is applied after every step, one run of depth D yields D candidate solutions. Width K perturbs the latent with Gaussian noise $z \leftarrow z + \sigma\xi$, $\xi \sim \mathcal{N}(0, I)$, at every step (Sghaier et al., 2026) and runs K copies of the instance in one batched forward. Every candidate is feasible by construction, and its objective c is evaluated, so the best of them is selected by exact ranking. The same selection procedure can also produce training labels (Section 2.3). Solution quality improves along both axes, and the cost is linear in $K \times D$. Both axes therefore scale compute time on the same weights, with no additional parameters or training.

Decoding and selection. An output head g_θ reads the answer state. For MIS, the head is a per-node two-way classifier, and the difference between a node's positive and negative logits is its score. For TSP, the head computes a bilinear successor score $g_{ij} = \langle W_q y_i, W_k y_j \rangle / \sqrt{d_s}$ over ordered node pairs, with d_s the dimension of the projected vectors. A lightweight decoder turns the scores into a feasible solution, and the objective c evaluates it for selection. For MIS, the per-step decoder is greedy and also serves as the solved check in training (Section 2.3). Nodes are scanned in order of decreasing score, and a node is added whenever none of its neighbors has been picked. Optionally, Gumbel noise of temperature τ perturbs the scores. We wrap the recursion in *sequential peeling*. After D steps, up to k_{p} nodes are committed, taken in order of decreasing score with conflicting nodes skipped. The closed neighborhoods of the committed nodes are masked, and the recursion restarts on the residual graph until it is empty. Committing k_{p} nodes each time costs about $|S|/k_{\mathrm{p}}$ rounds of D steps, with $|S|$ the final set size, and improves the solution substantially over one-shot decoding (Table 7). We decode TSP by greedy edge insertion as in Sun & Yang (2023). Candidate edges are ranked by $\bar{g}_{ij}/d_{ij}$, where $\bar{g}_{ij}$ is the symmetrized successor probability from a row-wise softmax of g and d_{ij} the Euclidean distance. They are inserted in rank order while respecting the degree and subtour constraints, until a tour is formed. The tour selected at the end is repaired once with 2-opt.

2.3 Training and inference

Training is deep supervision over a batch of instances. Each slot of the batch carries one instance across steps and is refilled once that instance halts. Each step applies a cross-entropy loss between $g_\theta(y)$ and the label, per node for MIS and per successor for TSP. Inference runs D recursion steps on K noisy copies, decodes and scores after every step, and returns the best-scoring solution. A problem-specific wrapper completes the result. For TSP, it repairs the obtained tour with 2-opt. For MIS, it wraps the recursion in sequential peeling, which replaces the per-step selection. Each of the K copies peels independently, and the objective compares the K finished sets at the end. Algorithm 1 in Appendix A collects these procedures, which MIS and TSP share, with the problem-specific hooks of Table 4 plugged in.

Halting on a solved check. TRM trains its halting head to predict whether the current answer matches the label token by token, and halts on the head's decision. We drop this signal in favor of the problem-specific check in Table 4 and halt on it directly, without a learned head. For MIS, the greedy decoding of the current scores must reach at least the label size, which credits any of the many optima a graph may have. For TSP, the label tour is oriented counterclockwise so that every city has a unique successor. The check requires the predicted successors to agree with the label exactly. An instance halts when the check passes or when it exhausts its budget of D steps. The depth an instance receives is therefore adaptive during training. At test time, the depth is set explicitly by D, and selection by the objective c takes the place of the halting decision. Since every step applies the same set of weights, evaluation may run deeper than training.

Self-relabeling. Optionally, we let the model relabel its own training set. Every T epochs, the model solves every training instance, and the solution replaces the current label whenever it is strictly better, *i.e.* a larger independent set or a shorter tour. The labels therefore only ever improve. The same loop can also train from random initialization without any external solver. One pass of self-relabeling with the untrained model seeds every label before training starts, and every later pass tightens the labels. Section 3.6 reports the results for the self-supervised settings.

3 EXPERIMENTS

3.1 SETUP

Benchmarks. We follow the training and test data splits of DIFUSCO (Sun & Yang, 2023). For MIS, these are Erdős–Rényi (ER) graphs (Erdős & Rényi, 1960) with 700 to 800 nodes and edge probability 0.15 (ER-[700-800], 16,384 training and 128 test graphs), and graphs obtained by reducing the CNF formulas of SATLIB (Hoos & Stützle, 2000) to MIS, with up to 1347 nodes (39,500 training and 500 test graphs). Both benchmarks come from Qiu et al. (2022). For TSP, we generate 128,000, 64,000 and 6,400 training instances with 500, 1000 and 10000 cities respectively, uniformly distributed in the unit square, and we adopt the TSP-500/1000/10000 test sets of Fu et al. (2021) with 128, 128 and 16 instances. We compute all training labels ourselves, with KaMIS (Lamm et al., 2017; Hespe et al., 2019) at a 10-second budget per graph for ER, Gurobi (Gurobi Optimization, LLC, 2026) for SATLIB, and Concorde (Applegate et al., 2006) or LKH-3 (Helsgaun, 2017) for TSP (Appendix C). The training setup is detailed in Appendix B.

Baselines. Tables 1 and 2 divide the compared methods into three groups: general-purpose neural solvers that report results on both MIS and TSP (heatmap, diffusion and divide-and-conquer solvers), learning-free methods (samplers and heatmap-free search), and neural solvers specific to MIS or TSP. Classical solvers (KaMIS, Concorde and LKH-3) serve as references.

Metrics and runtime. For MIS, we report the mean independent set size and the optimality gap to KaMIS. For TSP, we report the mean tour length and the gap against Concorde at $n = 500$ and 1000 and against LKH-3 at $n = 10000$, as in DIFUSCO and its successors. Quality numbers are taken from the original papers, except for the reruns listed in Appendix E. We measure only the runtime, on one RTX 4090 with each method's released code and checkpoints, from input to final solution (Appendix D). The reference solvers are listed with the CPU times reported in the original papers.

3.2 MAIN RESULTS ON MIS

Table 1 compares quality and cost. On ER-[700-800], the learned solvers span a wide range. The supervised diffusion solvers have the highest gaps, the recent RL and unsupervised solvers improve on them, and Moco matches the reference at the price of hundreds of heatmap-search steps per graph. TRM4CO runs the same weights at every depth and width. At a moderate width it sits among the strongest single-model solvers, and with more rollouts it passes Moco and the KaMIS reference, as the only learned solver without a sampler to do so. The learning-free samplers reach slightly lower gaps still. RLSA, alone and on top of COExpander, dominates on ER graphs, which confirms the observation of Sun et al. (2023) that these graphs reward long search more than learning. TRM4CO closes most of that distance with a general recipe, but at a much higher cost, since peeling reruns the recursion about $|S|/k_{\mathrm{p}}$ times. SATLIB shows the recursion without peeling. All neural solvers are within 1% of KaMIS there, and TRM4CO at $K = 10$ with one-shot decoding matches the reference and solves almost every instance exactly, in the cost band of the diffusion solvers with sampling.

3.3 MAIN RESULTS ON TSP

Table 2 compares gaps and cost at 500, 1000, and 10000 cities. At $n = 500$ and 1000, TRM4CO has the lowest gap of all learned solvers, at a cost within the band of diffusion solvers. Search matters in these pipelines. DIFUSCO with MCTS improves on its 2-opt variant only through the search, and the heatmap-free controls of Pan et al. (2026) land in the same range under MCTS, at more than $20\times$ our cost per instance. Learning-free methods fare worse on TSP than on MIS, where samplers alone beat the KaMIS reference (Table 1). An ER graph has many maximum independent sets, whereas a TSP instance on the unit square almost surely has a unique optimal tour, so a search without learned guidance has many targets on MIS, but only a single one on TSP. Among the TSP-specific solvers, the closest to our method at $n = 500$ is LEHD with random re-construction, at a higher gap and more than $20\times$ our cost, and at $n = 1000$ it is DRHG with $10\times$ the cost. At $n = 10000$, TRM4CO matches COExpander and attains a lower gap than every other diffusion solver. A considerable gap remains between these general neural solvers and the hybrid methods that run Lin–Kernighan or destroy-and-repair search inside their loops. These methods reach lower gaps at a lower cost.

Table 1: MIS results. We list mean independent set size on ER-[700-800] and SATLIB, gap in % with respect to the KaMIS reference, and seconds per instance on one RTX 4090. Type: S = supervised, U = unsupervised, RL = reinforcement learning (details in Appendix E). "–" means that the number is not available, and "†" marks evaluation on the method's own instances from the same distribution. The KaMIS time is measured on CPU. Bold marks the lowest gap on each benchmark.

Dataset		ER-[700-800]			SATLIB		
Method	Type	Size	Gap %	s/inst	Size	Gap %	s/inst
KaMIS (reference)	heuristic	44.87	0.00	24	425.96	0.00	4.5
General-purpose neural solvers (MIS and TSP)							
DIMES (Qiu et al., 2022)	RL	42.06	6.26	–	423.28	0.63	–
DIFUSCO (Sun & Yang, 2023)	S	41.12	8.36	7.4	425.13	0.19	1.6
T2T (Li et al., 2023)	S + search	41.37	7.80	9.0	425.22	0.17	1.5
UDC (Zheng et al., 2024)	RL	42.88	4.44	6.9	–	–	–
Fast T2T (Li et al., 2024)	S + search	41.73	7.00	0.92	425.23	0.17	1.4
Moco (Dernedde et al., 2025)	U + search	44.87	0.00	22.8	425.02	0.22	109
DISCO (Zhao et al., 2025)	S	42.21	5.93	–	425.06	0.21	–
COExpander (Ma et al., 2025)	S	42.56	5.15	1.2	425.28	0.16	1.3
COExpander + RLSA	S + sampler	**45.27**	**-0.89**	4.5	425.45	0.12	8.6
CADO (Song et al., 2026)	S + RL	43.62	2.79	–	425.43	0.12	–
CAM (Feng et al., 2026b)	U	44.16	1.58	0.2	–	–	–
HyCO (Li et al., 2026)	S + RL	42.38†	5.55†	–	425.40†	0.13†	–
Learning-free methods							
iSCO (Sun et al., 2023)	sampler	45.15	-0.62	55	424.16	0.42	87
ReSCO (Li & Zhang, 2025)	sampler	45.24	-0.82	73	424.21	0.41	258
RLSA (Feng & Yang, 2025)	sampler	45.05	-0.40	0.65	–	–	–
MIS-specific neural solvers (no TSP results)							
LwD (Ahn et al., 2020)	RL	41.17	8.25	–	424.80†	0.27†	–
GFlowNets (Zhang et al., 2023)	U	41.14†	8.31†	–	423.54†	0.57†	–
DiffUCO (Sanokowski et al., 2024)	U	43.98	1.98	–	–	–	–
SDDS (Sanokowski et al., 2025)	U	43.31	3.48	–	–	–	–
RLNN (Feng & Yang, 2025)	U	43.34	3.41	0.26	–	–	–
DDEA (Salvà Soler & Raidl, 2026)	S + evolution	44.13	1.65	396	–	–	–
TRM4CO (general-purpose; noise σ=0.2; peel k_{p}=2, Gumbel noise τ=0.5 on ER; one-shot on SATLIB)							
TRM4CO (K=10, D=32)	S + self	44.37	1.11	18.7	**425.96**	**0.00**	2.2
TRM4CO (K=100, D=64)	S + self	44.93	-0.13	368	–	–	–
TRM4CO (K=200, D=32)	S + self	45.04	-0.38	373	–	–	–

3.4 Scaling test-time compute

Figure 2 shows the contribution of the two scaling axes. On ER-[700-800], width matters more than depth. At $k_{\mathrm{p}} = 8$ the gap falls steadily with K, while the curves for $D \in \{16, 32\}$ lie on top of each other. Finer peeling helps most. With $k_{\mathrm{p}} = 2$ the gap is lower at every K and crosses the KaMIS reference at both depths, since every peel round takes the best of K fresh rollouts on the residual graph. Depth adds a small but consistent gain at $k_{\mathrm{p}} = 2$. On TSP the gap after 2-opt falls with K at both depths, and on TSP-1000 it is still falling at the largest width we ran. Overall cost is linear in $K \times D$ because the rollouts run as one batched forward. Figure 3 places these points among the baselines. On TSP-500 the TRM4CO curve lies below every learned solver. On ER-[700-800] it reaches the lowest gap of the learned solvers without a sampler, but RLSA and COExpander with RLSA reach the same or lower gaps at a small fraction of the cost, so there the samplers dominate.

3.5 Ablations

Table 7 in Appendix F varies one factor at a time. First, the structure of a combinatorial optimization problem has to enter the network in an appropriate way. For MIS, that is the self-attention. A message-passing GNN encoder in front of the recursion leaves a one-shot gap up to twice that of the edge-biased attention from Eq. (2), which needs no encoder at all. For TSP, it is the input

Table 2: TSP results on uniform instances. We report mean tour length, gap in % against Concorde ($n = 500, 1000$) and LKH-3 ($n = 10000$), and seconds per instance on one RTX 4090. The Search column indicates the search procedure that produces each result, either post-processing after the network or a custom refinement loop (details in Appendix E). The meanings of "–", "†", and the reference solver time follow those of Table 1. Bold marks the lowest gap at each problem size.

Dataset		TSP-500			TSP-1000			TSP-10000		
Method	Search	Len.	Gap %	s/inst	Len.	Gap %	s/inst	Len.	Gap %	s/inst
Concorde / LKH-3 (reference)	–	16.55	0.00	18	23.12	0.00	187	71.77	0.00	1980
General-purpose neural solvers (MIS and TSP)										
DIMES (Qiu et al., 2022)	MCTS	16.84	1.76	67	23.69	2.46	136	74.06	3.19	1219
DIFUSCO (Sun & Yang, 2023)	2-opt	16.65	0.57	2.9	23.45	1.43	12.4	73.89	2.95	266
DIFUSCO	MCTS	16.63	0.46	51	23.39	1.17	104	73.62	2.58	1076
T2T (Li et al., 2023)	2-opt	16.61	0.37	4.4	23.30	0.78	11.0	73.81	2.84	–
UDC (Zheng et al., 2024)	conquer	16.82	1.66	1.8	23.53	1.78	2.7	74.71	4.09	1.9
Fast T2T (Li et al., 2024)	2-opt	16.58	0.21	2.5	23.22	0.42	5.9	72.90	1.57	–
Moco (Dernedde et al., 2025)	2-opt	16.62	0.40	98	–	–	–	–	–	–
DISCO (Zhao et al., 2025)	2-opt	–	–	–	–	–	–	73.81	2.84	–
COExpander (Ma et al., 2025)	2-opt	16.59	0.25	1.0	23.27	0.64	2.2	72.80	1.45	25
CADO (Song et al., 2026)	2-opt	16.65	0.61	–	23.32	0.88	–	73.69	2.68	–
CAM (Feng et al., 2026b)	2-opt	16.91	2.18	3.2	23.70	2.51	12.6	–	–	–
HyCO (Li et al., 2026)	2-opt	16.74	1.17	–	23.64	2.26	–	–	–	–
Learning-free methods										
iSCO (Sun et al., 2023)	sampler	16.64	0.54	–	23.33	0.91	–	74.02	3.14	–
SoftDist (Xia et al., 2024)	MCTS	16.78	1.44	50	23.63	2.20	101	74.03	3.13	1012
Zero heatmap (Pan et al., 2026)	MCTS	16.66	0.66	50	23.39	1.16	100	74.50	3.79	102
GT-Prior (Pan et al., 2026)	MCTS	16.63	0.50	50	23.31	0.85	100	73.31	2.13	101
TSP-specific neural solvers (no MIS results)										
Att-GCN (Fu et al., 2021)	MCTS	16.97	2.54	20	23.86	3.22	40	74.93	4.39	401
BQ-NCO (Drakulić et al., 2023)	beam search	16.64	0.55	3.5	23.44	1.38	8.6	–	–	–
UTSP (Min et al., 2023)	local search	16.68	0.84	20	23.39	1.18	41	–	–	–
LEHD (Luo et al., 2023)	RRC	16.57	0.16	53.5	23.29	0.75	256	–	–	–
GLOP (Ye et al., 2024)	revisers	16.91	1.99	0.8	23.84	3.11	1.6	75.29	4.90	4.7
SIL (Luo et al., 2024)	PRC	–	–	–	23.31	0.81	349	73.32	2.00	1011
DualOpt (Zhou et al., 2025)	LKH-3 + NN	–	–	–	23.31	0.83	2.0	72.62	1.18	13.9
DRHG (Li et al., 2025)	LNS	16.58	0.23	25.8	23.19	0.29	52	72.85	1.33	106
HyperNS (Lu et al., 2026)	LK search	–	–	–	23.20	0.35	–	**72.44**	**0.93**	–
PCI (Basson & Preux, 2026)	2-opt	16.57†	0.17†	–	23.19†	0.31†	–	–	–	–
TRM4CO (general-purpose; noise σ=0.2)										
TRM4CO (K=40, D=32)	2-opt	**16.56**	**0.11**	2.1	**23.18**	**0.27**	4.4	72.81	1.45	364

encoding. Raw coordinates leave a gap $5.6\times$ that of sinusoidal features at the scale of nearest-neighbor distances. However, a k-nearest-neighbor (kNN) edge bias, as on MIS, adds nothing, and a finer sinusoidal embedding hurts. Second, decoding composes with the recursion. On MIS, sequential peeling cuts the gap of the same checkpoint by two thirds, and more width and depth halve what remains. On TSP, the greedy walk strands nodes and recovers only half of the loss even with multi-start, whereas edge insertion cannot strand nodes, so its raw tours have half the gap of the multi-start walk, and one 2-opt pass brings them within 1% of optimal. Third, the prefix tokens provide global memory. Removing them raises the one-shot MIS gap by half at the same schedule, so a few registers that every node can attend to give the recursion a scratchpad that node tokens alone do not.

3.6 Self-relabeling as self-supervision

We can generate and improve the labels throughout training by keeping the best solution (Section 2.3). Table 3 compares this with supervised training on ER-[700-800] and TSP-500, under the same decoding settings as Tables 1 and 2.

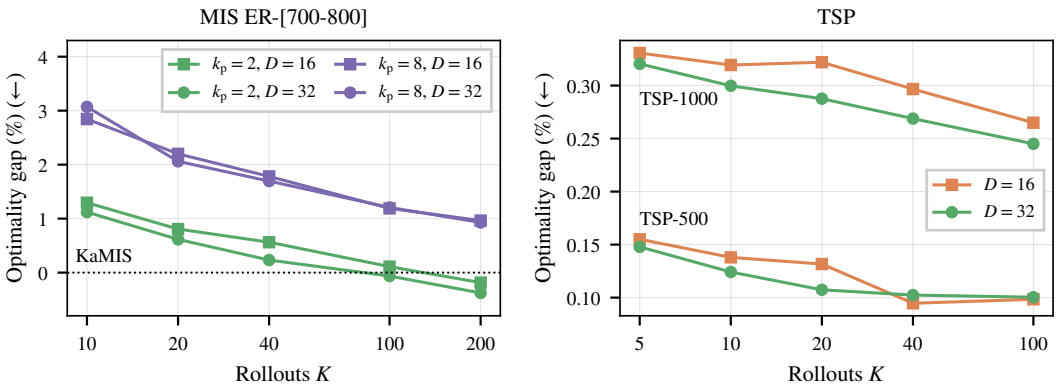


Figure 2: Test-time scaling. Left: MIS on ER-[700-800], optimality gap against the number of rollouts K (log scale) for sequential peeling with $k_p \in \{2, 8\}$ nodes per round at depth $D \in \{16, 32\}$. Right: TSP-500 and TSP-1000, gap after 2-opt against K for $D \in \{16, 32\}$.

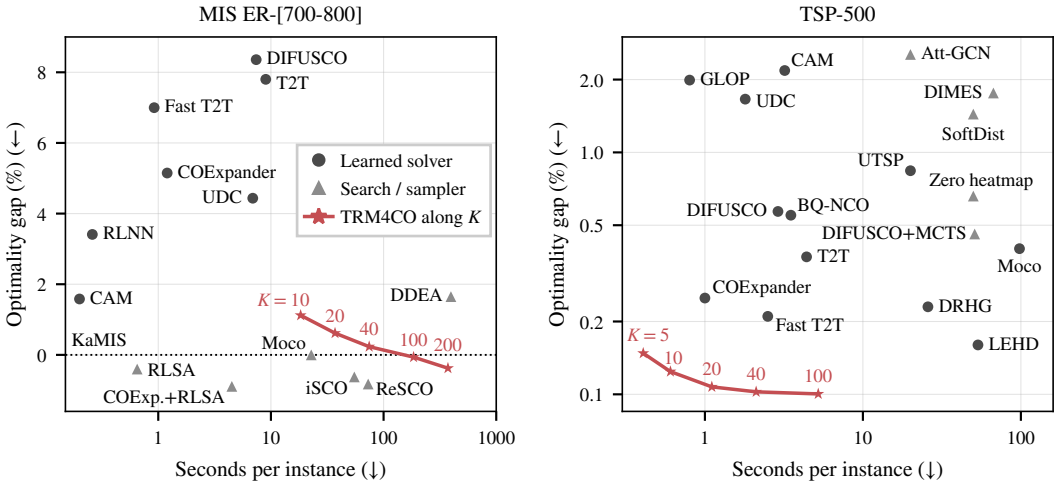


Figure 3: Quality against wall-clock time. We plot optimality gap against seconds per instance on one RTX 4090 for ER-[700-800] (left) and TSP-500 (right). The MIS gap is measured against the KaMIS reference (dotted line) of Table 1, so negative values mean that the reference is beaten. Circles are learned solvers, triangles are pipelines dominated by search or sampling (MCTS, samplers, evolution), and stars are TRM4CO for increasing K ($k_p = 2$ and $D = 32$ on ER-[700-800], $D = 32$ on TSP-500).

Labels beyond the solver. Continuing the supervised MIS checkpoint with the labels replaced by its own better solutions cuts the test gap by more than half and gives our best checkpoint, the one behind Table 1. This indicates that self-relabeling can further improve a trained model, since it then trains on larger sets than those found by the solver for the initial training labels. The relabeling passes raise about 16% of the training instances above their KaMIS labels, by one node on average.

No solver at all. Starting from random initialization with no labels available, the model first seeds the initial labels with its own solution guesses and then hill-climbs on them. Once there is a signal, the model exploits it in training and keeps self-relabeling. On MIS its labels end up within one node of the KaMIS labels on average and match or beat them on 42% of the training graphs. Its test gap comes within one third of a point of the supervised continuation and is below that of the supervised checkpoint itself. On TSP-500 the label-free run closes most of the distance to the LKH-3 tours, from about 10% above them to less than 2%, but its test gap is still 20× that of the supervised model. Figure 5 in Appendix G traces the evolution of the training labels.

Table 3: Self-relabeling. Test gap in % for MIS on ER-[700-800] against KaMIS, decoded with sequential peeling at the setting of Table 1 ($k_{\mathrm{p}} = 2$, $\tau = 0.5$, $K = 10$, $D = 32$), and for TSP-500 against Concorde at the setting of Table 2 ($K = 40$, $D = 32$, 2-opt). The supervised rows train on solver labels, the continuation row trains on top of the supervised checkpoint, and the from-scratch rows start from random initialization without any solver labels.

Run	Labels	Gap %
MIS ER-[700-800]		
Supervised	KaMIS	2.53
Self-relabel continuation	KaMIS, then own	**1.11**
Self-relabel from scratch	Own only	1.43
TSP-500		
Supervised	LKH-3	**0.11**
Self-relabel from scratch	Own only	2.26

4 Related work

Constructive policies. Autoregressive policies build a solution one decision at a time (Vinyals et al., 2015; Kool et al., 2019; Kwon et al., 2020). LEHD (Luo et al., 2023) and BQ-NCO (Drakulić et al., 2023) train on small instances and then generalize. GLOP (Ye et al., 2024), UDC (Zheng et al., 2024), DualOpt (Zhou et al., 2025), and H-TSP (Pan et al., 2023) reach larger instances by decomposition. HyperNS (Lu et al., 2026) instead guides a Lin–Kernighan search with a heatmap, and GOAL (Drakulić et al., 2025) injects edges into attention as we do.

Heatmap solvers and search. Heatmap solvers score all variables in one pass and leave construction to a search. Att-GCN (Fu et al., 2021) pairs a learned heatmap with MCTS for large TSP instances, and DIMES (Qiu et al., 2022) makes the heatmap a differentiable policy. DIFUSCO (Sun & Yang, 2023) models the heatmap with a diffusion model, and T2T (Li et al., 2023), Fast T2T (Li et al., 2024), DISCO (Zhao et al., 2025), CADO (Song et al., 2026), DDEA (Salvà Soler & Raidl, 2026), and PCI (Basson & Preux, 2026) refine its training, sampling, or search. TRM4CO instead refines its prediction by recursion inside the network. Xia et al. (2024) and Pan et al. (2026) show that the search dominates the heatmap in these pipelines, and Böther et al. (2022) and Angelini & Ricci-Tersenghi (2023) reach a similar verdict for MIS. FrontierCO (Feng et al., 2026a) and the survey of Ba et al. (2026) find neural solvers behind tuned heuristics at scale.

Unsupervised and self-improving solvers. UTSP (Min et al., 2023), DiffUCO (Sanokowski et al., 2024), SDDS (Sanokowski et al., 2025), and CAM (Feng et al., 2026b) train without solver labels by optimizing relaxed objectives. Moco (Dernedde et al., 2025) learns a heatmap optimizer that spends its compute at test time. SIL (Luo et al., 2024) improves a constructive policy on its own solutions after local search. Samplers such as iSCO (Sun et al., 2023) and ReSCO (Li & Zhang, 2025) need no learning at all. Our self-relabeling is closest to SIL, with the recursive model's own decoded solutions as labels and no relaxed objective.

Recursive reasoning. HRM (Wang et al., 2025) and TRM (Jolicoeur-Martineau, 2025; Sghaier et al., 2026) solve puzzles by latent recursion with adaptive halting. Looped Transformers reuse one block of layers to gain depth (Wang et al., 2026; Suleymanzade et al., 2026). To the best of our knowledge, we are the first to apply latent recursion to combinatorial optimization at this scale.

5 Conclusion

We have presented TRM4CO, a small recursive network with problem-specific encoding and decoding that is competitive with or better than search-heavy pipelines on MIS and TSP. Its key ingredient is test-time scaling in depth and in width on the same weights, which efficiently turns compute into solution quality. Self-relabeling improves the model further, on top of solver labels or without any. TRM4CO attacks the remaining gap between neural solvers and learning-free search from the perspective of recursive reasoning. Our backbone is general, but the encoder and decoder remain problem-specific, and reducing that dependence is the natural next step.

AI USE STATEMENT

In this work, we used generative AI tools to implement methods, to provide feedback on experiments, and to interpret results. We have not used generative AI tools to generate synthetic datasets, to develop theoretical models or conceptual frameworks, to formulate mathematical claims, to prove or write proofs, or to propose or refine hypotheses, and translation, dataset cleaning, and qualitative data analysis are not applicable to this work. Additionally, we used generative AI tools to create and edit software code, to create scientific figures, to draft and edit parts of the paper for readability, to identify, summarize, and format relevant literature and references, and to suggest experimental parameters. We have reviewed all AI-assisted work. All AI-generated code was verified and tested by the authors, every reported number was produced by our own runs or taken from the cited papers as described in Appendix E and checked by the authors, and all AI-drafted text was revised by the authors. We take responsibility for the final content of this work, including text, claims, or artifacts produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

The supplementary material contains an anonymized version of our code, with the training and evaluation scripts, the configuration of every reported run, our best checkpoints for ER-[700-800], TSP-500, and TSP-1000, and a sample of test instances with their reference solutions. Its README maps each of our rows in Tables 1 and 2 to a command. The full datasets are those of Qiu et al. (2022) and Fu et al. (2021) and are too large to attach. Appendix B lists the hyperparameters, Appendix C the datasets and reference solutions, and Appendix D the runtime protocol. The full code, checkpoints, and relabeled datasets will be released upon acceptance.

REFERENCES

Sungsoo Ahn, Younggyo Seo, and Jinwoo Shin. Learning what to defer for maximum independent sets. In *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pp. 134–144, 2020.

Maria Chiara Angelini and Federico Ricci-Tersenghi. Modern graph neural networks do worse than classical greedy algorithms in solving combinatorial optimization problems like maximum independent set. *Nature Machine Intelligence*, 5(1):29–31, 2023.

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

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

Mickaël Basson and Philippe Preux. Leveraging structural constraints for diffusion-based neural TSP solvers. In *Learning and Intelligent Optimization (LION)*, 2026.

Maximilian Böther, Otto Kißig, Martin Taraz, Sarel Cohen, Karen Seidel, and Tobias Friedrich. What's wrong with deep learning in tree search for combinatorial optimization. In *International Conference on Learning Representations*, 2022.

Tim Dernedde, Daniela Thyssens, Sören Dittrich, Maximilian Stubbemann, and Lars Schmidt-Thieme. Moco: A learnable meta optimizer for combinatorial optimization. In *Advances in Knowledge Discovery and Data Mining (PAKDD)*, volume 15872 of *Lecture Notes in Computer Science*, pp. 236–248. Springer, 2025.

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

Darko Drakulić, Sofia Michel, and Jean-Marc Andreoli. GOAL: A generalist combinatorial optimization agent learner. In *International Conference on Learning Representations*, 2025.

Vijay Prakash Dwivedi, Anh Tuan Luu, Thomas Laurent, Yoshua Bengio, and Xavier Bresson. Graph neural networks with learnable structural and positional representations. In *International Conference on Learning Representations*, 2022.

Paul Erdős and Alfréd Rényi. On the evolution of random graphs. *Publications of the Mathematical Institute of the Hungarian Academy of Sciences*, 5:17–61, 1960.

Shengyu Feng and Yiming Yang. Regularized Langevin dynamics for combinatorial optimization. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 16977–16992. PMLR, 2025.

Shengyu Feng, Weiwei Sun, Shanda Li, Ameet Talwalkar, and Yiming Yang. FrontierCO: Real-world and large-scale evaluation of machine learning solvers for combinatorial optimization. In *International Conference on Learning Representations*, 2026a.

Shengyu Feng, Tarun Suresh, and Yiming Yang. Unsupervised diffusion solver for combinatorial optimization via combinatorial adjoint matching. In *Proceedings of the 43rd International Conference on Machine Learning*, volume 306 of *Proceedings of Machine Learning Research*, 2026b.

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

Alex Graves. Adaptive computation time for recurrent neural networks. *arXiv preprint arXiv:1603.08983*, 2016.

Gurobi Optimization, LLC. Gurobi Optimizer Reference Manual. `https://www.gurobi.com`, 2026.

Keld Helsgaun. An extension of the Lin-Kernighan-Helsgaun TSP solver for constrained traveling salesman and vehicle routing problems. Technical report, Roskilde University, 2017.

Demian Hespe, Christian Schulz, and Darren Strash. Scalable kernelization for maximum independent sets. *ACM Journal of Experimental Algorithmics*, 24(1):1.16:1–1.16:22, 2019.

Holger H. Hoos and Thomas Stützle. SATLIB: An online resource for research on SAT. In Ian P. Gent, Hans van Maaren, and Toby Walsh (eds.), *SAT 2000: Highlights of Satisfiability Research in the Year 2000*, pp. 283–292. IOS Press, 2000.

Alexia Jolicoeur-Martineau. Less is more: Recursive reasoning with tiny networks. *arXiv preprint arXiv:2510.04871*, 2025.

Keller Jordan, Yuchen Jin, Vlado Boza, Jiacheng You, Franz Cesista, Laker Newhouse, and Jeremy Bernstein. Muon: An optimizer for hidden layers in neural networks. `https://kellerjordan.github.io/posts/muon/`, 2024.

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.

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

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

Sebastian Lamm, Peter Sanders, Christian Schulz, Darren Strash, and Renato F. Werneck. Finding near-optimal independent sets at scale. *Journal of Heuristics*, 23(4):207–229, 2017.

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

Muheng Li and Ruqi Zhang. Reheated gradient-based discrete sampling for combinatorial optimization. *Transactions on Machine Learning Research*, 2025.

Yang Li, Jinpei Guo, Runzhong Wang, and Junchi Yan. T2T: From distribution learning in training to gradient search in testing for combinatorial optimization. In *Advances in Neural Information Processing Systems*, volume 36, pp. 50020–50040, 2023.

Yang Li, Jinpei Guo, Runzhong Wang, Hongyuan Zha, and Junchi Yan. Fast T2T: Optimization consistency speeds up diffusion-based training-to-testing solving for combinatorial optimization. In *Advances in Neural Information Processing Systems*, volume 37, pp. 30179–30206, 2024.

Yuheng Li, Di Yang, Haipeng Chen, and Yanhai Xiong. HyCO: A hybrid neural solver for combinatorial optimization. *arXiv preprint arXiv:2609.07990*, 2026.

Tongkai Lu, Shuai Ma, and Chongyang Tao. Empowering targeted neighborhood search via hyper tour for large-scale TSP. *Frontiers of Computer Science*, 2026.

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

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

Jiale Ma, Wenzheng Pan, Yang Li, and Junchi Yan. COExpander: Adaptive solution expansion for combinatorial optimization. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 42130–42164. PMLR, 2025.

Ben Mildenhall, Pratul P. Srinivasan, Matthew Tancik, Jonathan T. Barron, Ravi Ramamoorthi, and Ren Ng. NeRF: Representing scenes as neural radiance fields for view synthesis. In *Computer Vision – ECCV 2020*, Lecture Notes in Computer Science, pp. 405–421. Springer, 2020.

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

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

Xuanhao Pan, Chenguang Wang, Chaolong Ying, Ye Xue, and Tianshu Yu. Beyond the heatmap: A rigorous evaluation of component impact in MCTS-based TSP solvers. In *International Conference on Learning Representations*, 2026.

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

Joan Salvà Soler and Günther R. Raidl. A denoising diffusion-based evolutionary algorithm framework: Application to the maximum independent set problem. In *Machine Learning, Optimization, and Data Science (LOD)*, Lecture Notes in Computer Science, pp. 1–16. Springer, 2026.

Sebastian Sanokowski, Sepp Hochreiter, and Sebastian Lehner. A diffusion model framework for unsupervised neural combinatorial optimization. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 43346–43367, 2024.

Sebastian Sanokowski, Wilhelm Berghammer, Haoyu Wang, Martin Ennemoser, Sepp Hochreiter, and Sebastian Lehner. Scalable discrete diffusion samplers: Combinatorial optimization and statistical physics. In *International Conference on Learning Representations*, 2025.

Amin Sghaier, Ali Parviz, and Alexia Jolicoeur-Martineau. Probabilistic tiny recursive model. *arXiv preprint arXiv:2605.19943*, 2026.

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

Ayhan Suleymanzade, Chanhyuk Lee, Floor Eijkelboom, Nicholas M. Boffi, İsmail İlkan Ceylan, and Jinwoo Kim. Thinking with looped flows. *arXiv preprint arXiv:2609.11801*, 2026.

Haoran Sun, Katayoon Goshvadi, Azade Nova, Dale Schuurmans, and Hanjun Dai. Revisiting sampling for combinatorial optimization. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pp. 32859–32874, 2023.

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

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

Guan Wang, Jin Li, Yuhao Sun, Xing Chen, Changling Liu, Yue Wu, Meng Lu, Sen Song, and Yasin Abbasi Yadkori. Hierarchical reasoning model. *arXiv preprint arXiv:2506.21734*, 2025.

Shaowen Wang, Ge Zhang, Kairong Luo, Yuhao Wu, Shaofan Liu, Jiaheng Liu, Wenhao Huang, Shen Yan, and Jian Li. SMELT: Scaling laws for compute-matched MoE looped transformers. *arXiv preprint arXiv:2609.01343*, 2026.

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 *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 54178–54190, 2024.

Haoran Ye, Jiarui Wang, Helan Liang, Zhiguang Cao, Yong Li, and Fanzhang Li. GLOP: Learning global partition and local construction for solving large-scale routing problems in real-time. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pp. 20284–20292, 2024.

Chengxuan Ying, Tianle Cai, Shengjie Luo, Shuxin Zheng, Guolin Ke, Di He, Yanming Shen, and Tie-Yan Liu. Do transformers really perform badly for graph representation? In *Advances in Neural Information Processing Systems*, volume 34, pp. 28877–28888, 2021.

Dinghuai Zhang, Hanjun Dai, Esmeralda S. Whitammer, Aaron Courville, Yoshua Bengio, and Ling Pan. Let the flows tell: Solving graph combinatorial problems with GFlowNets. In *Advances in Neural Information Processing Systems*, volume 36, pp. 11952–11969, 2023.

Hang Zhao, Kexiong Yu, Yuhang Huang, Renjiao Yi, Chenyang Zhu, and Kai Xu. DISCO: Efficient diffusion solver for large-scale combinatorial optimization problems. *Graphical Models*, 141: 101284, 2025.

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

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*, volume 39, pp. 27178–27186, 2025.

A ALGORITHM AND PROBLEM-SPECIFIC HOOKS

Algorithm 1 lists the procedures of Section 2.3 that MIS and TSP share, and Table 4 the problem-specific hooks it calls. Figure 4 unrolls one recursion step.

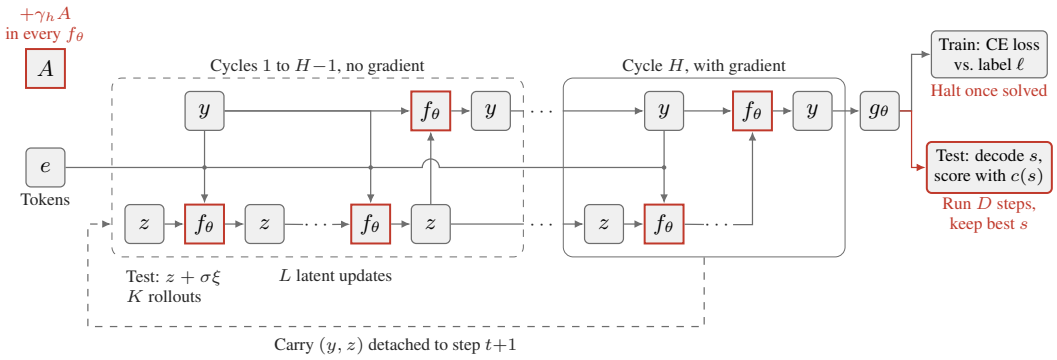


Figure 4: One recursion step of TRM4CO, unrolled. Gray is the TRM schedule of Eq. (1), with L latent updates and one answer update per cycle, H cycles per step, gradient only through the last cycle, and (y, z) carried detached to the next step. Test-time noise on z (Sghaier et al., 2026) gives K independent rollouts. Red is what we change. For MIS, the adjacency A biases every attention call (Eq. 2). At test time, every step (TSP) or every peeling round (MIS) is decoded into a feasible solution and scored with the objective. In training, the solved check of Table 4 halts an instance in place of the learned halting signal.

Table 4: Problem-specific hooks of Algorithm 1. Everything not listed here (the core, the recursion schedule, deep supervision, and the self-relabeling loop) is shared.

Hook	MIS	TSP
Instance x	Adjacency $A \in \{0,1\}^{n\times n}$	Coordinates $X \in [0,1]^{n\times 2}$
Tokens $\mathrm{Emb}(x)$	Linear[deg, RWPE, $u \sim \mathcal{U}(0,1)$]	Linear[X, $\sin$ / $\cos(2^l \pi X)]_{l<L_{\mathrm{pe}}}$
Attention bias	Adjacency A	None (standard self-attention)
Head $g_\theta(y)$	Node logits $[n, 2]$	Bilinear successor logits $[n, n]$
Decoder Dec	Greedy independent set in score order	Greedy edge insertion by $\bar{g}_{ij}/d_{ij}$
Wrapper $\mathcal{W}$	Sequential peeling around the recursion	2-opt on the tour returned by SOLVE
Objective $c(s)$	$\lvert S\rvert$ (max)	Tour length (min)
$\mathrm{Solved}(g_\theta(y), \ell, x)$	Greedy $\lvert S\rvert \geq \lvert\ell\rvert$	Successors match the canonical ℓ
Labels ℓ	KaMIS 10 s (ER), exact (SATLIB)	LKH-3 (1000 trials)
Augmentation	Random peel of label nodes	Dihedral-8 of the unit square

B HYPERPARAMETERS

Training. The loss is a cross-entropy per node (MIS) or per successor (TSP) against the labels of Table 4, with peel augmentation for MIS and the eight symmetries of the square for TSP applied on the fly (see below). Each model trains on two H100 GPUs. Several reported checkpoints continue earlier runs, and the epoch counts in Table 5 are summed over all stages. The ER supervised checkpoint runs 10,000 epochs with the edge-biased architecture, 5000 at $D = 64$ and 5000 at $D = 32$, warm-started from a run with a message-passing encoder. Its self-relabeling continuation adds 3000 epochs. The SATLIB and TSP-10000 checkpoints continue a first run of 1000 epochs for another 1000. The label-free ER chain of Table 3 runs 5000 epochs with one-shot relabeling and three stages of 3000 epochs with peeling, and the label-free TSP-500 chain runs stages of 500, 500, and 1000 epochs.

Implementation details. The halting head of TRM is retained, but we block its gradient. Its loss is still computed with weight 0.5 as in TRM for logging, and the head is used neither for halting nor

Algorithm 1 TRM4CO: one recursion step, inference with depth D and width K, and training. Problem-specific hooks are listed in Table 4, and the rest is shared between MIS and TSP.

Require: core f_θ, head g_θ, prefix tokens p, initial states y_0, z_0
Require: hooks Emb, Dec, c, Solved, and wrapper $\mathcal{W}$ of Table 4

procedure STEP(y, z, e, A) ▷ Eq. (1), with Eq. (2) inside f_θ
  for $i = 1, \ldots, H$ **do** ▷ cycles $1 \ldots H-1$ without gradient
    for $l = 1, \ldots, L$ **do**
      $z \leftarrow f_\theta(z + y + e;\ A)$
    end for
    $y \leftarrow f_\theta(z + y;\ A)$
  end for
  return y, z
end procedure

procedure SOLVE(x; D, K, σ) ▷ inference; x may be a batch of instances
  $e \leftarrow [p;\ \mathrm{Emb}(x)]$
  $A \leftarrow$ adjacency of x (MIS) or 0 (TSP)
  $(y^{(j)}, z^{(j)}) \leftarrow (y_0, z_0)$ for $j = 1, \ldots, K$
  $s^\star \leftarrow$ none
  for $t = 1, \ldots, D$ **do** ▷ depth
    for $j = 1, \ldots, K$ **in one batched forward pass do** ▷ width
      $z^{(j)} \leftarrow z^{(j)} + \sigma\xi$ with $\xi \sim \mathcal{N}(0, I)$
      $(y^{(j)}, z^{(j)}) \leftarrow$ STEP($y^{(j)}, z^{(j)}, e, A$)
      $s \leftarrow \mathrm{Dec}(g_\theta(y^{(j)}), x)$
      if $s^\star$ is none or $c(s)$ improves on $c(s^\star)$ **then**
        $s^\star \leftarrow s$
      end if
    end for
  end for
  return $s^\star$
end procedure

procedure TRAIN($\{(x_i, \ell_i)\}$; D, T) ▷ deep supervision with problem-specific halting
  fill B batch slots with augmented instances, each with e, A, (y_0, z_0), and $t = 0$
  loop
    $(y, z) \leftarrow$ STEP(y, z, e, A) on all slots
    $t \leftarrow t + 1$
    $\mathcal{L} \leftarrow \mathrm{CE}(g_\theta(y), \ell)$
    update θ with $\nabla_\theta \mathcal{L}$
    detach (y, z)
    for each slot with $\mathrm{Solved}(g_\theta(y), \ell, x)$ or $t = D$ **do**
      refill the slot with a new augmented instance
    end for
    if T epochs have passed since the last relabel pass **then**
      $\ell_i \leftarrow$ the better of ℓ_i and $\mathcal{W}(x_i)$ for all i ▷ self-relabel
    end if
  end loop
end procedure

for selection. As in TRM, a random minimum step count is imposed on a small fraction of training instances so that later steps keep receiving a gradient. The loss is the stablemax cross-entropy of Jolicoeur-Martineau (2025). Body weights are trained with Muon (Jordan et al., 2024), heads and biases with AdamW, and prefix tokens with sign-SGD. An exponential moving average of the weights is maintained. Training uses $D = 16$ for TSP and 32 for MIS, $P = 16$ prefix tokens, $H = 3$ cycles of $L = 6$ latent updates, and a 12-step random-walk positional encoding for MIS (Table 5). The TSP solved check compares predicted successors with the canonical (counterclockwise) label because decoding a tour would cost n sequential steps per training step. The 2-opt repair may be

restricted to the m longest edges of the tour. The main-table numbers use a full scan. During self-relabeling, the TSP tours of a relabel pass are repaired with 2-opt and canonicalized before the comparison with the stored label.

Table 5: Hyperparameters of the reported models. Epochs are passes over the training set, summed over all stages of a continued run. The self-relabeling continuation of Table 3 adds 3000 epochs with a 2000-step warm-up. Test-time K, D, σ, τ, k_{p}, and m are given with each table.

Parameter	Value
Hidden size, heads, blocks	512, 8, 2
MLP expansion	4
Recursion H, L	3, 6
Prefix tokens P	16
Structure, MIS	Edge-biased attention, 12-step random-walk positional encoding
Structure, TSP	Self-attention, $L_{\mathrm{pe}} = 6$ coordinate encoding
Successor head dimension d_s	128 (TSP)
Training depth D	32 (MIS; 64 in the first ER stage), 16 (TSP)
Global batch size	768 (ER, TSP-500), 384 (SATLIB), 512 (TSP-1000), 48 (TSP-10000)
Epochs, MIS	10,000 (ER), 2000 (SATLIB)
Epochs, TSP	500 (TSP-500), 1000 (TSP-1000), 2000 (TSP-10000)
Learning rate	AdamW 10^{-4}, Muon $2 \cdot 10^{-3}$, 5000 warm-up steps, cosine to 1%
Weight decay, EMA rate	0.1, 0.999
Augmentation	Peel, prob. 0.9, depth ≤ 9 (ER), ≤ 32 (SATLIB); dihedral-8 (TSP)
Self-relabel pass, ER continuation	Every 200 epochs, $K = 4$, $D = 16$, $k_{\mathrm{p}} = 2$

Augmentations. Two augmentations run on the fly. For MIS, peel augmentation removes a random number of label nodes one at a time, each with its closed neighborhood, so that training also sees the residual graphs that sequential peeling visits. For TSP, one of the eight symmetries of the square is applied with the tour orientation kept canonical (counterclockwise).

C Datasets and references

MIS. The ER-[700-800] and SATLIB graphs are those distributed with DIMES (Qiu et al., 2022). We label the ER training graphs with KaMIS (Lamm et al., 2017) at a 10-second budget per graph, which gives a mean set size of 44.52 on the training graphs and 44.63 on the test graphs. DIFUSCO and the papers that follow it compare against a longer KaMIS run on the test graphs with a mean size of 44.87 (24 seconds per graph), which Table 1 and Figures 2 and 3 use as the reference. The ablation table (Table 7) reports gaps against our own test labels, 44.63, so its gaps are about half a point lower for the same set size. SATLIB labels are exact, computed with Gurobi (Gurobi Optimization, LLC, 2026). Their mean of 425.96 on the test graphs coincides with the KaMIS reference of those papers.

TSP. Training instances are drawn uniformly from the unit square and labeled with LKH-3 (Helsgaun, 2017) using 1000 trials and the best of 10 runs per instance. The test sets are the 128, 128, and 16 instances of Fu et al. (2021). The reference tours that ship with their public copies are not optimal. The tours attributed to Concorde (Applegate et al., 2006) are 0.23% and 0.47% above the optimum at $n = 500$ and 1000, and the $n = 10000$ file carries placeholder tours. We therefore recompute the references, with Concorde at $n = 500$ and 1000 (certified optimal, mean lengths 16.5458 and 23.1181) and with LKH-3 at $n = 10000$ (10,000 trials, best of 10 runs, mean length 71.7708, the LKH-3 value quoted in the literature). All gaps in Table 2 are against these references, and published lengths are re-expressed accordingly (Appendix E).

D Runtime protocol

All runtimes in Tables 1 and 2 are our own measurements on one RTX 4090 with the released code and checkpoints of each method, in the inference setting whose quality number the table quotes. For each compared method, we time the complete pipeline from the instance to the final solution, including heatmap generation, decoding, and any post-processing such as 2-opt, rewriting, or search,

Table 6: Learned parameters of each network. Ours and Moco are counted from checkpoints; the others are derived from stated hyperparameters or official code, since no paper states its count.

Method	MIS	TSP
Moco	0.36M	0.85M
T2T / Fast T2T	1.34M	5.33M
DIFUSCO / DISCO	–	5.33M
CAM	0.18M	5.20M
UDC	4.35M	1.56M
DRHG	–	2.65M
HyperNS (heatmap network only)	–	∼11.1M
TRM4CO	6.83M	6.96M

on a subset of our test instances (8, 4, and 2 instances at $n = 500$, 1000, and 10000 for TSP, and 8 graphs for MIS), and report the mean per instance after a warm-up instance or batch. Batched solvers (DRHG, SIL, LEHD, BQ-NCO, iSCO, ReSCO) are timed on batches of 2 to 8 instances, and the batch time is divided by the batch size, which overstates their per-instance cost relative to the batch sizes in their papers. MCTS pipelines run the search of Fu et al. (2021) at the time limit of the respective codebase. The limit is 0.1 seconds per city for DIFUSCO, DIMES, SoftDist, and the controls of Pan et al. (2026), and 0.04 seconds per city for Att-GCN and UTSP, so their runtime is set by that budget plus the heatmap time. Methods without public code (DISCO, CADO, HyCO, PCI, HyperNS) or without released weights for our benchmarks (DIMES on MIS, GFlowNets, LwD, DiffUCO, SDDS) have no runtime, as do settings that exceed the 24 GB memory of the card (T2T and Fast T2T sampling at $n = 10000$). Our own TSP-10000 row exceeds the memory of the card as well, since its 40 rollouts do not fit in 24 GB at once. We run it on an RTX 5090 (340 seconds per instance) and scale the time by the 4090-to-5090 runtime ratio of TSP-1000 at the same K and D (4.43 to 4.13 seconds), which gives the 364 seconds in Table 2 with the unrounded timings. Our own runtimes use the full test sets in batches of 320 rollouts (for example, 32 instances at $K = 10$ and 8 at $K = 40$) and exclude the first batch, which includes compilation. The MIS $K = 200$ point uses 6 graphs in batches of 2. Table 6 lists the parameter counts of the inference networks.

E Notes on the main tables

Sources of the quality numbers. Every quality number of a compared method is the one reported in its own paper, in its main sampling setting. Where a paper does not cover one of our benchmarks but a compared paper evaluates the method on it, we use that number. DiffUCO and SDDS on ER-[700-800] are the re-implementations of Feng et al. (2026b), LwD's ER number is the rerun of Qiu et al. (2022), and T2T and Fast T2T at $n = 10000$ for TSP are evaluated by Zhao et al. (2025) and Ma et al. (2025). Zero heatmap and GT-Prior are the heatmap-free controls of Pan et al. (2026). For TSP, where a paper prints only the length or only the gap, the other is derived against our reference, and where it uses another reference, the gap is recomputed against ours.

Own instance sets. The dagger marks methods evaluated on their own generated instances of the same distribution, with the gap against their own reference run. Where a method's code and checkpoints are released, we re-evaluate it on our test sets in the setting of its paper row and report that number without a dagger. These are COExpander with and without RLSA on both MIS benchmarks, and UDC, BQ-NCO, DRHG, and LEHD on TSP. Their full-set results agree with their own-instance numbers to within 0.1 points, except for COExpander with RLSA (45.27 against 44.98 on ER-[700-800] and 425.45 against 425.32 on SATLIB) and DRHG at $n = 500$ (0.23% against the reported 0.11%, with either released checkpoint). The remaining daggers are HyCO and GFlowNets on both MIS benchmarks, LwD on SATLIB, and PCI at $n = 500$ and 1000, whose code or checkpoints are not released. Lengths and sizes on different instances are not directly comparable, so the gap is the number to compare.

Type and Search columns. In Table 1, S + RL combines supervised and reinforcement learning, as supervised pretraining followed by RL fine-tuning for CADO and as an RL constructor whose solution prefix is completed by a supervised diffusion model for HyCO. S + self is supervised train-

ing followed by self-relabeling (ours), and the suffixes "+ search", "+ sampler", and "+ evolution" name the objective-guided gradient search of T2T and Fast T2T, the RLSA sampler applied after COExpander, and the evolutionary loop of DDEA. U + search is Moco's meta-optimizer loop. In Table 2, MCTS denotes the search of Fu et al. (2021) at the time limit of the respective codebase (Appendix D), and RRC, PRC, revisers, and conquer are the iterative neural reconstruction loops of LEHD, SIL, GLOP, and UDC.

Reference solvers. KaMIS, Concorde, and LKH-3 are listed with the CPU times reported by Qiu et al. (2022) and Fu et al. (2021), divided by the number of instances, and are not comparable with the GPU times of the other rows.

F Full ablations

Table 7 lists every variant behind the findings of Section 3.5.

G Self-relabeling details

Figure 5 traces the training labels of the runs of Table 3 over the epochs of self-relabeling.

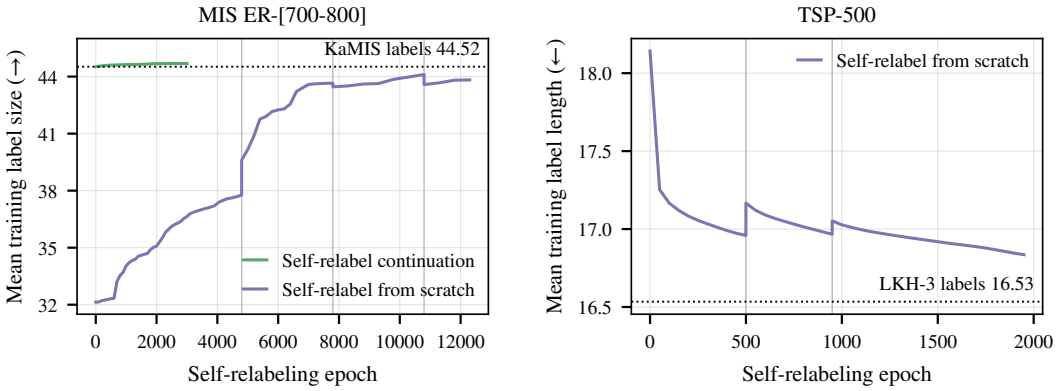


Figure 5: Training labels under self-relabeling, for the runs of Table 3. Mean size (MIS, left) or length (TSP-500, right) of the training labels against the epochs of self-relabeling, for the continuation of the supervised MIS checkpoint and for the label-free chains from random initialization. Vertical lines mark the warm-started stages of a chain. Each stage restarts its labels from the model's own first pass, which produces the steps in the curves. The dotted line is the mean of the solver labels.

A decoder mismatch pitfall. Relabeling with one-shot decoding produces the best one-shot model among all our checkpoints, but it inverts the benefit of sequential peeling. At the same training stage, the supervised model still gains 3 to 5 points from peeling, while the relabeled model loses 1 to 3. Our conjecture is that training toward the model's own argmax sets makes scores mean "in my preferred set" rather than "safe in any near-optimal set", so top-score commitments strand later rounds. Relabeling and selecting checkpoints with the deployment decoder removes the inversion and gives our best MIS checkpoint, the one behind the rows of Table 1.

Table 7: Ablations, one factor at a time. MIS on ER-[700-800] (gap in % against our KaMIS labels, 44.63), TSP-500 (gap in % against Concorde), and TSP-10000 (gap in % against LKH-3). Each block varies one factor of the default configuration, with K, D, and σ as noted. The MIS decoding block uses the supervised checkpoint of Table 3 with $\sigma = 0.5$ and $\tau = 0$, and the rollout and Gumbel rows an earlier checkpoint with $\sigma = 0.7$, so their gaps are not directly comparable with Table 1.

Factor	Variant	Gap %
MIS structure injection (one-shot, K=1)		
	3-layer GNN encoder + self-attention	20.2
	6-layer GNN encoder + self-attention	13.5
	No encoder + edge-biased attention (default)	10.0
MIS prefix tokens (same schedule, best checkpoint of the last training window)		
	$P = 0$	15.0
	$P = 16$ (default)	10.0
MIS decoding (same checkpoint, K=10, D=32)		
	One-shot greedy	9.06
	Sequential peel, k_{p}=2	2.67
	Sequential peel, k_{p}=2, K=40, D=64	1.14
MIS Gumbel noise on peeling (k_{p}=2, σ=0.7, K=4)		
	Sequential peel, τ=0	6.61
	Sequential peel, τ=0.1	6.42
	Sequential peel, τ=0.3	5.86
	Sequential peel, τ=1.0	6.96
MIS rollouts (k_{p}=2, σ=0.7)		
	K=4	7.0
	K=100	1.97
TSP-500 input encoding (same schedule, best in-run checkpoint)		
	Raw coordinates, self-attention	15.89
	Raw coordinates, kNN edge-biased attention	17.27
	L_{pe}=6, self-attention (default)	2.84
	L_{pe}=6, kNN edge-biased attention	3.20
	L_{pe}=12, self-attention	6.72
TSP-500 decoding (same checkpoint, K=10, D=16)		
	Walk, one-shot (K=1, last step)	8.44
	Walk, argmax	2.76
	Walk, multi-start	1.36
	Walk, multi-start + 2-opt (m=16)	0.247
	Walk, multi-start + τ=0.3 + full 2-opt	0.169
	Insertion, raw	0.611
	Insertion + full 2-opt (default)	0.151
TSP-10000 2-opt budget m (insertion, K=4, D=16)		
	m=16	2.73
	m=64	2.19
	m=256	1.91
	m=1024	1.77
	Full scan	1.69