Efficiently Solving the TSP with Non-Autoregressive Self Improvement Learning

Debarpan Debnath
Machine Learning Lab
International Institute of Information Technology
Hyderabad, India
debarpan.debnath@research.iiit.ac.in

Praveen Paruchuri
Machine Learning Lab
International Institute of Information Technology
Hyderabad, India
praveen.p@iiit.ac.in

***Abstract*—Neural approaches that rely on autoregressive (AR) solution construction for solving the Traveling Salesman Problem (TSP)—a cornerstone challenge in combinatorial optimization—can be computationally expensive during both training and inference, since each step of the solution construction requires a forward pass through the neural network. Although non-autoregressive (NAR) methods offer the promise of efficient solution generation in a single network pass, prior NAR approaches have typically depended on Monte Carlo Tree Search (MCTS) to reach near-optimal performance, which substantially slows down the solution generation and demands significant additional computational resources. To address this, we propose NARSIL, a NAR method that generates TSP solutions extremely fast without sacrificing the solution quality. NARSIL is trained in an unsupervised manner using Self-Improvement Learning (SIL) and does not require access to optimal solutions during training. Experimental results demonstrate that NARSIL achieves a 9–11$\times$ speed-up over the previous fastest baseline while delivering superior solution quality, and a 71–265$\times$ speed-up over high-accuracy baselines while producing comparable solution quality.**

***Index Terms*—travelling salesman problem, neural combinatorial optimization, self improvement learning, unsupervised learning**

I. Introduction

The Travelling Salesman Problem (TSP) is an important problem in combinatorial optimization (CO), with widespread applications across domains such as logistics, robotics, genetics, and circuit design [1]. As an NP-hard problem, TSP poses significant computational challenges. While exact solvers like Concorde [2] can find optimal solutions for instances with tens of thousands of nodes, the computation time escalates dramatically with problem size. Heuristic-based methods such as [3], [4] have delivered impressive results — finding near-optimal solutions for problems with millions of nodes — but they rely heavily on hand-crafted rules and domain specific knowledge [5].

In recent years, machine learning (ML) has gained traction as a promising paradigm for developing data-driven solvers that can address diverse CO problems without extensive manual engineering [6]. The TSP is conventionally formulated as a sequential decision-making process, where a tour is constructed incrementally by deciding the next city to visit at each step. Many recent learning-based methods [7]–[12] adopt this approach by autoregressively constructing the tour guided by a neural network model. However, this strategy can be slow and computationally expensive during both training and inference, as it requires querying the neural model at each step of the solution construction, thereby limiting scalability.

Non-autoregressive (NAR) alternatives have been proposed which mitigate the inefficiencies of the AR approach, where the neural network outputs probability distribution over the edges of each node in a single pass. These probability matrices are commonly known as *heatmaps* in the literature [13]. However, heatmap based approaches have largely relied on Monte Carlo Tree Search (MCTS) [14]–[18] to achieve high-quality solutions. While MCTS improves the solution accuracy, it significantly increases inference latency. Moreover, diffusion based NAR approaches [18], [19] require multiple forward passes through the network, reducing efficiency.

In this work, we propose a transformer-based NAR model that generates probability heatmaps in a single forward pass. The model is trained using self-improvement learning (SIL) combined with geometric data augmentation. TSP solutions are then obtained by greedily decoding from these probability heatmaps and subsequently refined with 2-opt local search. Because the initial greedy solutions are already of high quality — owing to the strong heatmaps produced by the model — only a small number of 2-opt iterations are required. As a result, the additional time and computational cost incurred during the 2-opt phase remain very small.

Our contributions are threefold:

1) We propose a NAR approach which can generate TSP solutions which are either superior or of comparable quality to that of AR methods, while being orders of magnitude more efficient.
2) We demonstrate that this approach is not only highly efficient, but it is also achieved in an unsupervised manner through self-improvement learning, which does not require optimal solutions.
3) Our approach achieves near-optimal performance on benchmark instances, with optimality gap of 0.420% on TSP500 and 0.916% on TSP1000, and speed-ups as high as $4875\times$ against Concorde and up-to $525\times$ against other baselines.

Code is available at https://github.com/debnath-d/NARSIL.

II. Motivation

Consider an autoregressive model capable of producing optimal TSP solutions step by step. For the entire predicted tour to be optimal, each step must be optimal, including the first one. Due to the cyclic nature of TSP, the optimal solution is invariant to the starting point. Consequently, the model must produce the optimal tour irrespective of the starting node. Therefore, the model must be able to predict the optimal first step from any node, as the optimal tour can begin from any node in the cycle. In theory, this implies that the model should be capable of predicting the optimal first step from every node in parallel — without relying on partial tour context. Then the entire tour could be predicted in a single pass, eliminating the need for autoregressive generation. Hence, we argue that the optimal TSP solution depends solely on the spatial distribution of the nodes in the graph and is independent of any partially constructed tour.

While a TSP solution is conventionally defined as a sequence of nodes, [10] point out that this can lead to multiple representations of the same underlying solution. The solution to a TSP-n problem (instance with n nodes) can have $2n$ representations. For example with $n = 5$, $\tau = (v_1, v_2, v_3, v_4, v_5)$ and $\tau' = (v_2, v_3, v_4, v_5, v_1)$ represent the same solution, and there are a total of 5 such representations starting with each node. On similar lines, $\tau'' = (v_5, v_4, v_3, v_2, v_1)$ in the reverse direction also represents the same solution and there would be 5 such reverse representations, corresponding to each of the aforementioned representations. To avoid this problem, we represent the TSP solution as an adjacency list. For the above example, τ can be represented as:

$$
\begin{aligned}
v_1 &: \{v_2, v_5\} \\
v_2 &: \{v_1, v_3\} \\
v_3 &: \{v_2, v_4\} \\
v_4 &: \{v_3, v_5\} \\
v_5 &: \{v_1, v_4\}
\end{aligned}
$$

As the same optimal tour can be produced in clockwise and anti-clockwise directions, the optimal entry and exit edges for every city are interchangeable. Therefore, our equivalent objective is to predict these two optimal edges (entry and exit) for every node directly, without the context of any partially constructed tour. This avoids introducing sequential bias in our neural model.

III. Related Work

A. Autoregressive Methods

A prominent approach in neural TSP solving, the autoregressive (AR) framework constructs solutions sequentially by selecting one city at a time. Early pioneering work by [7] introduced the Pointer Network, which employs recurrent neural networks (RNNs) to decode TSP solutions autoregressively under supervised learning. This framework was subsequently enhanced by [8], through reinforcement learning (RL) to eliminate the dependency on pre-computed optimal solutions, using negative tour length as a reward signal in an actor-critic architecture.

The Transformer architecture [20] was introduced to this domain by [9] through the Attention Model (AM) that leverages self-attention mechanisms for improved performance across multiple combinatorial problems including TSP and CVRP. Building upon AM, [10] introduced Policy Optimization with Multiple Optima (POMO), which exploits the symmetric nature of TSP by training on multiple starting points simultaneously.

Subsequent developments have refined these approaches through various enhancements: multiple decoders [21], matrix encoding [22], heterogeneous attention mechanisms [23], and symmetry-aware training [24]. Advanced variants include heavy decoder architectures [11] that dynamically re-embed node representations at each decoding step, achieving impressive performance on large-scale instances at the cost of computational efficiency.

Despite their successes, autoregressive methods face inherent limitations: they require multiple forward passes through the neural network, suffer from error propagation in long sequences, and exhibit high training variance on large graphs [6].

B. Non-Autoregressive Methods

Non-autoregressive (NAR) approaches address the computational bottlenecks of sequential construction by generating solution representations in a single forward pass. [13] introduced *heatmaps* — probability distributions over edges indicating their likelihood of appearing in the optimal tour. Their Graph Convolutional Network (GCN)-based approach demonstrated the potential of direct edge prediction for solving the TSP.

Recent advances in NAR methods have explored various architectural and training innovations. [15] enhanced the heatmap approach with graph partitioning, where subheatmaps are created and merged back to build the complete heatmap for the entire graph. [16] proposed DIMES, employing RL with efficient sampling strategies for REINFORCE-based gradient estimation. [18] introduced DIFUSCO, a diffusion-based model that treats TSP as iterative denoising of edge selection variables, requiring fully supervised training on solutions at each scale. However, the denoising scheme of diffusion-based methods also requires multiple passes through the neural network during inference, similar to AR approaches, limiting their effectiveness.

A critical limitation of existing NAR methods is their dependence on compute-intensive and time-consuming search techniques to achieve competitive performance. These approaches typically rely on Monte Carlo Tree Search (MCTS) [15], [16] to convert heatmaps into valid TSP tours. Although MCTS improves solution quality, it trades off accuracy for increased execution time and computational resources. Moreover, as noted by [25], MCTS can achieve strong performance even without a neural model — simply by using the distance matrix

instead, indicating that MCTS is largely responsible for the method's overall effectiveness.

C. Training Paradigms

Training strategies for neural TSP solvers span supervised, unsupervised, and reinforcement learning paradigms. Supervised approaches [13], [18] require expensive optimal or near-optimal solutions as labels, limiting scalability to larger instances. Reinforcement learning methods [8]–[10], [16] optimize through environment interaction but often suffer from high variance and training instability. [17] demonstrated unsupervised learning for heatmap generation using geometric scattering-based networks with a heuristically designed loss function.

Self-improvement learning has emerged as a promising alternative [26]–[28], iteratively improving model predictions through self-generated pseudo-labels, which avoids the complexity of RL and the need for designing a reward function. For a given problem instance, [27] proposes to sample multiple solutions from the current model and select the best solution as pseudo-label for training AR models.

IV. Methodology

A. Model Architecture

Our architecture follows the standard Transformer design from [20], which was first used by [9] for solving the TSP. The details are described below:

1) Input Representation and Embedding: Let a TSP instance be defined by a set of N nodes, where each node i possesses features $x_i \in \mathbb{R}^{d_{in}}$. For standard 2D TSP instances, $d_{in} = 2$. The input features $X \in \mathbb{R}^{N\times 2}$ are first processed by a multi-layer perceptron (MLP) to embed them into a higher-dimensional space of dimension d_h. This initial embedding network allows the model to learn richer, non-linear feature representations of individual nodes before contextualization. We use a 3 layer MLP which is defined as:

$$H^{(0)} = \text{MLP}(X) = \text{ReLU}(\text{ReLU}(XW_1+b_1)W_2+b_2)W_3+b_3 \tag{1}$$

where W_k and b_k are learned parameters of the network layers. The resulting embedding matrix $H^{(0)} \in \mathbb{R}^{N\times d_h}$ serves as the input to the Transformer encoder.

2) Transformer Encoder: The encoder, composed of a stack of L identical layers, refines the node embeddings by capturing the global context of the graph structure. Each encoder layer contains two primary sub-layers: multi-head self-attention (MHA) and a feed-forward network (FFN). Residual connections and layer normalization are applied around each sub-layer. For an input $H^{(l-1)}$ to layer l, the computation proceeds as:

$$A^{(l)} = \text{LayerNorm}(H^{(l-1)} + \text{MHA}(H^{(l-1)})) \tag{2}$$

$$H^{(l)}_{enc} = \text{LayerNorm}(A^{(l)} + \text{FFN}(A^{(l)})) \tag{3}$$

The MHA mechanism computes scaled dot-product attention, allowing each node to aggregate information from all other nodes in the graph:

$$\begin{aligned}\text{MHA}(H) &= \text{Concat}(\text{head}_1,\dots,\text{head}_h)W^O,\\ &\text{where head}_i = \text{Attention}(HW_i^Q, HW_i^K, HW_i^V)\end{aligned} \tag{4}$$

The final output of the encoder stack, $H_{enc} \in \mathbb{R}^{N\times d_h}$, represents a set of contextually-rich node embeddings.

3) Transformer Decoder: The decoder has an additional cross-attention sub-layer, as in the standard Transformer. The decoder stack also consists of L layers. We feed the final encoder output H_{enc} as the initial input sequence to the decoder, i.e., $H^{(0)}_{dec} = H_{enc}$. Within each decoder layer l, the cross-attention mechanism uses queries generated from the output of the decoder's self-attention sub-layer, while keys and values are supplied by the encoder output H_{enc}:

$$A^{(l)}_{self} = \text{LayerNorm}(H^{(l-1)}_{dec} + \text{MHA}_{self}(H^{(l-1)}_{dec})) \tag{5}$$

$$\begin{aligned}A^{(l)}_{cross} = \text{LayerNorm}(&A^{(l)}_{self}\\ &+ \text{MHA}_{cross}(Q = A^{(l)}_{self}, K = H_{enc}, V = H_{enc}))\end{aligned} \tag{6}$$

$$H^{(l)}_{dec} = \text{LayerNorm}(A^{(l)}_{cross} + \text{FFN}(A^{(l)}_{cross})) \tag{7}$$

4) Output Probability Calculation: We first compute the *compatibility*, which we interpret as unnormalized log-probabilities (logits), similar to [9] for calculation of output probabilities. The compatibility is computed by combining the outputs from the encoder and decoder where the final decoder output $H_{dec} = H^{(L)}_{dec}$ serves as the query representations, and the final encoder output H_{enc} serves as key representations:

$$U = \frac{H_{dec}(H_{enc})^T}{\sqrt{d_h}} \tag{8}$$

The diagonal elements of the resulting matrix are masked to prevent self-loops ($U_{ii} = -\infty$). A softmax function is then applied row-wise to obtain the probability matrix P, where P_{ij} represents the likelihood of optimal edge from node i to node j:

$$P = \text{softmax}(U) \tag{9}$$

B. Training Pipeline

The model is trained using a combination of data augmentation and self-improvement learning. The training pipeline is described next and visualized in Figure 1.

1) Geometric Transformation of Graphs: Previous works [10], [24] have observed that TSP solutions remain unchanged under rotation and reflection of the problem instance. To exploit this fact, we apply geometric transformations to each training instance and generate $K-1$ augmented versions. First, we centre the graph by subtracting its centroid from the node coordinates. Next, we create $K-1$ copies, apply reflection across the y-axis to half of them, followed by random rotations to all $K-1$ copies, resulting in K total versions of each graph. This leads to a couple of advantages:

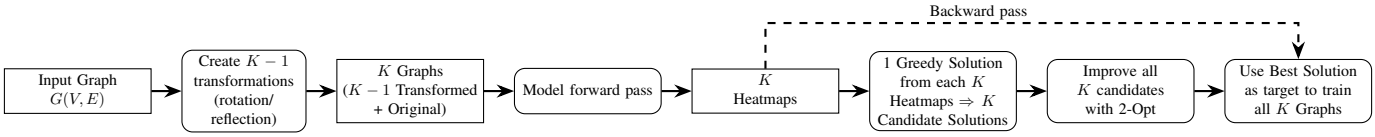

Fig. 1. Self-improvement training pipeline with geometric augmentation

- It enables the model to learn representations that are invariant to rotation and reflection, which improves generalization as the model learns to produce consistent solution across different geometric orientations of the same underlying problem structure.
- By sharing the pool of candidate solutions across the K versions for finding the training target for SIL, computational efficiency is significantly improved.

2) Edge-by-edge solution construction: Autoregressive (AR) TSP solvers construct tours node-by-node where only the edges incident to the current node are considered for selection in each step. In contrast, following previous non-autoregressive (NAR) works [16], [18], we adopt an alternative edge-by-edge approach which iteratively builds the tour by considering all edges in the graph and greedily adding the highest probability edge (assigned by the neural model) to the solution set in each step, subject to two constraints: (i) the degree of any node must not exceed 2, and (ii) no cycles should be formed prematurely. This construction method is detailed in Algorithm 1. When multiple edges have the same probability, we select the edge with the shortest length. This situation typically occurs when all valid edges available for selection have zero probability assigned by the model.

Notably, this edge-by-edge solution construction is not possible with AR methods, as it requires upfront probabilities for all edges in the graph, whereas AR models compute probabilities at each step only for the edges incident to the current node.

3) Self-Improvement Learning: Recent works [26]–[28] have shown that self-improvement learning (SIL) is an effective alternative to reinforcement learning (RL) for learning in the absence of labelled data for CO problems such as the TSP. We adapt the SIL framework in this work in the following ways—

- Unlike previous works where self-improvement is applied to AR models, we apply it to the NAR approach.
- Instead of sampling multiple solutions and selecting the best as the pseudo-label, we generate a single greedy solution and apply 2-opt improvement and use the improved solution as the pseudo-label.

a) Candidate Tour Generation: For each input graph and its $K-1$ augmentations, the model outputs the probability matrix (heatmap) P. For each P, we then generate a candidate solution using the edge-by-edge tour construction method described in Algorithm 1.

b) 2-Opt Improvement: We apply the classical 2-opt local search heuristic [29] to further improve each candidate tour. The 2-opt algorithm iteratively removes two non-adjacent edges from the current tour and reconnects the resulting paths by adding two new edges in the reversed configuration, provided this swap reduces the total tour length. This process continues until no improving 2-opt move exists, yielding a locally optimal tour with respect to the 2-opt neighbourhood.

c) Pseudo-Label Generation: After generating a pool of candidate tours, we evaluate each tour by calculating the tour length. The edges of the tour τ^* with the shortest tour length

Algorithm 1 Edge-by-edge TSP tour construction

Input: Symmetric cost matrix $C \in \mathbb{R}_+^{N\times N}$, $C_{ii} = \infty$
Input: Row-normalized probability matrix $P \in [0,1]^{N\times N}$
Output: TSP tour (as set of N edges)

$Q_{ij} \leftarrow P_{ij} \cdot P_{ji} \quad \forall 1 \le i < j \le N$
Initialize valid edges: $\mathcal{E} \leftarrow \{\{i,j\} \mid 1 \le i < j \le N\}$
Initialize empty tour: $\tau \leftarrow \emptyset$
Initialize node degree: $d[v] \leftarrow 0 \quad \forall v \in \{1,\dots,N\}$
Initialize union-find data structure UF with N singleton components
while $|\tau| < N$ **do**
  $Q^\star \leftarrow \max_{e'\in\mathcal{E}} Q_{e'}$
  if $Q^\star > 0$ **then**
    $e \leftarrow \arg\max_{e'\in\mathcal{E}} Q_{e'}$
  else
    $e \leftarrow \arg\min_{e'\in\mathcal{E}} C_{e'}$
  end if
  $\tau \leftarrow \tau \cup \{e\}$
  Let $\{u, v\} \leftarrow e$
  $d[u] \leftarrow d[u] + 1$
  $d[v] \leftarrow d[v] + 1$
  if $d[u] = 2$ **then**
    Remove all remaining edges incident to u from $\mathcal{E}$
  end if
  if $d[v] = 2$ **then**
    Remove all remaining edges incident to v from $\mathcal{E}$
  end if
  $\mathsf{UF}.\mathrm{union}(u, v)$
  if $|\tau| < N - 1$ **then**
    Remove from $\mathcal{E}$ all $e' = \{i,j\} \in \mathcal{E}$ such that $\mathsf{UF}.\mathrm{find}(i) = \mathsf{UF}.\mathrm{find}(j)$
  end if
end while
return Constructed tour edge list τ

TABLE I
RESULTS ON TSP-500 AND TSP-1000. RESULTS OF BASELINES ARE TAKEN DIRECTLY FROM THE ORIGINAL PAPERS. ABBREVIATIONS — SL: SUPERVISED LEARNING, RL: REINFORCEMENT LEARNING, UL: UNSUPERVISED LEARNING, MCTS: MONTE CARLO TREE SEARCH, G: GREEDY, RRC: RANDOM RE-CONSTRUCT, BS: BEAM SEARCH

METHOD	TYPE	TSP-500			TSP-1000		
		LEN. ↓	GAP ↓	TIME ↓	LEN. ↓	GAP ↓	TIME ↓
Concorde	Exact Solver	16.546	0.00%	37.66m	23.118	0.00%	6.65h
Autoregressive methods							
LEHD	SL+G	—	1.560%	0.3m	—	3.168%	1.6m
BQ-NCO	SL+G	—	1.18%	55s	—	2.29%	2m
LEHD	SL+RRC 50	—	0.482%	3.4m	—	1.416%	22m
	SL+RRC 100	—	**0.343%**	8m	—	1.218%	43m
BQ-NCO	SL+BS16	—	0.55%	15m	—	1.38%	38m
Non-autoregressive methods							
DIMES	RL+G	18.93	14.38%	0.97m	26.58	14.97%	4.65m
DIFUSCO	SL+G	18.35	10.85%	3.61m	26.14	13.06%	11.86m
Fast $T2T_1$	SL+G	17.80	7.57%	17s	25.23	9.13%	55s
Fast $T2T_2$	SL+G	**16.93**	**2.33%**	2.20m	**23.96**	**3.64%**	9.20m
NARSIL (Ours)	**SIL+G**	17.126	3.272%	**1.50s**	24.644	6.102%	**3.37s**
Att-GCN	SL+MCTS	16.966	2.537%	5.91m	23.863	3.224%	12.47m
DIMES	RL+MCTS	16.87	1.93%	2.92m	23.73	2.64%	6.87m
UTSP	UL+MCTS	16.685	0.839%	2.7m	23.390	1.177%	6.02m
DIFUSCO	SL+MCTS	16.63	0.46%	10.13m	23.39	1.17%	24.47m
DIFUSCO	SL+G+2-opt	16.80	1.49%	3.65m	23.56	1.90%	12.06m
SoftDist	SoftDist+MCTS	16.78	1.44%	1.67m	23.63	2.20%	3.34m
Fast $T2T_1$	SL+G+2-opt	16.75	1.23%	15s	23.45	1.42%	57s
Fast $T2T_2$	SL+G+2-opt	**16.61**	**0.39%**	2.17m	**23.25**	**0.58%**	8.62m
NARSIL (Ours)	**SIL+G+2-opt**	16.653	0.420%	**1.81s**	23.440	0.916%	**4.91s**

are selected as the pseudo-labels for training.

$$\tau^* = \underset{\tau \in \{\tau_1, \tau_2, \ldots, \tau_K\}}{\arg\min} \text{Cost}(\tau) \tag{10}$$

d) Loss function: The model parameters θ are updated by minimizing the negative log-likelihood of generating the best-found tour τ^*. This is similar to supervised learning, but the model's output is used to generate the pseudo-labels. As the training progresses and the model improves, so does the quality of pseudo-labels, creating a virtuous cycle and enabling it to discover even better solutions in subsequent iterations.

$$\mathcal{L}(\theta) = -\mathbb{E}_{G \sim \mathcal{D}} \left[\frac{1}{2 \cdot |\tau|} \sum_{(i,j) \in \tau^*} \log P_{ij}(G;\theta) + \log P_{ji}(G;\theta) \right] \tag{11}$$

C. Inference Strategy

We perform evaluation on the original graph without any additional geometric augmentations for fair comparison with other methods. For each problem instance, the transformer model outputs probability heatmap, on which we perform greedy edge-by-edge tour construction following Algorithm 1. The constructed tour is then further improved with 2-opt.

V. EXPERIMENTS

Hyperparameters. We use a balanced encoder-decoder architecture where both the encoder and decoder have $L = 12$ layers each. The attention layers have 8 heads, each with embedding dimension of 16 ($d_h = 8 \times 16 = 128$). The feed-forward layers have hidden dimension of $4 \times d_h = 512$. We set $K = 2$, which means only 1 additional geometrically transformed version is used for each problem instance. We generate a greedy solution following Algorithm 1, for all $K = 2$ graphs. We avoid using larger values of K, since we found during experimentation that the model learns faster from more diverse data in the batch.

TABLE II
RELATIVE RUNTIME OF BASELINES AGAINST NARSIL ON TSP-500 AND TSP-1000

Method	AR/NAR	Training	Relative runtime TSP-500	Relative runtime TSP-1000
NARSIL (G+2-opt)	**NAR**	**Unsupervised**	**1×**	**1×**
Fast $T2T_1$ (G+2-opt)	NAR	Supervised	8×	11×
LEHD (G)	AR	Supervised	9×	19×
BQ-NCO (G)	AR	Supervised	30×	24×
SoftDist (MCTS)	NAR	Unsupervised	55×	40×
UTSP (MCTS)	NAR	Unsupervised	89×	73×
DIMES (MCTS)	NAR	Unsupervised	96×	83×
Fast $T2T_2$ (G+2-opt)	NAR	Supervised	71×	105×
DIFUSCO (G+2-opt)	NAR	Supervised	120×	147×
Att-GCN (MCTS)	NAR	Supervised	195×	152×
LEHD (RRC 50)	AR	Supervised	112×	268×
DIFUSCO (MCTS)	NAR	Supervised	335×	299×
BQ-NCO (BS16)	AR	Supervised	497×	464×
LEHD (RRC 100)	AR	Supervised	265×	525×
Concorde	—	Exact	1248×	4875×

Training. We train the model from scratch starting with small graphs ($N = 20$) and continue training with a progressive curriculum of increasing graph sizes till $N = 1000$ in successive stages. We employ a Cosine Annealing with Warm Restarts schedule [30], with a maximum learning rate $\eta_{\max} = 1 \times 10^{-4}$. Adam [31] is used as the optimizer for training through back-propagation.

As the graph size changes, we adjust the batch size such that the number of tokens (nodes) trained per batch remains constant. This ensures that GPU memory requirements remain roughly consistent, irrespective of the graph size. Specifically, we set the number of nodes per batch (per GPU) to 6000, resulting in a batch size (per GPU) of $6000/500 = 12$ for TSP-500 and $6000/1000 = 6$ for TSP-1000. Training is conducted on $4\times$ NVIDIA RTX 2080 Ti GPUs in parallel, yielding an effective batch size of $4\times$ the aforementioned per-GPU values.

Baselines. We compare our approach with: (1) *Classical Solver:* Concorde [2] (2) *AR methods*: BQ-NCO [12], LEHD [11] (3) *NAR methods*: Att-GCN [15], DIMES [16], DIFUSCO [18], UTSP [17], SoftDist [25], Fast T2T [19].

Testing. We evaluate NARSIL on Euclidean TSP instances (uniform $[0,1]^2$ coordinates), following previous works. Test datasets for TSP-500 and TSP-1000 (128 test instances each) are taken from [15]. All stages of the solution generation process of NARSIL (given in Table III) are performed on a single RTX 2080 Ti GPU.

A. Results

Main results are presented in Table I. The authors of Fast T2T reported their results in many different settings. We have included the fastest setting (denoted with Fast $T2T_1$) and the highest accuracy setting (denoted with Fast $T2T_2$). On TSP-500, NARSIL with 2-opt achieves a gap of 0.420% in **1.81 seconds** — nearly matching the best reported gap of

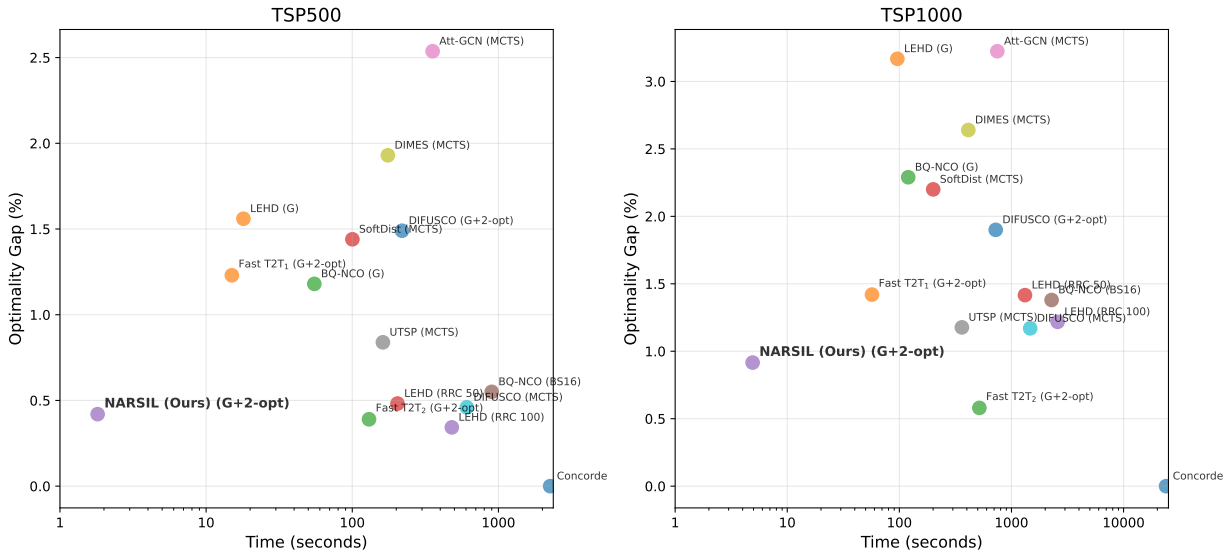

Fig. 2. Solution quality and runtime trade-off of various baselines and NARSIL on TSP-500 and TSP-1000. Time on x-axis is in logarithmic scale.

TABLE III
RUNTIME BREAKDOWN OF NARSIL (IN SECONDS) FOR TSP-500 AND TSP-1000 (ON SINGLE RTX 2080 TI)

	TSP-500	TSP-1000
Heatmap generation	0.595	1.573
Greedy decoding	0.904	1.799
2-opt	0.307	1.534
Total Time	1.806	4.906

Fast $T2T_2$ (0.39%, 2.17 minutes) but with dramatically lower latency. Against MCTS-based NAR methods e.g., SOFTDIST (1.44% gap, 1.67 minutes), DIFUSCO (1.17% gap, 24.47 minutes), UTSP (1.177% gap, 6.02 minutes), NARSIL with 2-opt delivers comparable or superior quality at 55–335× lower inference time. Similar trends hold for the larger TSP-1000 instances. NARSIL with 2-opt reaches a 0.916% gap in **4.91 seconds**, approaching Fast $T2T_2$'s 0.58% gap (8.62 minutes) at a fraction of the runtime.

Figure 2 illustrates the speed–accuracy trade-off of the evaluated methods. NARSIL stands out by achieving the best inference speed while delivering solution quality that remains very competitive (often near or on par) with the strongest baselines.

Table II quantifies the relative inference runtime normalized to NARSIL (= 1×). NARSIL (G+2-opt) is the fastest neural method by a wide margin. Fast $T2T_1$ (G+2-opt) is 8–11× slower, Fast $T2T_2$ (G+2-opt) is 71–105× slower, while MCTS-enhanced baselines range from 55× (SoftDist+MCTS) to 335× (DIFUSCO+MCTS) on TSP-500. Supervised AR methods with refinement (LEHD RRC 100, BQ-NCO BS16) are 265–525× slower. Concorde is slower than NARSIL by 1248× in TSP-500 and by 4875× in TSP-1000.

These results demonstrate that NARSIL delivers either superior or highly competitive solution quality while achieving 1–2 orders of magnitude faster inference. The speed advantage is particularly pronounced on the larger TSP-1000 instances, highlighting the practical efficiency of NARSIL for real-world deployment scenarios where both solution quality and low latency are critical.

B. Comparison of Self-Improvement Learning (SIL) with Supervised Learning (SL) on TSP20

The SL setup remains identical to our SIL setup, the only difference here is that the labels are not self-generated. Instead, we created a training dataset of 100,000 TSP20 instances, with optimal solutions generated by Concorde as labels.

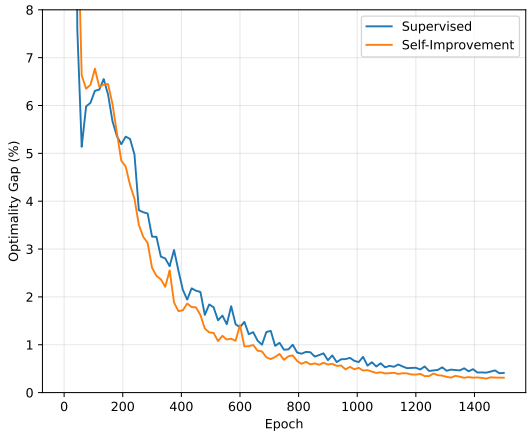

Fig. 3. Convergence speed of different training algorithms on TSP-20

Figure 3 shows that SIL outperforms SL in convergence speed. It may seem counter-intuitive, given that SL is trained on ground-truth optimal labels, whereas SIL relies on noisier pseudo-labels which are suboptimal. The key reason is data

scale — SL is constrained to a fixed dataset (100,000 labelled instances in this case), while SIL can generate and learn from a practically unlimited number of unlabelled instances with self-generated targets. This massive increase in effective training data more than compensates for the reduced label quality, leading to faster convergence.

VI. Conclusion

In this work, we introduced NARSIL, a non-autoregressive (NAR) approach trained in an unsupervised manner through self-improvement learning (SIL) that solves the Traveling Salesman Problem (TSP) in a significantly efficient manner. NARSIL sets a new standard for inference efficiency as it delivers highly competitive solution quality — either outperforming or closely approaching the best supervised baselines — while running orders of magnitude faster. Importantly, it achieves this performance without requiring expensive optimal solutions during training. This unique combination of near-optimal quality and unprecedented speed makes NARSIL especially well-suited for large-scale, real-time, or resource-constrained applications.

References

[1] J. K. Lenstra and A. R. Kan, "Some simple applications of the travelling salesman problem," *Journal of the Operational Research Society*, 1975.

[2] D. L. Applegate, R. E. Bixby, V. Chvátal, and W. J. Cook, "Concorde TSP Solver," 2006. [Online]. Available: http://www.math.uwaterloo.ca/tsp/concorde/

[3] K. Helsgaun, "LKH-3 (3.0.7)," 2017. [Online]. Available: http://webhotel4.ruc.dk/~keld/research/LKH-3/

[4] E. D. Taillard and K. Helsgaun, "Popmusic for the travelling salesman problem," *European Journal of Operational Research*, vol. 272, no. 2, pp. 420–429, 2019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0377221718305745

[5] N. Vesselinova, R. Steinert, D. F. Perez-Ramirez, and M. Boman, "Learning combinatorial optimization on graphs: A survey with applications to networking," *IEEE Access*, vol. 8, pp. 120 388–120 416, 2020.

[6] Y. Bengio, A. Lodi, and A. Prouvost, "Machine learning for combinatorial optimization: a methodological tour d'horizon," *European Journal of Operational Research*, vol. 290, no. 2, pp. 405–421, 2021.

[7] O. Vinyals, M. Fortunato, and N. Jaitly, "Pointer networks," in *Advances in Neural Information Processing Systems*, vol. 28, 2015, pp. 2692–2700.

[8] I. Bello, H. Pham, Q. V. Le, M. Norouzi, and S. Bengio, "Neural combinatorial optimization with reinforcement learning," in *International Conference on Machine Learning (Workshop)*, 2017.

[9] W. Kool, H. van Hoof, and M. Welling, "Attention, learn to solve routing problems!" in *International Conference on Learning Representations*, 2019. [Online]. Available: https://openreview.net/forum?id=ByxBFsRqYm

[10] Y.-D. Kwon, J. Choo, B. Kim, I. Yoon, Y. Gwon, and S. Min, "Pomo: Policy optimization with multiple optima for reinforcement learning," *Advances in Neural Information Processing Systems*, vol. 33, pp. 21 188–21 198, 2020.

[11] F. Luo, X. Lin, F. Liu, Q. Zhang, and Z. Wang, "Neural combinatorial optimization with heavy decoder: Toward large scale generalization," in *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. [Online]. Available: https://openreview.net/forum?id=RBI4oAbdpm

[12] D. Drakulic, S. Michel, F. Mai, A. Sors, and J.-M. Andreoli, "BQ-NCO: Bisimulation quotienting for efficient neural combinatorial optimization," in *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. [Online]. Available: https://openreview.net/forum?id=BRqlkTDvvm

[13] C. K. Joshi, T. Laurent, and X. Bresson, "An efficient graph convolutional network technique for the travelling salesman problem," ArXiV, arXiv preprint arXiv:1906.01227, 2019.

[14] C. B. Browne, E. Powley, D. Whitehouse, S. M. Lucas, P. I. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, and S. Colton, "A survey of monte carlo tree search methods," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 4, no. 1, pp. 1–43, 2012.

[15] Z.-H. Fu, K.-B. Qiu, and H. Zha, "Generalize a small pre-trained model to arbitrarily large tsp instances," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 35, no. 8, pp. 7474–7482, May 2021. [Online]. Available: https://ojs.aaai.org/index.php/AAAI/article/view/16916

[16] R. Qiu, Z. Sun, and Y. Yang, "DIMES: A differentiable meta solver for combinatorial optimization problems," in *Advances in Neural Information Processing Systems*, A. H. Oh, A. Agarwal, D. Belgrave, and K. Cho, Eds., 2022. [Online]. Available: https://openreview.net/forum?id=9u05zr0nhx

[17] Y. Min, Y. Bai, and C. P. Gomes, "Unsupervised learning for solving the travelling salesman problem," in *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. [Online]. Available: https://openreview.net/forum?id=lAEc7aIW20

[18] Z. Sun and Y. Yang, "DIFUSCO: Graph-based diffusion solvers for combinatorial optimization," in *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. [Online]. Available: https://openreview.net/forum?id=JV8Ff0lgVV

[19] Y. Li, J. Guo, R. Wang, H. Zha, and J. Yan, "Fast t2t: Optimization consistency speeds up diffusion-based training-to-testing solving for combinatorial optimization," in *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024. [Online]. Available: https://openreview.net/forum?id=xDrKZOZEOc

[20] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in Neural Information Processing Systems*, vol. 30, 2017, pp. 6000–6010.

[21] L. Xin, W. Song, Z. Cao, and J. Zhang, "Multi-decoder attention model with embedding glimpse for solving vehicle routing problems," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 35, no. 13, pp. 12 042–12 049, May 2021. [Online]. Available: https://ojs.aaai.org/index.php/AAAI/article/view/17430

[22] Y.-D. Kwon, J. Choo, I. Yoon, M. Park, D. Park, and Y. Gwon, "Matrix encoding networks for neural combinatorial optimization," *Advances in Neural Information Processing Systems*, vol. 34, pp. 5138–5149, 2021.

[23] J. Li, L. Xin, Z. Cao, A. Lim, W. Song, and J. Zhang, "Heterogeneous attentions for solving pickup and delivery problem via deep reinforcement learning," *IEEE Transactions on Intelligent Transportation Systems*, vol. 23, no. 3, pp. 2306–2315, 2021.

[24] M. Kim, J. Park, and J. Park, "Sym-NCO: Leveraging symmetricity for neural combinatorial optimization," in *Advances in Neural Information Processing Systems*, A. H. Oh, A. Agarwal, D. Belgrave, and K. Cho, Eds., 2022. [Online]. Available: https://openreview.net/forum?id=kHrE2vi5Rvs

[25] Y. Xia, X. Yang, Z. Liu, Z. Liu, L. Song, and J. Bian, "Position: Rethinking post-hoc search-based neural approaches for solving large-scale traveling salesman problems," in *Forty-first International Conference on Machine Learning*, 2024. [Online]. Available: https://openreview.net/forum?id=cEJ9jNJuJP

[26] A. Corsini, A. Porrello, S. Calderara, and M. Dell'Amico, "Self-labeling the job shop scheduling problem," *arXiv preprint arXiv:2401.11849*, 2024.

[27] J. Pirnay and D. G. Grimm, "Self-improvement for neural combinatorial optimization: Sample without replacement, but improvement," *Transactions on Machine Learning Research*, 2024, featured Certification. [Online]. Available: https://openreview.net/forum?id=agT8ojoH0X

[28] F. Luo, X. Lin, Y. Wu, Z. Wang, T. Xialiang, M. Yuan, and Q. Zhang, "Boosting neural combinatorial optimization for large-scale vehicle routing problems," in *The Thirteenth International Conference on Learning Representations*, 2025. [Online]. Available: https://openreview.net/forum?id=TbTJJNjumY

[29] G. A. Croes, "A method for solving traveling-salesman problems," *Operations Research*, vol. 6, pp. 791–812, 1958.

[30] I. Loshchilov and F. Hutter, "SGDR: Stochastic gradient descent with warm restarts," in *International Conference on Learning Representations*, 2017. [Online]. Available: https://openreview.net/forum?id=Skq89Scxx

[31] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.