# Learning-Guided Local Search for Asymmetric Traveling Salesman Problem

**Lejun Zhou** [1] **Yi Ju** [1] **Scott Moura** [1]

## Abstract

The Asymmetric Traveling Salesman Problem (ATSP) is a generalization of the well-known NP-hard Traveling Salesman Problem (TSP), where edge costs depend on the direction of travel. While recent learning-based solvers for TSP use machine-learned heatmaps to guide Monte Carlo Tree Search (MCTS), we find that MCTS alone drives most of the performance, with heatmaps offering limited standalone value. Moreover, existing MCTS methods are largely restricted to 2D Euclidean TSPs, limiting real-world applicability. To address these gaps, we propose a new model combining an autoencoder and Graph Convolutional Network (GCN) to generate more informative heatmaps. We also design a tailored MCTS pipeline for ATSP, overcoming limitations of prior frameworks. Our approach achieves state-of-the-art results among learning-based methods on ATSP with low inference-time cost.

## 1. Introduction

The Vehicle Routing Problem (VRP) is a fundamental combinatorial optimization challenge with broad applications in logistics, transportation, and supply chain management. The goal is to determine optimal routes for a fleet of vehicles to visit a set of locations while minimizing a given cost function, such as travel distance, time, or fuel consumption.

As a key special case of the Vehicle Routing Problem (VRP), the Traveling Salesman Problem (TSP) involves a single vehicle visiting all locations exactly once before returning to the starting point. TSP is extensively studied as a fundamental benchmark in combinatorial optimization, serving as a foundation for the development and evaluation of a wide range of algorithms (Pop et al., 2024). Despite its simplified structure, TSP remains a classic NP-hard problem, with the number of possible solutions growing factorially with the number of locations. Consequently, finding exact solutions for large-scale instances becomes computationally intractable, underlining TSP's importance as a benchmark for designing efficient optimization methods (Lawler, 1985; Applegate, 2006).

A more general and complex variant of the Traveling Salesman Problem is the Asymmetric Traveling Salesman Problem (ATSP), in which the cost of traveling from city $i$ to city $j$ is not necessarily equal to the cost of traveling from $j$ to $i$. This asymmetry naturally arises in many real-world contexts, such as one-way street networks, direction-dependent traffic conditions, and constraints in logistics systems. Unlike the symmetric TSP, where the solution space can be significantly reduced due to bidirectional symmetry, the ATSP presents unique structural challenges that render many classical algorithms ineffective or inefficient. The ATSP not only increases the solution space but also alters the combinatorial landscape of the problem, requiring specialized algorithms and transformation techniques (Gutin & Punnen, 2006). These directional dependencies demand more sophisticated modeling and optimization strategies, making ATSP a critical testbed and benchmarking tool for advanced heuristics and learning-based solvers.

Traditionally, TSP and ATSP have been tackled using mathematical programming techniques from the operations research (OR) community. Early approaches relied on Integer Linear Programming (ILP) and Mixed-Integer Linear Programming (MILP) formulations, including the Miller-Tucker-Zemlin (MTZ) constraints (Miller et al., 1960) and the Dantzig-Fulkerson-Johnson (DFJ) formulations (Dantzig et al., 1954). These methods guarantee optimality but suffer from exponential solving times as problem size increases. To mitigate this, heuristics and metaheuristics have been developed, such as the Lin-Kernighan heuristic (Lin & Kernighan, 1973), Simulated Annealing (Kirkpatrick et al., 1983), and Genetic Algorithms (Holland, 1992). While these methods provide high-quality approximations, they still face fundamental limitations. Exact methods remain impractical for large instances due to their high computational complexity. Meanwhile, heuristic approaches lack guarantees on solution quality while often requiring extensive parameter tuning. Moreover, formulating TSP and ATSP constraints for real-world scenarios

[1]Department of Civil & Environmental Engineering, University of California, Berkeley, USA. Correspondence to: Lejun Zhou <lejun@berkeley.edu>.

*The second AI for MATH Workshop at the 42nd International Conference on Machine Learning*, Vancouver, Canada. 

further increases model complexity, making these traditional techniques challenging to apply at scale (Laporte, 1992).

In recent years, learning-based approaches have emerged as an alternative for solving combinatorial optimization problems, including TSP. One of the earliest works in this direction, Pointer Networks (Vinyals et al., 2015), demonstrated the feasibility of using sequence-to-sequence models for TSP by leveraging attention mechanisms. Later, reinforcement learning-based methods, such as Neural Combinatorial Optimization (Bello et al., 2016) and the Attention Model (Kool et al., 2018), introduced policy gradient approaches for direct solution generation. Other methods, such as Graph Neural Networks (GNNs) (Khalil et al., 2017), further improved generalization by embedding combinatorial structures into graph representations. These learning-based methods showed promise in approximating solutions efficiently, but they struggled with scaling to large problem instances and often failed to outperform strong handcrafted heuristics.

To enhance scalability, recent studies have explored heatmap-guided Monte Carlo Tree Search (MCTS) as a hybrid learning-and-search paradigm. In this framework, machine learning models generate heatmaps that estimate the likelihood of each edge belonging to the optimal solution. These heatmaps are then used as priors within MCTS to guide the search toward high-quality solutions (Fu et al., 2021; Qiu et al., 2022; Min et al., 2023; Sun & Yang, 2023). By combining data-driven generalization with search-based refinement, this approach achieves low optimality gaps on large-scale TSP instances. However, despite its potential, heatmap-guided MCTS faces critical limitations. The generated heatmaps often provide only coarse or noisy guidance, resulting in poor performance when used to generate solutions without MCTS. Besides, the MCTS procedure remains computationally intensive, especially for large-scale instances. These challenges raise a fundamental question: Are heatmaps truly crucial for enhancing solution quality, or is the observed performance gain primarily attributable to the computationally expensive MCTS simulation itself? (Xia et al., 2024)

In addition, although dozens of research papers on learning-based approaches for TSP/VRP were published[1], most models are exclusively tailored to Euclidean instances[2]. Only very limited (and to some extent isolated) research has been performed on TSP/VRPs whose distance matrices are not induced by a Euclidean norm. To the best of the authors' knowledge, MatNet (Kwon et al., 2021) , BQ-NCO (Drakulic et al., 2023) and UniCO (Pan et al., 2025), are the only three papers which make general and systematic contributions to this problem. Although MatNet was published 3 years ago, few papers include a task on non-Euclidean TSP when evaluating their innovations. As one example, GLOP (Ye et al., 2024) did include an ATSP task which directly used pre-trained MatNet models as sub-solver.

Furthermore, existing MCTS-based methods are inherently tied to the symmetric structure of classical TSP, where properties such as bidirectional edge equivalence simplify search and pruning. This reliance on symmetry makes them ill-suited for direct application to ATSP. To date, no significant innovation has been introduced to adapt MCTS for asymmetric problems, underscoring the need for novel algorithmic developments in this space.

Building on these insights, we introduce a new MCTS framework specifically adapted to the Asymmetric Traveling Salesman Problem (ATSP), thereby extending its applicability to a broader class of combinatorial optimization problems. In addition, we propose a novel model that integrates an autoencoder with a Graph Convolutional Network (GCN) to generate heatmaps specifically tailored to complement MCTS. Our key contributions are as follows:

- **Reevaluating heatmaps and MCTS:** We revisit the contributions of heatmaps in previous learning-based solvers and show that their standalone effectiveness is limited, with MCTS playing the dominant role in optimization. In addition, we analyze the limitations of existing MCTS frameworks—such as their reliance on random initialization and symmetric assumptions—which hinder their applicability to asymmetric problems like ATSP.

- **Enhancing heatmap quality via model design:** We propose a new heatmap generation model that integrates an autoencoder with a Graph Convolutional Network (GCN). The autoencoder enables the model to capture global structural patterns and compress relevant information to low dimensional node features, resulting in more informative and effective heatmaps.

- **Extending MCTS to ATSP:** With the structural properties of ATSP, we design a modified MCTS framework capable of handling asymmetry in edge costs. This extension enables MCTS to solve a broader class of combinatorial problems beyond symmetric TSP, with improved performance on ATSP benchmarks.

[1]The github page Awesome Machine Learning for Combinatorial Optimization Resources collects 69 papers under the TSP section and 41 papers under the VRP section (accessed on 18 May, 2025).

[2]In this paper, by "non-Euclidean" instances, we primarily refer to the class of Asymmetric TSP (ATSP). These are characterized by distance matrices that may satisfy the triangle inequality but are not necessarily symmetric or derived from any underlying norm.

## 2. Preliminary Analysis

### 2.1. Evaluating Heatmaps

Recent learning-based solvers often rely on machine learning models to generate heatmaps, which are then used as priors in Monte Carlo Tree Search (MCTS) to guide solution construction. However, these works typically report only the final performance achieved with MCTS, leaving the true effectiveness of the heatmaps themselves unclear. To evaluate their role in the optimization process, we conduct an empirical study by assessing solutions for TSP instances constructed directly by heatmaps — without applying MCTS. Specifically, we use previously proposed model UTSP (Min et al., 2023) to generate heatmaps and construct solutions by greedily selecting edges with the highest probabilities. As a performance metric, we adopt the *optimality gap*, defined as the percentage difference between the obtained solution and the known optimal solution. Our results show that solutions derived solely from heatmaps exhibit substantial optimality gaps, indicating that heatmaps alone fail to reliably capture the structure of high-quality solutions.

To further assess whether the learned heatmaps offer superior guidance, we compare them against two straightforward baselines: (1) a naïve heuristic-based heatmap (referred to as *SoftDist* in the remainder of this paper), which assigns probabilities to edges based on their distances using a softmax function that favors shorter edges (Xia et al., 2024), and (2) a random heatmap with uniform edge probabilities. To ensure fairness, we allocate the same average MCTS search time $T = 0.01 \times N$ per instance across all methods, where $T$ denotes the average search time per instance (in seconds), and $N$ is the size of the TSP instance. For example, a TSP instance with 200 nodes is allocated a time budget of $0.01 \times 200 = 2$ seconds.

Surprisingly, solutions constructed using the *SoftDist* heatmap and the random heatmap—when combined with MCTS—perform comparably to, and in some cases even better than, those guided by certain state-of-the-art learned models. The corresponding optimality gaps are reported in Table 1, and visualizations of the generated solutions are provided in Appendix A.

These findings highlight a fundamental limitation of existing heatmap-based approaches: they often fail to learn meaningful structural information during training, preventing them from consistently encoding the fine-grained characteristics of optimal solutions. As a result, when used without additional refinement, such heatmaps may mislead the search process rather than enhance it. These observations underscore the need to rethink how to improve the informativeness and reliability of learned heatmaps in combinatorial optimization.

*Table 1.* Optimality Gap Comparison for Different Approaches

| Method | 100 | 200 | 500 | 1000 |
|---|---|---|---|---|
| LH+G | 43.7% | 37.7% | 44.3% | 40.2% |
| HH+G | 24.8% | 25.5% | 25.5% | 24.4% |
| LH+M | 0.03% | 0.76% | 4.23% | 5.62% |
| HH+M | 0.03% | 0.80% | 4.07% | 4.26% |
| RH+M | 0.03% | 0.75% | 4.58% | 4.85% |

**Description:** The table reports the average *optimality gap* (%) of different approaches on TSP instances with 100, 200, 500, and 1000 nodes, with 100 instances evaluated per node size. **LH** refers to heatmaps generated by model UTSP, **HH** refers to *SoftDist* heatmaps, and **RH** indicates random heatmaps. **G** represents greedy inference, where edges with the highest probabilities are selected directly, while **M** denotes MCTS-based inference, where heatmaps serve as priors to guide the search. In the table, lower optimality gap values correspond to higher solution quality.

### 2.2. Evaluating MCTS

Monte Carlo Tree Search, as applied to the TSP in previous papers, operates by first generating an initial solution, then applying 2-OPT, followed by a $k$-OPT[3] search procedure guided by heatmaps (hereafter referred to as MCTS for brevity). A brief overview of this MCTS framework is provided in Appendix B, and we refer interested readers to the original work (Fu et al., 2021) for a more comprehensive treatment.

Although previous studies demonstrated that MCTS was capable of producing near-optimal solutions for TSP, a closer examination revealed several critical limitations:

1. **Lack of heatmap usage in initialization.** The previous MCTS framework did not use the heatmap to generate the initial tour; instead, it relied on random initialization followed by local search improvements. This design choice was motivated by two key considerations. First, previous heatmaps often lacked the granularity needed to construct informative initial tours. Second, avoiding deterministic initialization promoted exploration, as random starts increased the likelihood of escaping local optima and discovering globally competitive solutions. However, this strategy was computationally intensive, as it required many iterations to reliably obtain a high-quality starting tour.

2. **Dependence on TSP symmetry.** The previous MCTS framework implicitly assumed symmetric edge costs, both in its 2-OPT and $k$-OPT simulation steps. During each local move, it evaluated the cost difference

[3] $k$-OPT is a local search heuristic that improves a given tour by removing $k$ edges and reconnecting the remaining segments in a different way to obtain a shorter tour. The most common variants are 2-OPT and 3-OPT. (Helsgaun, 2009)

between the tours using a simple formula:

$$\Delta = \text{cost of removed edges} - \text{cost of added edges},$$

and then performed a reversal of the affected sub-tour. While this was effective for symmetric TSPs, it was inappropriate for ATSP, where reversing a sub-tour did not preserve cost due to asymmetry in the distance matrix.

3. **Limited contribution from $k$-OPT.** In our experiments, we observed that most improvements of previous MCTS framework occurred during the 2-OPT phase, with the $k$-OPT search contributing little to the overall solution quality. This suggested that the effectiveness of MCTS was largely driven by 2-OPT, which did not utilize the heatmap. This observation helped explain why a wide range of heatmaps, regardless of quality, tended to produce competitive results when combined with MCTS—because the heatmap itself played a minor role.

Given these limitations, it becomes evident that the previous MCTS framework is not well-suited for solving ATSP instances. To address these issues, we propose a new MCTS pipeline specifically tailored for ATSP. Our framework incorporates heatmap-based initialization, handles asymmetric edge costs explicitly during local search, and leverages improved edge evaluation strategies to better exploit the structural information captured by learned heatmaps. The details of this modified pipeline are presented in the following sections.

### 2.3. Transformations for ATSP

The Asymmetric Traveling Salesman Problem (ATSP) is a generalization of the classic Traveling Salesman Problem (TSP), where the cost of traveling between two locations depends on the direction of travel. Formally, given a set of $n$ cities and a distance matrix $D \in \mathbb{R}^{n \times n}$, the objective is to find a Hamiltonian cycle (visiting each city exactly once) while minimizing the total travel cost. Unlike TSP, where the distance matrix is symmetric ($D = D^\top$), ATSP instances are characterized by **asymmetric costs** ($D \neq D^\top$).

The inherent asymmetry of the ATSP introduces significant complexity, as many optimization techniques developed for the symmetric TSP rely on the assumption that reversing a segment of a tour does not alter its cost (see Section 2.2). Consequently, these methods often perform poorly when applied directly to the ATSP, underscoring the need for specialized algorithms.

Inspired by the classical transformation proposed by Jonker and Volgenant (Jonker & Volgenant, 1983), which converts an ATSP instance into an equivalent expanded symmetric TSP, we extend this idea beyond merely transforming the distance matrix. Specifically, we apply the transformation to the learned heatmap, guided by structural insights into the ideal solution of the expanded symmetric instance.

The details of this heatmap-based transformation process are provided in Appendix C. The resulting transformed heatmap is compatible with downstream search operations and significantly enhances the effectiveness of MCTS in the asymmetric setting.

## 3. Methodology

### 3.1. Model Inference Pipeline

Our approach begins with a *distance matrix* that defines the cost between all node pairs in an ATSP instance. This matrix is first processed by an **autoencoder**, which aims to capture global structural patterns through a low dimensional representation, which can be treated as node features afterwards. The resulting node features augment the original edge features (distance matrix), are then fed into a **Graph Convolutional Network (GCN)** to generate a *heatmap*—a matrix estimating the likelihood that each edge belongs to the optimal tour.

We apply a transformation that expands both the original distance matrix and the heatmap, as detailed in Appendix C. This heatmap serves as a prior to guide our refined **MCTS** framework tailored for ATSP. Unlike previous approaches, our MCTS leverages the heatmap to generate initial tours for further refinement. Then the search process is restricted to **3-OPT** operations, guided by a potential function derived from the heatmap. The tour is iteratively refined until the predefined time budget is exhausted. An overview of the proposed model inference pipeline is illustrated in Figure 1.

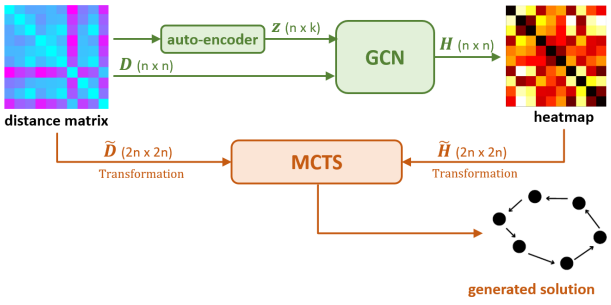


*Figure 1.* Model Inference Pipeline

### 3.2. GCN and Autoencoder

#### 3.2.1. Graph ConvNet

Our network builds upon the work (Joshi et al., 2019), which employs a Graph Convolutional Network (GCN) to directly output a probability heatmap matrix, where each entry represents the likelihood that a given edge is part of the optimal tour. Detailed architecture and implementation are provided in Appendix D.

#### 3.2.2. Autoencoder

In the TSP setting, node coordinates serve as effective input features, capturing spatial relationships directly. However, for the Asymmetric TSP (ATSP), such coordinates are typically unavailable. To obtain node embeddings that capture spatial structure purely from the distance matrix, we design an autoencoder-based representation learning module.

Let $D \in \mathbb{R}^{n \times n}$ denote the normalized distance matrix of an ATSP instance with $n$ nodes. For each node $i$, we extract its outgoing and incoming distance vectors, $D_{i,:}$ and $D_{:,i}$ respectively, and concatenate them with a learnable positional embedding $\text{PE}(i) \in \mathbb{R}^p$. The input to the encoder is thus:

$$x_i = [D_{i,:} \,\|\, D_{:,i} \,\|\, \text{PE}(i)] \in \mathbb{R}^{2n+p}, \tag{1}$$

where $\|$ denotes vector concatenation.

The encoder network maps this input to a latent embedding:

$$z_i = \text{Encoder}(x_i) \in \mathbb{R}^d, \tag{2}$$

where $d$ is the embedding dimension. A decoder then attempts to reconstruct the outgoing distance vector from this latent representation:

$$\hat{D}_{i,:} = \text{Decoder}(z_i) \in \mathbb{R}^n. \tag{3}$$

The model is trained to minimize the mean squared reconstruction error over all nodes:

$$\mathcal{L} = \frac{1}{n} \sum_{i=1}^{n} \left\| \hat{D}_{i,:} - D_{i,:} \right\|_2^2. \tag{4}$$

After training, we retain only the encoder and use the full $d$-dimensional embedding $z_i$ for each node. These learned embeddings serve as spatial representations that replace traditional coordinates and are used as input features for downstream models such as Graph Convolutional Networks (GCNs).[4]

[4]An alternative approach is to apply singular value decomposition (SVD) to the distance matrix. In our experiments, we found that SVD takes longer to generate node embeddings, while achieving similar final performance compared to the autoencoder. Therefore, we adopt the autoencoder-based approach in this work.

### 3.3. MCTS with Only 3-OPT

#### 3.3.1. Heatmap and Distance Matrix Transformation

To solve an ATSP instance, the original distance matrix is first passed through the autoencoder to generate node embeddings, which are then used by the GCN to produce an edge potential heatmap $H$. Before initiating the MCTS procedure, both the heatmap and the distance matrix are transformed. The details of this transformation are provided in Appendix C. After transformation, we obtain a expanded heatmap $\tilde{H}$ and a corresponding distance matrix suitable for use in the MCTS framework.

#### 3.3.2. Initial Solution Generation

To construct the initial solution, we adopt a stochastic procedure guided by the heatmap. Starting from a fixed node, the algorithm iteratively selects the next node based on a softmax distribution over unvisited candidates, where the logits correspond to the heatmap scores. Specifically, given the current node $c$, the probability of selecting an unvisited node $i$ is given by:

$$P(i \mid c) = \frac{\exp\left(\frac{\tilde{H}_{c,i}}{\tau}\right)}{\sum_{j \notin \text{visited}} \exp\left(\frac{\tilde{H}_{c,j}}{\tau}\right)}, \tag{5}$$

where $\tilde{H} \in \mathbb{R}^{2n \times 2n}$ is the expanded heatmap and $\tau > 0$ is a temperature parameter controlling the randomness of the selection. Lower values of $\tau$ produce greedier behavior, while higher values encourage more exploration.

#### 3.3.3. 3-OPT Local Search

We apply a 3-OPT local search to refine the initial solution by iteratively replacing three edges with a different set of three, aiming to reduce the overall tour cost.

Each 3-OPT move begins from a base node $a$, where we evaluate candidate connections $b'$. The candidate set is selected based on the following potential score:

$$Z_{a,b'} = \tilde{H}_{a,b'} + \alpha \cdot \frac{\ln(M+1)}{Q_{a,b'} + 1}, \tag{6}$$

where $\tilde{H}_{a,b'}$ denotes the heatmap score from the transformed probability matrix, which is updated dynamically throughout the simulation. $M$ is the total number of simulation steps, and $Q_{a,b'}$ is the number of times that edge $(a, b')$ has been assessed during 3-OPT. This formulation balances the learned prior with an exploration term, encouraging diverse search behavior.

We denote the successor of node $u$ in the tour as $u'$ (i.e., there exists an edge $u \to u'$). As illustrated in Figure 2,

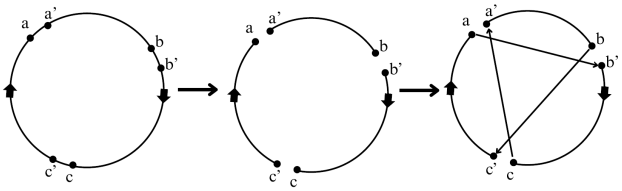


*Figure 2.* A 3-OPT Move Example

once a candidate pair $(a, b')$ is selected, the edges $(a, a')$ and $(b, b')$ are designated for removal.

To determine the third edge to cut, we simulate traversals starting from edge $(b, b')$ and explore the sequence of connected edges until we loop back to $(a, a')$. Among the visited candidates, the best third edge is selected based on the potential improvement in tour cost. After cutting three edges, the tour is divided into three subtours. Since our reconnection strategy always links the head of one subtour to the tail of another, the internal order of each subtour remains unchanged, and no edge reversals are required. Under this constraint, the change in cost of a 3-OPT move is computed as:

$$\Delta = (d_{a,a'} + d_{b,b'} + d_{c,c'}) - (d_{a,b'} + d_{b,c'} + d_{c,a'}), \quad (7)$$

where $(a, a')$, $(b, b')$, and $(c, c')$ are the original edges to be removed, and $(a, b')$, $(b, c')$, and $(c, a')$ are the new edges introduced by the move. An example of our 3-OPT move is illustrated in Figure 2. If a gainful move is found, we apply the exchange and update both the heatmap $\tilde{H}$ and the access frequency matrix $Q$ accordingly. Otherwise, we proceed to evaluate the next candidate $b'$ for the current base node until all candidates have been tested. This 3-OPT procedure is repeated for different base nodes until every node in the tour has been considered.

The **Initial Solution Generation** followed by **3-OPT Local Search** is repeated iteratively until the predefined time budget is exhausted. The complete procedure and detailed analysis are provided in Appendix E.

# 4. Experiments

## 4.1. Training

### 4.1.1. Training Data

We construct the training dataset using ATSP instances solved by the exact solver Concorde (Applegate et al., 2006). Details on the generation of ATSP instances can be found in Appendix F.

### 4.1.2. Training Procedure

During training, we adopt a supervised learning framework in which each ATSP instance is paired with its corresponding optimal solution, as illustrated in Figure 3. The GCN model is trained using the loss function detailed in Appendix D. The network architecture comprises four residual gated GCN layers, followed by two fully connected MLP layers, with a hidden dimension of 64 throughout. We train the model on 10,000 instances using a batch size of 64 for 100 epochs, leveraging an NVIDIA GeForce RTX 4060 Laptop GPU.

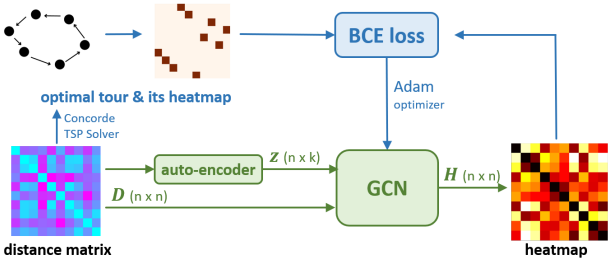


*Figure 3.* Supervised Training Pipeline

## 4.2. Test Results

### 4.2.1. 100 Node ATSP Test

We conduct tests using a 13th Gen Intel® Core™ i9-13900H CPU. To evaluate the performance of our model, we test it on 1000 ATSP instances with 100 nodes and compare its results against several baselines: an exact solver (Concorde) (Applegate et al., 2006), a classical heuristic algorithm (LKH-3) (Helsgaun, 2000), and a recent learning-based solver (UniCO) (Pan et al., 2025). The results are summarized in Table 2.

*Table 2.* Performance of our technique compared to non-learned baselines and state-of-the-art learning-based solver (UniCO) on 100-node ATSP instances.

| Method | Solving Time (s) | Optimality Gap (%) |
|---|---|---|
| Concorde | 31.00 | 0.00 |
| LKH-3 | 4.20 | 0.30 |
| UniCO | 0.67 | 3.55 |
| **Ours** | **0.60** | **1.71** |

Our method achieves a strong balance between speed and accuracy. While Concorde provides exact solutions, it requires significantly more computation time. Compared to LKH-3, our method is roughly seven times faster and achieves a competitive optimality gap. More notably, compared to

the latest learning-based solver UniCO, our model not only reduces the optimality gap from 3.55% to 1.71%, but also improves inference speed.[5] These results demonstrate that our GCN-guided MCTS framework is both computationally efficient and solution-effective for solving ATSP instances.

To further evaluate whether the heatmap generated by our neural network provides effective guidance for solution generation, we compare its performance against three alternative heatmaps. The first is a *random* heatmap, where all entries are assigned uniform probability, implying no learned information is incorporated. The second is the *SoftDist* heatmap, constructed from pairwise edge distances using a softmax function that favors shorter edges (Xia et al., 2024). The third is generated by the UTSP neural model (Min et al., 2023), which learns edge probabilities in an unsupervised manner. We evaluate all heatmaps using three inference strategies—greedy selection, sampling, and MCTS—on 100-node ATSP instances. As shown in Table 3, our learned heatmap consistently achieves the lowest optimality gap across all settings, demonstrating its strong guidance for solution generation.

Without applying MCTS, our heatmap still significantly outperforms the alternatives. These results indicate that neither SoftDist nor UTSP captures sufficient structural information for ATSP instances. In contrast, our model learns a more effective edge selection prior, enabling high-quality initial solutions even before refinement. When combined with MCTS, all methods benefit from local search improvements. This highlights the effectiveness of our search framework in producing near-optimal tours within a short computation time.

To evaluate the stability of our model's performance, we compute the optimality gap for each instance under varying time budgets and visualize the distribution using box plots in Figure 4. The figure illustrates how the optimality gap varies across three time budgets: 0.2s, 0.4s, and 0.6s per instance. As the time budget increases, the median optimality gap consistently decreases, indicating that the model effectively utilizes additional computation time to improve solution quality. Furthermore, the interquartile range (IQR) narrows with longer budgets, reflecting greater consistency and reduced variability across instances. These results demonstrate that our model delivers stable and robust performance, progressively refining solutions with minimal sensitivity to time constraints.

Additional experiments and analyses can be found in Appendix G.

[5]Since the original paper did not report the solving time for ATSP separately, we estimated it based on the ratio of solving times relative to the exact solver to obtain an approximate value.

*Table 3.* Performance comparison of different heatmaps under variant inference strategy on 100-node ATSP instances.

| Method | Inference | Optimality Gap (%) |
|---|---|---|
| Random | Greedy | 296.31 |
| UTSP | Greedy | 279.22 |
| SoftDist | Greedy | 113.69 |
| **Ours** | Greedy | **14.99** |
| Random | Sampling (x128) | 263.49 |
| UTSP | Sampling (x128) | 242.30 |
| SoftDist | Sampling (x128) | 85.81 |
| **Ours** | Sampling (x128) | **7.74** |
| Random | MCTS | 4.64 |
| UTSP | MCTS | 4.24 |
| SoftDist | MCTS | 3.42 |
| **Ours** | MCTS | **1.71** |

**Greedy** refers to selecting the next node with the highest probability directly from the heatmap. **Sampling (x128)** indicates that nodes are sampled according to the probability distribution in the heatmap, with 128 initial tours generated and the best one selected. **MCTS** refers to our proposed Monte Carlo Tree Search, with a searching time budget of 0.6 seconds per instance.

#### 4.2.2. 1000-Node ATSP Test

To assess the scalability of our model, we perform experiments on 1000-node ATSP networks. Specifically, we evaluate performance on 50 ATSP instances, each containing 1,000 nodes. To facilitate these tests, we adopt a divide-and-conquer paradigm through the GLOP framework (Ye et al., 2024). The results of our GLOP-augmented model, compared to the classical heuristic LKH-3 (Helsgaun, 2000), are presented in Table 4.

*Table 4.* Performance comparison between LKH-3 and our GLOP-augmented model on 1000-node ATSP instances

| Method | Avg. Solving Time (s) | Avg. Tour Distance |
|---|---|---|
| LKH-3 | 217.11 | 57.51 |
| Ours | 21.23 | 61.08 |

The results show that with significantly lower average solving time, our GLOP-augmented model achieves demonstrably superior efficiency on large-scale instances. Although the average tour distance is slightly higher, the trade-off is reasonable given the one order of magnitude reduction in computational cost, making our method well-suited for time-sensitive or resource-constrained scenarios.

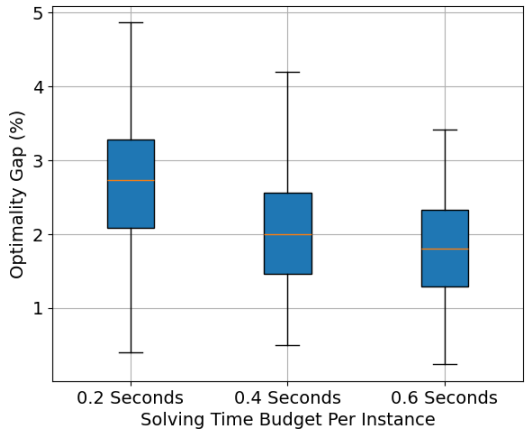


*Figure 4.* Distribution of optimality gaps for our method across instances under different time budgets. Each box plot represents the variability and robustness of the model's performance at a specific time setting.

## 5. Conclusion and Future Work

**Conclusion** In this paper, we evaluated the roles of heatmaps and Monte Carlo Tree Search (MCTS) in prior works. Building upon these insights, we proposed a new learning-based framework for solving the Asymmetric Traveling Salesman Problem (ATSP)—a complex NP-hard combinatorial optimization problem. Powered by learned priors and guided search, our method achieves strong performance and high efficiency compared to existing general neural solvers. It also establishes a useful benchmark for future research, paving the way for continued advancements in this domain.

**Limitations and Future Work** While our model advances beyond the classical TSP to tackle the more complex ATSP, it is not yet equipped to handle richer constraints such as stochastic or dynamic conditions. Moreover, this work remains limited to TSP-based problems. In future work, we aim to extend our approach to a broader class of combinatorial optimization problems, including the Capacitated Vehicle Routing Problem (CVRP), VRP with Time Windows, and beyond.

## References


Applegate, D., Bixby, R., Chvatal, V., and Cook, W. Concorde tsp solver, 2006.

Applegate, D. L. *The traveling salesman problem: a computational study*, volume 17. Princeton university press, 2006.

Bello, I., Pham, H., Le, Q. V., Norouzi, M., and Bengio, S. Neural combinatorial optimization with reinforcement learning. *arXiv preprint arXiv:1611.09940*, 2016.

Dantzig, G., Fulkerson, R., and Johnson, S. Solution of a large-scale traveling-salesman problem. *Journal of the operations research society of America*, 2(4):393–410, 1954.

Drakulic, D., Michel, S., Mai, F., Sors, A., and Andreoli, J.-M. Bq-nco: Bisimulation quotienting for efficient neural combinatorial optimization. *Advances in Neural Information Processing Systems*, 36:77416–77429, 2023.

Fu, Z.-H., Qiu, K.-B., and Zha, H. 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.

Gutin, G. and Punnen, A. P. *The traveling salesman problem and its variations*, volume 12. Springer Science & Business Media, 2006.

Helsgaun, K. An effective implementation of the lin–kernighan traveling salesman heuristic. *European journal of operational research*, 126(1):106–130, 2000.

Helsgaun, K. General k-opt submoves for the lin–kernighan tsp heuristic. *Mathematical Programming Computation*, 1:119–163, 2009.

Holland, J. H. *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*. MIT press, 1992.

Jonker, R. and Volgenant, T. Transforming asymmetric into symmetric traveling salesman problems. *Operations Research Letters*, 2(4):161–163, 1983.

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

Khalil, E., Dai, H., Zhang, Y., Dilkina, B., and Song, L. Learning combinatorial optimization algorithms over graphs. *Advances in neural information processing systems*, 30, 2017.

Kirkpatrick, S., Gelatt Jr, C. D., and Vecchi, M. P. Optimization by simulated annealing. *science*, 220(4598): 671–680, 1983.

Kool, W., Van Hoof, H., and Welling, M. Attention, learn to solve routing problems! *arXiv preprint arXiv:1803.08475*, 2018.

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

Laporte, G. The traveling salesman problem: An overview of exact and approximate algorithms. *European Journal of Operational Research*, 59(2):231–247, 1992.

Lawler, E. L. The traveling salesman problem: a guided tour of combinatorial optimization. *Wiley-Interscience Series in Discrete Mathematics*, 1985.

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

Miller, C. E., Tucker, A. W., and Zemlin, R. A. Integer programming formulation of traveling salesman problems. *Journal of the ACM (JACM)*, 7(4):326–329, 1960.

Min, Y., Bai, Y., and Gomes, C. P. Unsupervised learning for solving the travelling salesman problem. *Advances in Neural Information Processing Systems*, 36:47264–47278, 2023.

Pan, W., Xiong, H., Ma, J., Zhao, W., Li, Y., and Yan, J. UniCO: On unified combinatorial optimization via problem reduction to matrix-encoded general TSP. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=yEwakMNIex.

Pop, P. C., Cosma, O., Sabo, C., and Sitar, C. P. A comprehensive survey on the generalized traveling salesman problem. *European Journal of Operational Research*, 314(3):819–835, 2024.

Qiu, R., Sun, Z., and Yang, Y. Dimes: A differentiable meta solver for combinatorial optimization problems. *Advances in Neural Information Processing Systems*, 35: 25531–25546, 2022.

Sun, Z. and Yang, Y. Difusco: Graph-based diffusion solvers for combinatorial optimization. *Advances in neural information processing systems*, 36:3706–3731, 2023.

Vinyals, O., Fortunato, M., and Jaitly, N. Pointer networks. *Advances in neural information processing systems*, 28, 2015.

Xia, Y., Yang, X., Liu, Z., Liu, Z., Song, L., and Bian, J. Position: Rethinking post-hoc search-based neural approaches for solving large-scale traveling salesman problems. *arXiv preprint arXiv:2406.03503*, 2024.

Ye, H., Wang, J., Liang, H., Cao, Z., Li, Y., and Li, F. 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.

## A. Visual Comparison of Heatmap-Guided Tours

To illustrate the impact of different heatmap priors on solution quality, we visualize the tours generated for a TSP instance under the guidance of two distinct heatmaps: (1) UTSP Heatmap—a learned heatmap produced by the trained model (Min et al., 2023), and (2) SoftDist Heatmap—a naïve heuristic-based heatmap that assigns edge probabilities using a softmax function over inverse distances, thereby favoring shorter edges (Xia et al., 2024). Tours are constructed using either greedy inference or Monte Carlo Tree Search (MCTS) guided by these priors.

We begin with a 100-node TSP instance and visualize the tours generated under the guidance of the UTSP heatmap, as illustrated in Figure 5. The results show that the UTSP heatmap alone fails to capture critical structural patterns of the optimal route, resulting in poor performance when a greedy inference strategy is applied. In contrast, when the UTSP heatmap is combined with MCTS, the generated tour closely matches the optimal one in cost, indicating that the performance improvement is primarily driven by the MCTS procedure rather than the heatmap.

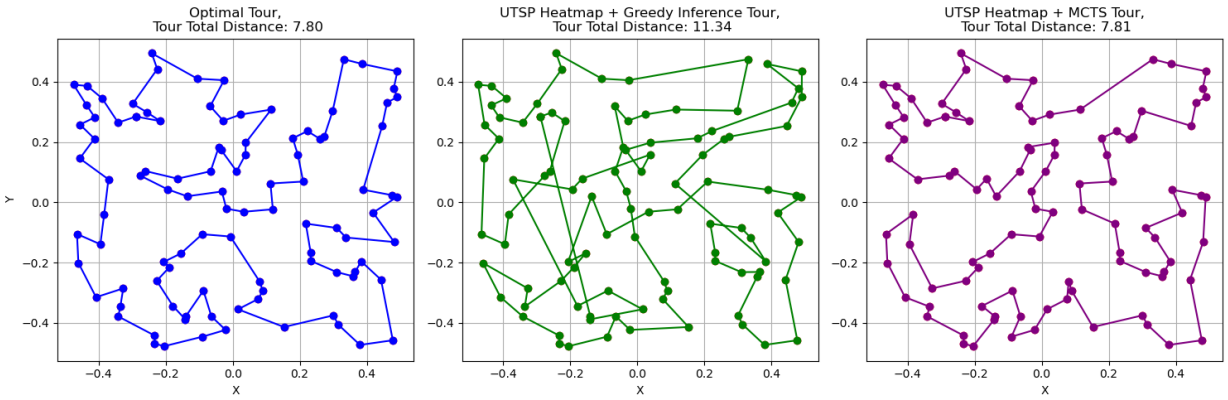


*Figure 5.* Tours generated for a 100-node TSP instance. The first tour is the optimal solution obtained from an exact solver, while the latter two are generated using the UTSP heatmap via greedy inference and MCTS.

We then apply the same 100-node TSP instance to visualize the tours generated under the guidance of the SoftDist heatmap, as shown in Figure 6. Compared to the UTSP heatmap, the SoftDist heatmap alone captures more structural information about the optimal solution, resulting in a lower-cost tour under greedy inference. When combined with MCTS, the resulting tour again closely approximates the optimal cost, further emphasizing the critical role of MCTS in achieving high-quality solutions.

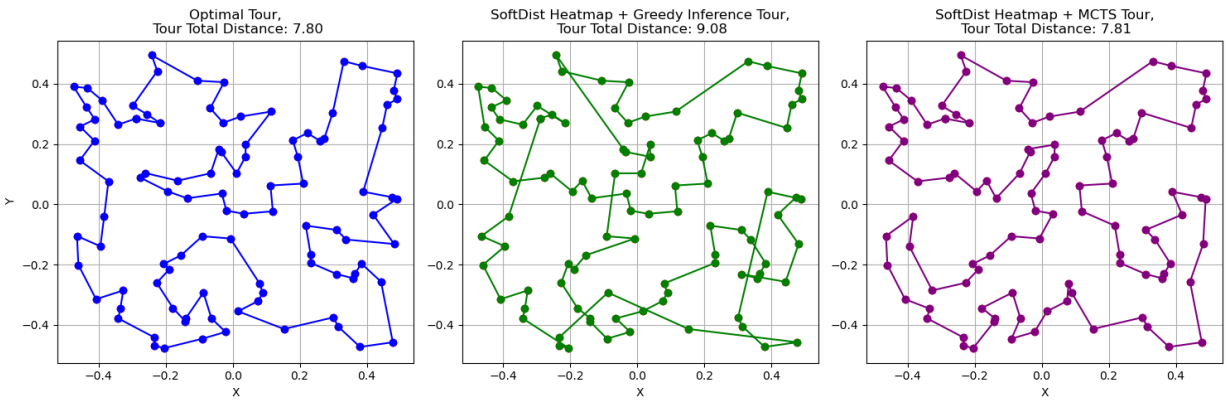


*Figure 6.* Tours generated for a 100-node TSP instance. The first tour is the optimal solution obtained from an exact solver, while the latter two are generated using the SoftDist heatmap via greedy inference and MCTS.

For reference, we also visualize the tours for two additional 100-node TSP instances. In both cases, the results consistently support the conclusions discussed above.

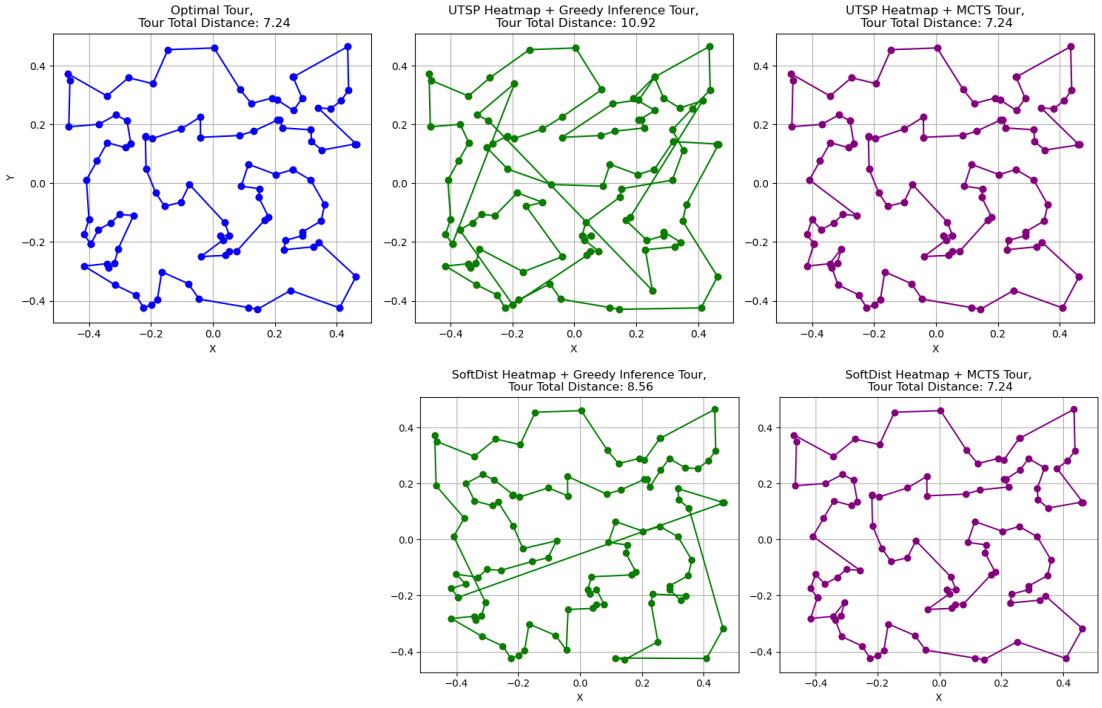


*Figure 7.* Tour visualizations for a 100-node TSP instance using UTSP and SoftDist heatmaps.

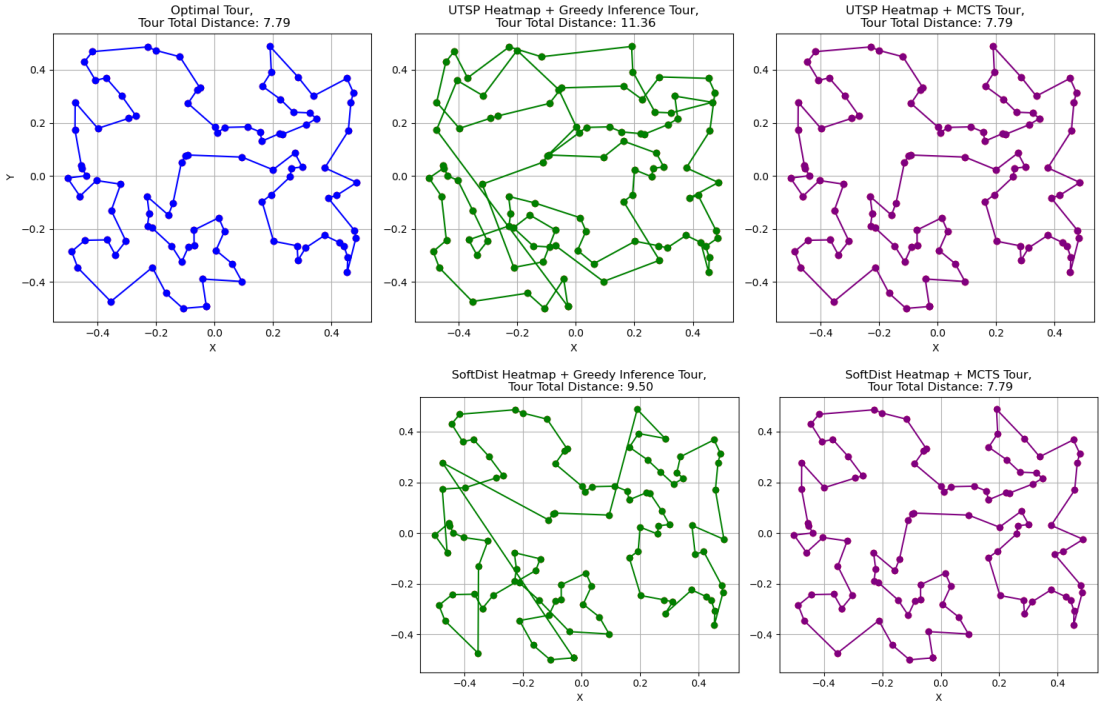


*Figure 8.* Tour visualizations for a second 100-node TSP instance using UTSP and SoftDist heatmaps.

## B. Previous Monte Carlo Tree Search

In this part, we briefly introduce the Monte Carlo Tree Search (MCTS) method from (Fu et al., 2021). MCTS is a $k$-OPT process guided by the heatmap, which iteratively refines a complete TSP solution $\pi$ by alternating edge deletions and additions. The selection of edges during $k$-OPT is influenced by a weight matrix $W$ and an access matrix $Q$, both of which are dynamically updated based on $k$-OPT outcomes. Here, $W_{i,j}$ scores the suitability of edge $(i, j)$ in the solution, while $Q_{i,j}$ records the number of times edge $(i, j)$ is selected.

**Initial Tour.** An initial tour $\pi$ is constructed randomly, with the constraint that each node is visited exactly once. Pure randomness is used here—rather than a heatmap-based initialization—to avoid deterministic patterns and encourage diverse starting solutions.

**2-OPT Search.** A 2-OPT local search is applied to improve the initial solution. In this phase, edge exchanges are guided purely by the distance matrix; the heatmap is not used. This helps refine the initial tour based solely on problem geometry.

**$k$-OPT Search.** In this phase, the heatmap $H$ is used to initialize the weight matrix $W$ via $W_{i,j} = 100 \times H_{i,j}$. The access matrix $Q$ is initialized with all elements set to zero. An edge potential matrix $Z$ guides the $k$-OPT process by balancing exploitation and exploration. The edge potential $Z_{i,j}$ is defined as:

$$Z_{i,j} = \frac{W_{i,j}}{\Omega_i} + \alpha \sqrt{\frac{\ln(M+1)}{Q_{i,j}+1}}, \tag{8}$$

where $\Omega_i$ is the average weight of edges connected to vertex $i$:

$$\Omega_i = \frac{\sum_{j \neq i} W_{i,j}}{n-1}, \tag{9}$$

$\alpha$ controls the exploration-exploitation trade-off, and $M$ is the total number of actions sampled so far.

Each $k$-OPT action is represented as a vertex decision sequence $(a_1, b_1, a_2, b_2, \ldots, a_k, b_k, a_{k+1})$ with $a_{k+1} = a_1$. This sequence involves removing $k$ edges $(a_i, b_i)$ and inserting $k$ new edges $(b_i, a_{i+1})$ for $1 \leq i \leq k$. Given $b_i$, the next vertex $a_{i+1}$ is sampled according to the edge potential $Z_{i,j}$. The tour $\pi$ is then updated to $\pi^{\text{new}}$, the associated metrics $M$ and $Q$ are updated accordingly.

**Backpropagation.** If a better solution $\pi^{\text{new}}$ is found such that $c(\pi^{\text{new}}) < c(\pi)$, the weights of the newly added edges from the $k$-OPT action are reinforced using the following rule:

$$W_{i,j} \leftarrow W_{i,j} + \beta \left[ \exp\left( \frac{c(\pi) - c(\pi^{\text{new}})}{c(\pi^{\text{new}})} \right) - 1 \right], \tag{10}$$

where $\beta$ is the update rate.

All the above steps are executed sequentially and repeated multiple times until the time budget is exhausted.

## C. ATSP Transformation

We transform the Asymmetric TSP (ATSP) instance into an equivalent Symmetric TSP (STSP) instance using the Jonker-Volgenant (1983) transformation (Jonker & Volgenant, 1983). This transformation is applied to both the distance matrix and the model-generated heatmap.

**Distance Matrix Transformation** Let ATSP($D$) denote an asymmetric traveling salesman problem (ATSP) defined on a distance matrix $D = (d_{ij})$, where $i, j \in N$, and $N = \{1, 2, \ldots, n\}$ is the set of nodes. The asymmetry is characterized by $d_{ij} \neq d_{ji}$ for $i \neq j$.

Let the modified matrix $\bar{D}$ be identical to $D$ except that

$$\bar{d_{ii}} = -M \quad (\text{for all } i \in N), \tag{11}$$

where $M$ is a very large constant.

Let $U = (u_{ij})$ be an $n \times n$ matrix with $u_{ij} = \infty$ for all $i, j \in N$. We propose transforming the asymmetric ATSP($D$) into a symmetric TSP defined on the distance matrix

$$\tilde{D} = \begin{bmatrix} U & \bar{D} \\ \bar{D}^\top & U \end{bmatrix}, \tag{12}$$

which we denote as TSP($\hat{D}$). The node set of TSP($\hat{D}$) is $\{1, 2, \ldots, n, n+1, \ldots, 2n\}$.

The optimal solutions of TSP($\hat{D}$) belong to a class of solutions with finite cost that contain exactly $n$ edges of weight $-M$. Due to the symmetry of $\hat{D}$, these solutions occur in pairs. It is easy to verify that each solution in this class takes the form:

$$i_1 \to (i_1 + n) \to i_2 \to (i_2 + n) \to \cdots \to i_n \to (i_n + n) \to i_1,$$

where $i_k \in N$ for $k = 1, 2, \ldots, n$.

There is a one-to-one correspondence between the set of such TSP($\tilde{D}$) solutions and the solutions of the original ATSP($D$). To recover the ATSP solution from TSP($\tilde{D}$), we can simply remove all nodes with indices greater than $n$, and add $n \cdot M$ to the total cost to offset the artificially inserted edges with cost $-M$.

**Heatmap Transformation** Based on the structural property of the optimal solution in the transformed space, we expand the asymmetric heatmap $H \in [0, 1]^{n \times n}$ into a expanded heatmap $\tilde{H} \in [0, 1]^{2n \times 2n}$ as follows:

$$\tilde{H} = \begin{bmatrix} 0 & U \\ H & 0 \end{bmatrix}, \quad \text{where} \quad U_{i,j} = \begin{cases} 1 & \text{if } j = i + n \\ 0 & \text{otherwise} \end{cases} \tag{13}$$

The top-right block $U$ encodes a strong prior, encouraging each original node $i$ to be followed by its corresponding dummy node $i + n$, thereby reinforcing the structural pattern commonly observed in optimal solutions of the expanded symmetric TSP. The bottom-left block replicates the original heatmap $H$, guiding transitions from dummy nodes back to real nodes and preserving the learned edge preferences of the ATSP model. Note that although the transformed distance matrix $\tilde{D}$ is symmetric, the expanded heatmap $\tilde{H}$ remains asymmetric.

**Tour Conversion** The resulting symmetric instance is solved as a TSP over $2n$ nodes. The output tour alternates between original and dummy nodes in the pattern $i \to i + n \to \cdots$. The final ATSP tour is recovered by removing the dummy nodes from this alternating sequence.

## D. Graph Convolutional Network (GCN)

**Input Layer** Each node is represented by a $k$-dimensional feature vector $\{x_i\}$ from the autoencoder, which is projected into an $h$-dimensional node embedding as follows:

$$\alpha_i = A_1 x_i + b_1, \tag{14}$$

where $A_1 \in \mathbb{R}^{h\times k}$ and $b_1 \in \mathbb{R}^h$.

Similarly, the edge distance $d_{ij}$ is embedded into an $h$-dimensional edge feature. The resulting edge embedding $\beta_{ij}$ is computed as:

$$\beta_{ij} = A_3 \cdot \text{ReLU}(A_2 d_{ij}), \tag{15}$$

where $A_2 \in \mathbb{R}^{h\times 1}$ and $A_3 \in \mathbb{R}^{h\times h}$.

**Graph Convolution Layer** Let $x_i^\ell$ and $e_{ij}^\ell$ denote the node and edge embeddings at layer $\ell$, corresponding to node $i$ and edge $(i,j)$, respectively. The embeddings at the next layer are updated as follows:

$$x_i^{\ell+1} = x_i^\ell + \text{ReLU}\left(\text{BN}\left(W_1^\ell x_i^\ell + \sum_{j\sim i} \eta_{ij}^\ell \odot W_2^\ell x_j^\ell\right)\right), \tag{16}$$

$$\eta_{ij}^\ell = \frac{\sigma(e_{ij}^\ell)}{\sum_{j'\sim i}\sigma(e_{ij'}^\ell) + \varepsilon}, \tag{17}$$

$$e_{ij}^{\ell+1} = e_{ij}^\ell + \text{ReLU}\left(\text{BN}\left(W_3^\ell e_{ij}^\ell + W_4^\ell x_i^\ell + W_5^\ell x_j^\ell\right)\right), \tag{18}$$

where $W_n^\ell \in \mathbb{R}^{h\times h}$ are learnable weight matrices, $\sigma(\cdot)$ denotes the sigmoid function, $\varepsilon$ is a small constant for numerical stability, ReLU is the rectified linear unit, and BN denotes batch normalization. At the input layer, we initialize with $x_i^0 = \alpha_i$ and $e_{ij}^0 = \beta_{ij}$.

**MLP Classifier** The edge embedding $e_{ij}^L$ from the final layer is used to estimate the likelihood that edge $(i,j)$ is part of the optimal tour. First, each embedding is passed through a *Multi-Layer Perceptron* (MLP) to produce a scalar score. Then, a softmax operation is applied over all outgoing edges from node $i$ to obtain a normalized probability distribution:

$$p_{ij} = \frac{\exp(\text{MLP}(e_{ij}^L))}{\sum_{j'\sim i}\exp(\text{MLP}(e_{ij'}^L))}. \tag{19}$$

This ensures that $\sum_{j'\sim i} p_{ij'} = 1$ for each node $i$, and $p_{ij} \in (0,1)$ represents the probability of selecting edge $(i,j)$.

**Loss function** Given the ground-truth TSP tour permutation $\pi$, we convert it into a binary adjacency matrix $Y \in \{0,1\}^{n\times n}$, where $y_{ij} = 1$ if edge $(i,j)$ belongs to the tour, and 0 otherwise. The model predicts a probability matrix $\hat{Y} \in (0,1)^{n\times n}$, where each element $\hat{y}_{ij}$ represents the predicted likelihood that edge $(i,j)$ is part of the optimal tour.

To handle the severe class imbalance (few tour edges vs. many non-tour edges), we minimize a weighted binary cross-entropy loss:

$$\mathcal{L}_{\text{BCE}} = -\frac{1}{n^2}\sum_{i=1}^{n}\sum_{j=1}^{n}\Big[\, w_1 \cdot y_{ij} \cdot \log(\hat{y}_{ij} + \varepsilon) + w_0 \cdot (1 - y_{ij}) \cdot \log(1 - \hat{y}_{ij} + \varepsilon)\Big] \tag{20}$$

where $\varepsilon$ is a small constant to ensure numerical stability.

According to the previous work (Joshi et al., 2019), the class weights $w_0$ and $w_1$ are computed to balance the positive and negative classes:

$$w_0 = \frac{n^2}{(n^2 - 2n)\cdot c}, \quad w_1 = \frac{n^2}{(2n)\cdot c}, \quad c = 2, \tag{21}$$

where $c$ is the number of classes. This formulation ensures that the loss is not dominated by the negative class and encourages the model to correctly identify edges belonging to the TSP tour.

## E. Monte Carlo Tree Search with only 3-OPT Optimization

The output of the Graph Convolutional Network is a heatmap $H \in \mathbb{R}^{n \times n}$. Following the transformation described in Appendix C, we construct an expanded symmetric distance matrix $\tilde{D}$ and a corresponding expanded heatmap $\tilde{H} \in \mathbb{R}^{2n \times 2n}$, where each entry $\tilde{H}_{i,j}$ represents the learned likelihood of selecting edge $(i, j)$ as part of the optimal tour. The expanded heatmap $\tilde{H}$ and distance matrix $\tilde{D}$ are then used as input to our MCTS framework.

### E.1. Heatmap-Guided Initialization

To generate an initial solution, we employ a softmax-based stochastic policy that balances exploration and exploitation, with selection probabilities guided by the values in $\tilde{H}$. Starting from a predefined node $c$, we iteratively construct a tour by sampling the next unvisited node $i$ according to the following softmax distribution:

$$P(i \mid c) = \frac{\exp\left(\frac{\tilde{H}_{c,i}}{\tau}\right)}{\sum_{j \notin \text{visited}} \exp\left(\frac{\tilde{H}_{c,j}}{\tau}\right)}, \tag{22}$$

where $\tau > 0$ is the softmax temperature controlling the randomness of the selection process. In our implementation, we set $\tau = 0.01$, which biases the initialization procedure strongly toward high-probability edges suggested by the heatmap.

After initialization, the resulting solution consistently takes the form:

$$i_1 \to (i_1 + n) \to i_2 \to (i_2 + n) \to \cdots \to i_n \to (i_n + n) \to i_1,$$

where $i_k \in N$ for $k = 1, 2, \ldots, n$. Note that all edges connecting each original node $i_k$ to its corresponding dummy node $(i_k + n)$ have artificially assigned negative costs and are therefore guaranteed to appear in any optimal solution of the transformed instance. As a result, in subsequent local search operations, we freeze these edges and restrict modifications to the remaining edges in the tour.

### E.2. 2-OPT Analysis

Before introducing the details of the 3-OPT procedure, we first explain why 2-OPT is not applicable in our setting. Consider the structure of the solution immediately after initialization. Suppose we attempt a 2-OPT move by removing two edges: the first from $(i_a + n)$ to $i_{a+1}$, and the second from $(i_b + n)$ to $i_{b+1}$. Note that $(i_a + n)$ and $(i_b + n)$ represent the dummy nodes corresponding to $i_a$ and $i_b$, respectively.

To reconnect the tour, we must link $(i_a + n)$ to another node. However, the only candidates here are $(i_b + n)$ and $i_{b+1}$. Connecting $(i_a + n)$ to $(i_b + n)$ is invalid because all edges between dummy nodes have infinite cost. On the other hand, connecting $(i_a + n)$ to $i_{b+1}$ would create two disconnected subtours, violating the tour constraint.

Therefore, no valid reconnection is possible under the 2-OPT scheme, making it ineffective for our expanded symmetric TSP formulation. This limitation motivates the use of 3-OPT, which provides greater flexibility in restructuring the tour. An illustrative example of a 2-OPT failure is shown in Figure 9.

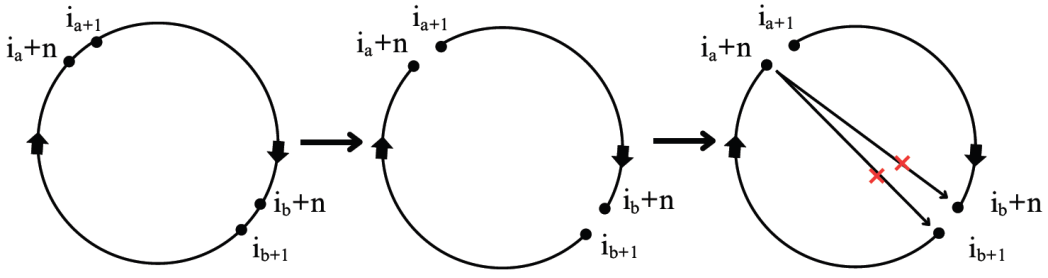


*Figure 9.* A 2-OPT Example

### E.3. 3-OPT Local Search

Following the initialization phase, we apply a 3-OPT local search procedure to refine the solution. The 3-OPT local search improves a tour by removing three edges and reconnecting the resulting segments in a different order to reduce the overall tour cost.

We begin by selecting a node $(i_a + n)$ from the current tour and evaluate potential reconnections with its candidate nodes. To guide this process, we define an edge potential $Z_{i,j}$ as:

$$Z_{i,j} = \tilde{H}_{i,j} + \alpha \frac{\ln(M+1)}{Q_{i,j}+1}, \tag{23}$$

where $\tilde{H}_{i,j}$ is the heatmap score from the expanded probability matrix $\tilde{H}$, $M$ is the total number of initial solution generations, and $Q_{i,j}$ denotes the number of times the edge $(i, j)$ has been selected, initialized to zero for all entries. Excluding the node already connected to $i$, we select the top $k$ nodes with the highest potentials as candidate reconnections (with $k = 10$ in our implementation). These candidates are then used to explore possible 3-OPT moves that preserve feasibility and improve the solution.

Next, we remove the current edge from node $(i_a + n)$ to $i_{a+1}$, and instead connect $(i_a + n)$ to $i_{b+1}$, a node selected from the candidate set. Since $i_{b+1}$ was originally connected from $(i_b + n)$, we must also remove the edge $(i_b + n) \to i_{b+1}$ to maintain tour consistency. At this stage—referred to as *Step 1* in the following discussion—two edges have been removed and one new edge has been added. This partial 3-OPT move is illustrated in Figure 10.

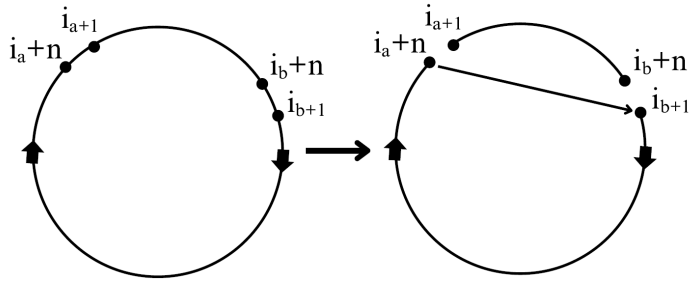


*Figure 10.* 3-OPT Move Step 1

After completing the first step, we proceed to identify a third edge for removal to complete the 3-OPT move. Starting from node $i_{b+1}$, we traverse the current tour until reaching $i_a + n$, ensuring the tour structure remains valid throughout. Along this segment, we consider removing the edge $(i_c + n) \to i_{c+1}$ for each candidate node $i_c$.

To evaluate the quality of each third-edge candidate, we perform a quick simulation to estimate the improvement. The improvement, denoted by $\Delta$, is calculated as follows:

$$\Delta = \left(d_{i_a+n,i_{a+1}} + d_{i_b+n,i_{b+1}} + d_{i_c+n,i_{c+1}}\right) - \left(d_{i_a+n,i_{b+1}} + d_{i_b+n,i_{c+1}} + d_{i_c+n,i_{a+1}}\right), \tag{24}$$

where $d_{u,v}$ denotes the distance from node $u$ to node $v$. A higher value of $\Delta$ indicates a greater potential improvement in the tour length resulting from the proposed 3-OPT move. Among all candidates, we select the edge with the highest $\Delta$ (i.e., the most beneficial one), denoted as $\Delta_{\text{best}}$.

If no improvement is found, i.e., $\Delta_{\text{best}} \leq 0$, we return to *Step 1* and select a different node from the candidate set to serve as $i_{b+1}$, repeating the process until all candidates have been explored.

If $\Delta_{\text{best}} > 0$, the move is considered beneficial and is applied to the current tour. The resulting edge configuration after the 3-OPT move is illustrated in Figure 11.

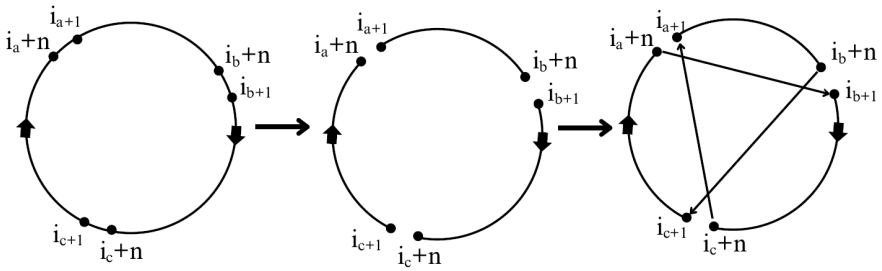


*Figure 11.* An Effective 3-OPT Move

Following a successful move, the heatmap $\tilde{H}$ is updated to reinforce the selected edges. Specifically, for each newly added edge $(i, j)$, we update its value according to the rule:

$$\tilde{H}_{i,j} \leftarrow \tilde{H}_{i,j} + \beta \cdot \frac{\Delta_{\text{3-OPT}}}{c(\pi^{\text{new}})}, \tag{25}$$

where $\beta$ is the update rate and $c(\pi^{\text{new}})$ denotes the cost of the new solution $\pi^{\text{new}}$.

In addition to updating the heatmap, we also update the access matrix $Q$, where $Q_{i,j}$ tracks the number of times edge $(i, j)$ has been selected. For all edges currently in the tour, we increment their corresponding entries in $Q$ by 1:

$$Q_{i,j} \leftarrow Q_{i,j} + 1 \quad \text{for all } (i, j) \in \pi^{\text{new}}. \tag{26}$$

The 3-OPT local search procedure is repeated by selecting different starting nodes $(i_a + n)$ and evaluating 3-OPT moves for various candidates in ***Step 1***, until all starting nodes in the tour have been considered. Once this process is complete, the heatmap-guided initialization is invoked again using the updated heatmap $\tilde{H}$ to generate a new starting solution, which is subsequently refined through another round of 3-OPT local search. This alternating cycle of initialization and local refinement continues iteratively until the predefined time budget is exhausted.

## F. Instances Generation

During training, we generate ATSP instances by randomly sampling from a simple and generic distribution. The procedure for constructing these training instances is outlined below:

1. **Sampling coordinates.** Sample node coordinates $\mathbf{x}_i$ independently and uniformly in the unit square, i.e., $x_{i1}, x_{i2} \sim U(0,1)$ independently. The Euclidean distance between nodes $i$ and $j$ is computed as:

$$\overline{d}_{ij} = \|\mathbf{x}_i - \mathbf{x}_j\|_2. \tag{27}$$

2. **Constructing the relative distance matrix.** We define a relative distance matrix $\tilde{\Gamma} \in \mathbb{R}^{n\times n}$ and set the final distance matrix as $\Gamma_{ij} = \tilde{\Gamma}_{ij} \cdot \overline{d}_{ij}$. Although $\tilde{\Gamma}$ is generally asymmetric, we begin by constructing a symmetric base and then inject controlled asymmetry. This approach reflects the intuition that, while $\Gamma_{ij} \neq \Gamma_{ji}$ in the ATSP, the two values are still highly correlated. The details are provided below:

   (a) *Symmetric component.* Generate a symmetric matrix $\tilde{\Gamma}^{\text{sym}}$ as:

$$\tilde{\Gamma}^{\text{sym}} = \exp\{Y\}, \quad \text{where} \quad Y = \frac{1}{\sqrt{2}}\left(\tilde{Y} + \tilde{Y}^\top\right), \tag{28}$$

   with element-wise exponentiation: $(\tilde{\Gamma}^{\text{sym}})_{ij} = \exp(Y_{ij})$. The scaling factor $\frac{1}{\sqrt{2}}$ ensures the variance:

$$\text{Var}\left(\frac{1}{\sqrt{2}}(\tilde{Y}_{ij} + \tilde{Y}_{ji})\right) = 2(\bar{s}^{\text{sym}})^2. \tag{29}$$

   Here, $\tilde{Y}_{ij} \sim \mathcal{N}(0, (\bar{s}^{\text{sym}})^2)$ i.i.d., leading to a log-normal distribution over $Y_{ij}$. This design induces symmetry in log-space:

$$\mathbb{P}\left[\frac{1}{r} < Y_{ij} < \frac{1}{t}\right] = \mathbb{P}\left[t < Y_{ij} < r\right], \quad \forall r > t > 1. \tag{30}$$

   In our implementation, we use $\bar{s}^{\text{sym}} = 0.5$.

   (b) *Anti-symmetric component.* To introduce asymmetry, we sample an anti-symmetric matrix:

$$\tilde{\Gamma}^{\text{asym}} = \frac{1}{\sqrt{2}}\left(Z - Z^\top\right), \tag{31}$$

   where $Z_{ij} \sim \mathcal{N}(0, (\bar{s}^{\text{asym}})^2)$ i.i.d., and $\bar{s}^{\text{asym}} = 0.05$.

   Finally, the two components are combined to form the relative distance matrix:

$$\tilde{\Gamma} = \tilde{\Gamma}^{\text{sym}} \odot \left(1 + \tilde{\Gamma}^{\text{asym}}\right), \tag{32}$$

   where $\odot$ denotes element-wise multiplication. The resulting ATSP distance from node $i$ to node $j$ is given by:

$$\Gamma_{ij} = \tilde{\Gamma}_{ij} \cdot \overline{d}_{ij}. \tag{33}$$

# G. Experiemnt Details

## G.1. Autoencoder Training

**Training Hyperparameters** The autoencoder is trained on a dataset of normalized ATSP distance matrices using the Adam optimizer with an initial learning rate of $1 \times 10^{-3}$. Training is conducted over 100 epochs with a batch size of 32. A StepLR scheduler with a decay factor of 0.2 is applied every 10 epochs to progressively reduce the learning rate. The model incorporates a learnable positional embedding of dimension 8 for each node, which is concatenated with both the corresponding row and column of the distance matrix before encoding. The encoder consists of two fully connected layers with ReLU activation, projecting the input into a 16-dimensional latent space. The decoder mirrors this structure and reconstructs the original distance row from the latent embedding.

**Training Curves** Figure 12 shows the training and validation loss curves of the autoencoder over 100 epochs. The model converges rapidly within the first 10 epochs, with both training and validation losses dropping significantly from their initial values. After this initial phase, the loss plateaus and stabilizes around a mean squared error of approximately 0.035, indicating that the model has effectively learned to reconstruct the distance rows from the latent embeddings. The close alignment between training and validation curves suggests that the model generalizes well to unseen instances and does not suffer from overfitting.

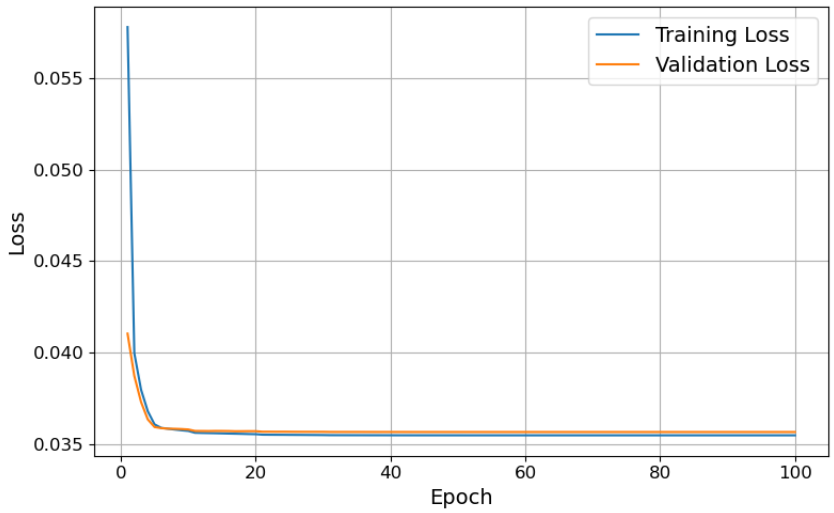


*Figure 12.* Autoencoder Training Loss & Validation Loss Curve

## G.2. GCN Training

**Training Hyperparameters** The Graph Convolutional Network (GCN) is trained using a weighted binary cross-entropy loss, with edge supervision derived from optimal ATSP tours. The model takes as input both the normalized distance matrix and node embeddings generated by the pretrained autoencoder. Training is conducted for up to 100 epochs with a batch size of 64, using the Adam optimizer and an initial learning rate of $1 \times 10^{-3}$. The network architecture comprises 4 GCN layers, each with 64 hidden units, followed by a two-layer multilayer perceptron (MLP) for edge prediction.

**Training Curves** As shown in Figure 13, both the training and validation losses decrease steadily during the initial epochs, reflecting effective learning and convergence of the model. After approximately 40 epochs, the loss values begin to plateau, indicating that the model has reached a stable state. Notably, the training and validation losses remain closely aligned throughout the training process, suggesting good generalization performance without signs of overfitting. By epoch 100, both losses converge to around 0.075, demonstrating that the model achieves consistent performance on both the training and validation sets.

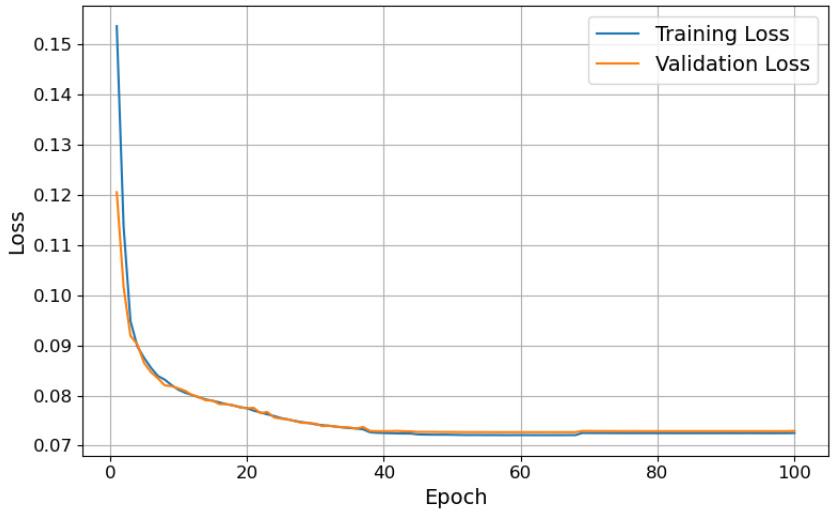


*Figure 13.* GCN Training Loss & Validation Loss Curve

### G.3. Test

**Test Hyperparameters** During testing, we employ a heatmap-guided MCTS framework followed by a 3-OPT local search. In the initialization phase, the next node is selected using a softmax-based stochastic policy with temperature $\tau = 0.01$, which promotes exploitation of high-probability edges suggested by the heatmap. During the 3-OPT refinement, each node evaluates up to 10 candidate reconnections based on an edge potential function, defined in Equation 23, with $\alpha = 0.1$. After a successful 3-OPT move, the heatmap is updated according to Equation 25, using a parameter $\beta = 1.0$.

**Time Scaling Curves** To evaluate the trade-off between solution quality and computational effort, we analyze the optimality gap as a function of search time. The model is tested under varying time budgets on the same set of 50 ATSP instances with 100 nodes, and the average optimality gap is computed for each setting. As shown in Figure 14, the optimality gap decreases rapidly during the initial phase as search time increases. Within the first 2 seconds, the gap drops from over 3% to approximately 1.5%, indicating that most of the improvement occurs early. Beyond this point, the rate of improvement slows, and the curve begins to plateau. After 6 seconds, additional search yields only marginal gains, with the optimality gap stabilizing around 1.12%. In this work, we typically allocate 0.6 seconds of search time per 100-node ATSP instance to balance efficiency and performance.

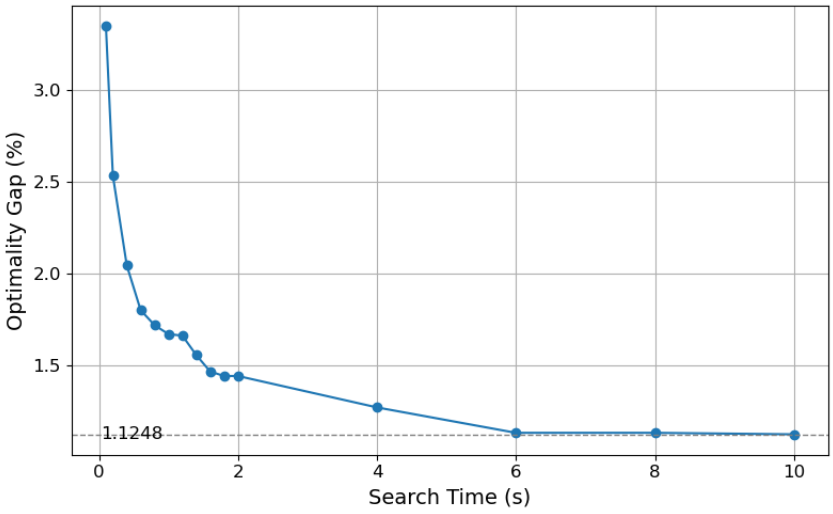


*Figure 14.* Optimality Gap vs Search Time Curve