Artificial Intelligence Review (2025) 58:267
https://doi.org/10.1007/s10462-025-11267-x

Solving the traveling salesman problem with machine learning: a review of recent advances and challenges

Entesar Alanzi[1,2] · **Mohamed El Bachir Menai**[1]

Accepted: 12 May 2025 / Published online: 4 June 2025

Abstract
The Traveling Salesman Problem (TSP) is a classic NP-hard problem that is central to various applications in logistics, robotics, and scheduling. This paper presents a comprehensive review of Machine Learning (ML) approaches for solving the TSP. Unlike traditional solvers, including exact and heuristic methods, ML models can generalize beyond training data, solving larger problems without the need for re-optimization, and enabling fast inference in milliseconds. These properties make ML-based TSP solvers particularly advantageous in real-time, dynamic, and resource-constrained environments. ML-based approaches are classified into traditional ML and deep learning (DL) approaches, including Pointer Network (Ptr-Net)-based, Graph Neural Network (GNN)-based, Transformer-based, hybrid DL-heuristic-based, and multi-model DL-based approaches. Our analysis highlights the growing dominance of GNNs and Transformers, attributed to their ability to capture complex inter-node relationships and manage the graph-based nature of TSP. Hybrid DL-heuristic approaches, which integrate DL with heuristics like Lin-Kernighan-Helsgaun (LKH), exhibit superior performance on large instances, combining learning flexibility with heuristic efficiency. A comparative summary and discussion highlight each approach's strengths and limitations, identifying key challenges such as scalability, computational overhead, generalization and preserving quality at scale. Finally, the paper outlines promising future directions, emphasizing the need for scalable, generalizable, and resource-efficient solutions to address real-world TSP applications.

Keywords Traveling salesman problem · Machine learning · Deep learning · Graph neural network · Transformers

✉ Entesar Alanzi
eaalanzi@imamu.edu.sa

Mohamed El Bachir Menai
menai@ksu.edu.sa

[1] Department of Computer Science, College of Computer and Information Sciences, King Saud University, Riyadh, Saudi Arabia

[2] Department of Computer Science, Imam Mohammad Ibn Saud Islamic University (IMSIU), Riyadh, Saudi Arabia

1 Introduction

The Traveling Salesman Problem (TSP) is a classic NP-hard optimization problem, where the objective is to find the shortest possible route that visits a set of cities exactly once and returns to the starting point. It is based on the concept of Hamiltonian cycles, introduced by William Hamilton in the 19th century (Applegate et al. 2006b), and was later formulated as an optimization problem in the 20th century by Karl Menger (Schrijver 2005). The TSP is not merely a theoretical problem; it also has practical applications in numerous real-world scenarios where efficiency in routing and scheduling is crucial. These include logistics (Larni-Fooeik et al. 2024), robotics (Agatz et al. 2018), DNA sequencing (Wang et al. 2024), and urban planning (Mostafa et al. 2023).

There are two traditional methods to tackle TSP: exact and heuristic algorithms. Exact algorithms, including Dynamic Programming (DP) (Held and Karp 1962) and Branch and Bound (BB) (Balas and Toth 1983) techniques, are designed to guarantee the identification of an optimal solution; however, they become impractical as the number of cities grows. Concorde (Applegate et al. 2006a) and Gurobi (Bixby et al. 2008) are the leading exact solvers. The largest instance solved to optimality by the Concorde solver is a standard instance with 85,900 cities. Although the total wall-clock running time was approximately 568.9 h, the computational effort required was equivalent to over 136 CPU years (Applegate et al. 2009). For instances that are significantly larger than this, Concorde fails to terminate within a reasonable period (Applegate et al. 2006b).

Heuristic algorithms, such as the Lin-Kernighan-Helsgaun (LKH) series (Helsgaun 2000, 2009, 2017) and the Edge Assembly Crossover (EAX) operator (Nagata and Kobayashi 2013) are designed to balance solution quality with computational efficiency. However, these methods often face significant limitations when scaling to larger instances. For instance, the LKH algorithm utilizes a 1-tree pre-processing method with quadratic complexity, making it challenging to apply efficiently to instances with millions of cities. Similarly, EAX is effective on small to medium-sized TSP instances but becomes computationally prohibitive as problem size grows. These limitations highlight the need for scalable solutions that maintain high solution quality across a range of instance sizes (Cheng et al. 2023). To address these challenges, Machine Learning (ML) techniques, especially Deep Learning (DL), have gained popularity. Inspired by breakthroughs in fields such as image processing and Natural Language Processing (NLP), ML has been explored as a powerful alternative for combinatorial optimization. Recently, deep neural networks have been employed to autonomously learn solutions for complex optimization problems, including the TSP (Joshi et al. 2022). This approach, termed Neural Combinatorial Optimization (NCO), represents a paradigm shift. Traditional solvers rely heavily on manual design and tuning by experts, while NCO automates the process through training, effectively generating solvers that require minimal human intervention (Liu et al. 2023). The TSP has served as a primary testbed for NCO, with many neural network architectures originally developed for TSP applications influencing other combinatorial optimization problems. Most neural TSP solvers focus on Euclidean TSP instances, where nodes are symmetrically positioned on a two-dimensional plane, simplifying distance calculations between nodes (Liu et al. 2023). Though NCO incurs higher offline training costs, it reduces human effort in solver design and offers automated scalability to diverse TSP scenarios.

ML-based solvers offer significant advantages in solving the TSP, particularly in terms of efficiency in time and energy consumption. Unlike traditional methods such as Concorde or LKH solvers that require extensive computational resources, deep domain-specific knowledge and become impractical for very large-scale problems, ML-based solvers excel in these environments. These solvers, centered around a learning-driven paradigm, are designed to develop a solver by training that effectively handle large-scale instances and provide solutions much faster and with substantially lower resource consumption. For example, experiments with small-sized, randomly generated TSP instances have demonstrated that ML-based solvers can achieve solution quality comparable to traditional solvers while using only about one-tenth of the computational resources typically required (Liu et al. 2023).

ML solvers are especially valuable in industries such as logistics and transportation, which require quick decisions because of their scalability and effectiveness. Furthermore, as more data is processed, ML models' adaptability allows them to improve better over time, which makes them ideal for dynamic environments where TSP configurations change frequently. Traditional methods, which are often static and need to be manually tuned for various problem instances, are in sharp contrast to this adaptability.

Additionally, ML-based solvers can learn from new data, which enables them to adapt their strategies in response to evolving conditions and requirements. In real-world applications where problem instances may change over time, this capacity for continuous learning gives ML-based solvers a significant edge. As a result, ML-based solvers provide a flexible and resource-efficient substitute for exact and heuristic solvers, producing scalable, adaptive solutions that perfectly suit the requirements of complex, large-scale TSP applications.

State-of-the-art ML approaches for the TSP utilize a variety of models including Pointer Networks (Ptr-Nets) (Vinyals et al. 2015), Graph Neural Networks (GNNs) (Joshi et al. 2019; Sui et al. 2023; Xing and Tu 2020), Transformers (Bresson and Laurent 2021; Kool et al. 2019; Kwon et al. 2020), Convolutional Neural Networks (CNN) (Miki et al. 2018; Miki and Ebara 2019), traditional ML techniques (Mele et al. 2021), Multi-model DL-based approaches (Ye et al. 2024; Pan et al. 2023; Zheng et al. 2024), and hybrid DL-heuristic-based approaches (Fu et al. 2023; Tian et al. 2024; Li et al. 2024). These approaches can be broadly classified into end-to-end approaches, which aim to generate solutions in a single process (Bello et al. 2017; Deudon et al. 2018; Khalil et al. 2017; Sui et al. 2021; Vinyals et al. 2015), and iterative search-based approaches, which refine solutions over multiple cycles (Fu et al. 2021; Hudson et al. 2022; Xin et al. 2021b; Zheng et al. 2023). In the domain of solution construction, the proposed methods are divided into constructive methods, where solutions are built incrementally step-by-step (Bello et al. 2017; Deudon et al. 2018; Khalil et al. 2017; Kool et al. 2019; Kwon et al. 2020; Luo et al. 2023; Ma et al. 2019; Ouyang et al. 2021; Vinyals et al. 2015; Yang et al. 2023), and improvement methods, which iteratively enhance a complete but suboptimal solution (Fu et al. 2021; Joshi et al. 2019; Nowak et al. 2017; Qiu et al. 2022; Sanyal and Roy 2022; Sui et al. 2023; Xin et al. 2021b; Zheng et al. 2023). Moreover, the learning models can be trained using supervised learning, mimicking optimal solvers (Hudson et al. 2022; Sanyal and Roy 2022; Sui et al. 2023; Sun and Yang 2023), or through reinforcement learning, optimizing by trial and error based on feedback like the minimization of TSP tour length (Grinsztajn et al. 2023; Jung et al. 2024; Kim et al. 2023; Mele et al. 2021; Pan et al. 2023; Zheng et al. 2023). These ML-based solutions provide unique advantages for solving TSP instances in a reasonable time, addressing all stages of the problem from initial solution to refinement.

Our study provides a comprehensive and up-to-date examination of ML-based solvers specifically for TSP. Unlike previous surveys (Bengio et al. 2021; Bogyrbayeva et al. 2022; Junior Mele et al. 2021; Shi and Zhang 2022; Mazyavkina et al. 2021; Yang et al. 2024), which either focus broadly on combinatorial optimization or narrowly on specific ML paradigms, our work offers a holistic perspective tailored to the TSP, addressing practical challenges and identifying promising directions for future research. Our contributions can be summarized as follows:

- We present a comprehensive review of ML approaches for solving the TSP, covering both traditional ML methods and a wide range of DL-based approaches, including Ptr-Net-based, GNN-based, Transformer-based, Multi-model DL-based, Hybrid DL-Heuristic-based, and other DL-based approaches such as CNN-based models.
- We provide a detailed analysis of the strengths and limitations of each approach in terms of scalability, computational efficiency, solution quality, and generalization to large-scale instances. This analysis is supported by comparative tables and insightful discussions to enhance clarity and depth.
- We identify and discuss current challenges and future research directions, helping to guide subsequent studies and advancements in ML for the TSP.

The remainder of this paper is organized as follows. Section 2 provides the necessary background and fundamental concepts relevant to the TSP and ML. Section 3, Related Work, reviews various ML-based approaches that have been applied to the TSP, and includes a comparative summary of different approaches. Section 4 analyzes major advancements, highlights challenges, and discusses future research opportunities. Finally, Sect. 5 summarizes the key findings and concludes the paper.

2 Background knowledge

In this section, we provide a fundamental overview of the TSP and ML techniques, specifically aiming to introduce essential concepts necessary for understanding the rest of the paper.

2.1 Traveling salesman problem (TSP)

2.1.1 Definition

Formally in graph theory, the TSP can be defined on a complete undirected graph $G = (V, E)$ if it is symmetric or on a directed graph $G' = (V, E)$ if it is asymmetric, where V is a set of n nodes such that $V = \{1, \dots, n\}$ and E is a set of edges such that $E = \{(i, j) : i, j \in V, i \neq j\}$. Let d_{ij} be the distance from city i to city j, and $x_{ij} \in \{0, 1\}$ be a binary decision variable, which takes value 1 if the tour includes traveling from city i to city j, and 0 otherwise. Each city is reached from precisely one other city and serves as a departure point to exactly one other city (Davendra 2010). For asymmetric TSPs, the distance d_{ij} is not necessarily equal to d_{ji}. The objective function is to minimize the tour length, as in (1):

$$\min \sum_{i=1}^{n} \sum_{\substack{j=1 \\ j \neq i}}^{n} d_{ij} x_{ij}, \quad x_{ij} \in \{0,1\} \tag{1}$$

Subject to:

$$\sum_{\substack{j=1 \\ j \neq i}}^{n} x_{ij} = 1 \quad \forall i \in V \tag{2}$$

$$\sum_{\substack{i=1 \\ i \neq j}}^{n} x_{ij} = 1 \quad \forall j \in V \tag{3}$$

$$\sum_{i \in S} \sum_{j \in S} x_{ij} \leq |S| - 1 \quad \forall S \subset V, 2 \leq |S| \leq n-1 \tag{4}$$

In this formulation, (2) and (3) are identified as degree constraints, stipulating that each node must be traversed exactly once. Constraint (4) is known as the subtour elimination constraint, which prevents the formation of smaller, isolated tours within the solution. The notation $|S|$ represents the count of elements in the subset S(Purkayastha et al. 2020).

Although the TSP is straightforward in its description, it is an NP-hard problem (Karp 1972), exhibiting an exhaustive search complexity of $O(n!)$. This means that the time required to solve the problem increases exponentially with the number of cities, making it challenging to solve large instances optimally (Purkayastha et al. 2020).

2.1.2 TSP variants

The TSP is broadly classified into Symmetric TSP (STSP), Asymmetric TSP (ATSP), and Multiple TSP (MTSP) (Ilavarasi and Joseph 2014). These three types represent the core structural variations of the problem. Any real-world problem that can be reduced to TSP falls into one of these categories:

- **STSP** assumes that the distance between two cities is identical in both directions, making it applicable to problems where the travel distances, times, or costs remain unchanged regardless of the route's direction. Many real-world problems exhibit symmetry in travel costs. Examples include logistics on undirected road networks, where the distance between two cities remains constant regardless of the direction (Kumar et al. 2024), circuit board manufacturing, and drilling problems, where movement costs between positions do not depend on direction (Davendra 2010).
- **ATSP** extends the problem to cases where travel costs differ based on direction. It is represented by a directed graph where edge weights vary. A real-world problem is classified as ATSP when it involves directed movement with asymmetric travel costs, such

as varying fuel consumption, road restrictions, or wind resistance. This type occurs in transportation systems with one-way streets, flight networks, or robotic motion planning, where returning to a previous location may have a different cost than reaching it (Cook et al. 2024).

- **MTSP** generalizes the TSP by introducing multiple agents. Instead of a single salesman, several must collectively visit all locations. A problem is classified as MTSP when it requires more than one agent to complete the tour while minimizing total costs or travel time. This type is essential in vehicle routing (Davendra 2010), workforce scheduling, and autonomous aerial vehicles (Fang and Luo 2024), where multiple units share the task of covering destinations efficiently.

More specialized TSP variants extend these foundations to address specific constraints, including: the Euclidean TSP, which assumes cities are points on a plane with straight-line distances; the TSP with Time Windows (TSPTW), which imposes time restrictions for visiting cities (Stohy et al. 2021); the Time-Dependent TSP (TDTSP), which accounts for travel times varying with temporal factors (Barrena et al. 2023); and the Generalized TSP (GTSP), which groups cities into clusters, requiring exactly one city from each cluster to be visited (Pop et al. 2024).

2.1.3 Solution methods

Many algorithms have been designed to tackle TSP, each varying significantly in terms of time and space complexities. These algorithms are primarily classified into three categories: exact, heuristic, and ML-based types.

- **Exact methods**, including Dynamic Programming (DP) and Branch-and-Bound (BB), aim to find optimal solutions by exhaustively exploring the solution space.
  DP, exemplified by the Held-Karp algorithm (Held and Karp 1962), breaks the problem into smaller subproblems and stores their solutions to avoid redundant computations. This approach guarantees optimality but has exponential time complexity $O(n^2 \cdot 2^n)$ and high memory requirements $O(n \cdot 2^n)$, making it impractical for large instances (Borchate et al. 2024). Recent improvements, such as Approximate DP (ADP), Neuro-Dynamic Programming (NDP), and hyper-heuristics, reduce the computational burden by approximating value functions and limiting the number of stored subproblems (Anson 2024).
  BB (Balas and Toth 1983) explores the solution space using branching to generate possible solutions and bounding functions to eliminate suboptimal branches. Although its worst-case time complexity remains $O(n!)$, pruning significantly reduces the number of explored nodes, making it more scalable than DP. BB also naturally integrates heuristics to guide the search toward promising solutions and improve computational efficiency (Zabinsky et al. 2019; Jiang et al. 2014).
  Table 1 summarizes the key differences between these approaches, including their computational complexity, memory usage, scalability, and heuristic integration. The time complixty and memory requirements of DP limit its use to small instances. BB, in contrast, is more scalable due to its pruning mechanism and better supports heuristic integration, making it more practical for larger problem sizes when combined with effective

Table 1 Comparison of DP and BB techniques (n is the number of cities)

Feature	DP	BB
Approach	Uses recursion with memorization to explore all subsets	Explores solution space with pruning
Time Complexity	$O(n^2 2^n)$(Held and Karp 1962)	$O(n!)$
Space Complexity	$O(n2^n)$	$O(n!)$ (if the search tree is fully explored)
Scalability	Limited to small instances	Can handle larger instances than DP with effective bounding strategies
Heuristic Enhancement	Can integrate heuristics to discard unpromising subproblems, ignore redundant calculations and reduce memory overhead (Anson 2024)	Can integrate heuristics to prune large portions of the search space, generate good initial solutions that serve as tighter bounds, eliminate infeasible branches early and allow faster convergence (Jiang et al. 2014)

bounding techniques (Jiang et al. 2014).

- **Heuristic methods** focus on finding near-optimal solutions within reasonable computation time. The Lin-Kernighan algorithm (LK) (Lin and Kernighan 1973) employs a local search strategy using iterative k-opt moves, where segments of a tour are rearranged to reduce its length. The LKH algorithms (Helsgaun 2000, 2009, 2017) are among the most effective local search-based heuristics, which incorporate a series of powerful local search strategies based on the classic LK move operator. While heuristics are significantly faster than exact methods, they often require problem-specific tuning and may struggle to generalize across diverse TSP variants or large-scale instances.
- **Machine learning-based approaches**
  ML is commonly categorized into three main types:
  Supervised Learning (SL) is the most basic and widely used type of ML. It involves training a model on a labeled dataset, where the input–output pairs are known. The model is trained to associate inputs with their correct outputs by reducing the discrepancy between its predictions and the actual labels. While SL can achieve high accuracy and is relatively easy to implement and interpret, it requires a large amount of labeled data, which can be expensive and time-consuming to obtain (Mitchell and Mitchell 1997).
  Unsupervised Learning (UL) involves training a model on a dataset without labeled responses. The model tries to learn the underlying structure of the data by identifying patterns, clusters, or associations. A common example of UL is clustering, where similar data points are grouped together. The UL can work with unlabeled data, which is more readily available and is valuable for exploratory data analysis. However, evaluating the performance of unsupervised models is challenging, and the results can be less interpretable compared to SL (Mitchell and Mitchell 1997).
  Reinforcement learning (RL) trains models to make sequential decisions by interacting with an environment. An agent observes a state, takes an action, and receives feedback in the form of rewards or penalties, aiming to maximize cumulative rewards over time. RL is well-suited for tasks where the correct actions are not predefined but must

be discovered through trial and error. However, it requires extensive interactions with the environment, making training resource-intensive. Managing the trade-off between exploration (discovering new strategies) and exploitation (using known strategies) is critical. Popular RL algorithms include Q-Learning, SARSA, Deep Q-Network (DQN), and Asynchronous Advantage Actor-Critic (A3C) (Andrew and Richard 2018).

- Recent advancements in DL have led to the development of neural network-based approaches for solving the TSP. Enabled by the computational power of GPUs, these methods leverage specialized architectures to learn efficient heuristics and optimize tour generation. Among these, Ptr-Net-based, GNN-based, and Transformer-based models have emerged as key approaches:

  – **Pointer networks (Ptr-Net)-based approaches** Pointer Networks are a type of neural network designed to solve combinatorial optimization problems, such as the TSP, by directly generating a permutation of the input sequence (Vinyals et al. 2015). Ptr-Nets use attention mechanisms to adaptively select elements from the input sequence as output rather than producing fixed-size outputs. Figure 1 depicts the Ptr-Net architecture. To solve a TSP instance, the encoder processes the input sequence, such as the example $\{X_1, X_2, X_3, X_4\}$ shown in the figure, where each X_i represents the coordinates of city *i*. The encoder maps these inputs into hidden states. At each decoding step *i*, the decoder selects the next city index C_i in the tour by attending to the encoder's hidden states.
  The attention mechanism computes scores u_{ij} for each city *j* as:

$$u_{ij} = v^\top \tanh(W_1 h_j + W_2 d_i) \tag{5}$$

where h_j is the encoder's hidden state for input city X_j, and d_i is the decoder's hidden state at step *i*. W_1, W_2, and *v* are learnable parameters, and $v^\top$ denotes the transpose of *v* (Vinyals et al. 2015).

The scores are normalized using the softmax function:

$$p(C_i | C_1, \ldots, C_{i-1}, X) = \mathrm{softmax}(u_i) \tag{6}$$

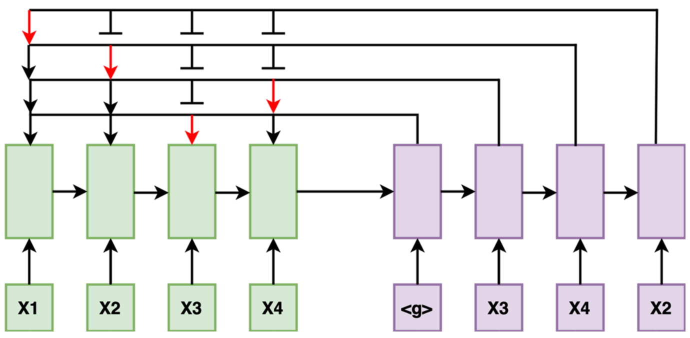

Fig. 1 A pointer network architecture introduced by (Vinyals et al. 2015)

The decoder selects the next city index C_i based on the highest probability $p(C_i)$, as shown by the red arrows in Fig. 1. This process continues until a complete tour is generated (Vinyals et al. 2015).

- **GNN-based approaches:** GNNs are specialized for processing data structured as graphs, making them well-suited for combinatorial optimization problems like the TSP. GNNs perform message passing, where nodes aggregate information from their neighbors through edges, iteratively updating their embeddings. This process enables GNNs to capture both local and global graph properties that are difficult to achieve with traditional neural networks. Notable GNN architectures include:

  Graph convolutional network (GCN): GCNs use convolutional operations to propagate information between nodes in a graph. They leverage a localized aggregation of neighboring node features to update the representations of the nodes (Khemani et al. 2024).

  Graph attention network (GAT): Introduced by Veličković et al. (Veličković et al. 2018) in 2018, GAT assigns attention coefficients to neighboring nodes of a target node, allowing the network to focus on the most relevant neighbors during the information propagation process. This mechanism enhances the model's ability to prioritize critical relationships in the graph (Khemani et al. 2024).

In the context of TSP, the GNN model predicts a heatmap that indicates the likelihood of edges being included in the optimal TSP tour. The solver then iteratively searches for an approximate solution using this heatmap as guidance. Figure 2 illustrates a GNN-based solver for TSP. Each TSP instance is modeled as a complete graph $G = (V, E)$, where each node $v \in V$ represents a city, and each edge $e_{vu} \in E$ represents a path between cities with the distance as its feature. The embeddings of nodes $h_v^{(L)}$ and edges $e_{vu}^{(L)}$ are learned through L layers of message passing based on neighboring nodes and edges. The message-passing process follows:

$$h_v^{(l+1)} = \text{UPDATE}\left(h_v^{(l)}, \sum_{u \in N(v)} \text{MESSAGE}(h_v^{(l)}, h_u^{(l)}, e_{vu}^{(l)})\right) \tag{7}$$

where $h_v^{(l)}$ is the hidden state of node v at layer l, and $N(v)$ represents the neighbors of node v. The MESSAGE function aggregates information from neighboring nodes and edges, and the UPDATE function refines node embeddings accordingly (Khemani et al. 2024).

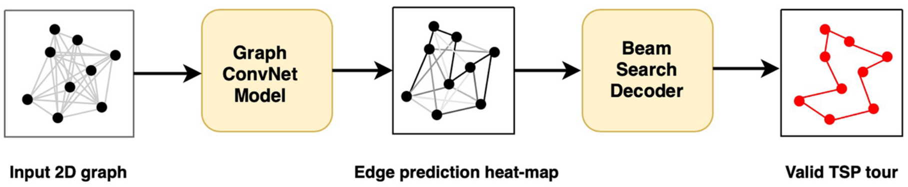

Fig. 2 A GNN-based approach proposed by Joshi et al. (2022)

After L layers, the final edge embeddings $e_{vu}^{(L)}$ are used to predict the probability of including edge e_{vu} in the tour:

$$p(e_{vu}) = \sigma\left(We_{vu}^{(L)}\right) \tag{8}$$

where W is a learnable parameter and σ is a sigmoid function (Khemani et al. 2024). The edges with the highest probabilities are selected to form the TSP tour, ensuring all nodes are visited exactly once. A decoder or heuristic (e.g., beam search) ensures that the selected edges form a valid Hamiltonian cycle (Joshi et al. 2022).

- **Transformer-based approaches:** Trnasformers represent a significant advancement in DL, particularly for NLP tasks, pioneered by Vaswani et al. (Vaswani et al. 2017). Unlike RNNs, Transformers do not rely on sequential processing and instead use self-attention mechanisms to capture dependencies between all elements in a sequence simultaneously. This allows for efficient parallelization and improved handling of long-range dependencies (Yang et al. 2023). As illustrated in Fig. 3, a Transformer consists of encoder and decoder stacks, each composed of multi-head self-attention layers and feed-forward networks. The encoder processes input embeddings with positional encodings, while the decoder refines predictions using masked self-attention and cross-attention over encoder outputs. This parallelized architecture significantly enhances performance in tasks requiring complex sequence modeling, such as machine translation and combinatorial optimization. In the context of TSP, Transformer-based approaches encode node coordinates as input features and iteratively construct near-optimal tours by attending to relevant nodes at each decision step. This architecture enables Transformer models to efficiently explore the solution space, learning effective heuristics for route optimization.

2.1.4 Benchmarking and testing

Benchmarking instances for the TSP are essential for evaluating the efficacy of different TSP algorithms. These instances act as standard test cases, providing a consistent and fair basis for assessing various methods. The use of benchmarking instances is pivotal for both the development of novel algorithms and the refinement of existing solutions. There are two primary approaches to obtaining TSP instances: data generation and established benchmark sets (Liu et al. 2023).

- **Data generation**: Instances are generated to encompass a wide range of problem characteristics:
  - **Uniform random euclidean (RUE) instances**: These instances are generated uniformly by placing points on a two-dimensional plane.
  - **Clustered instances (CLU)**: These instances are generated by randomly placing points around different central points.
- **Existing benchmark sets**:

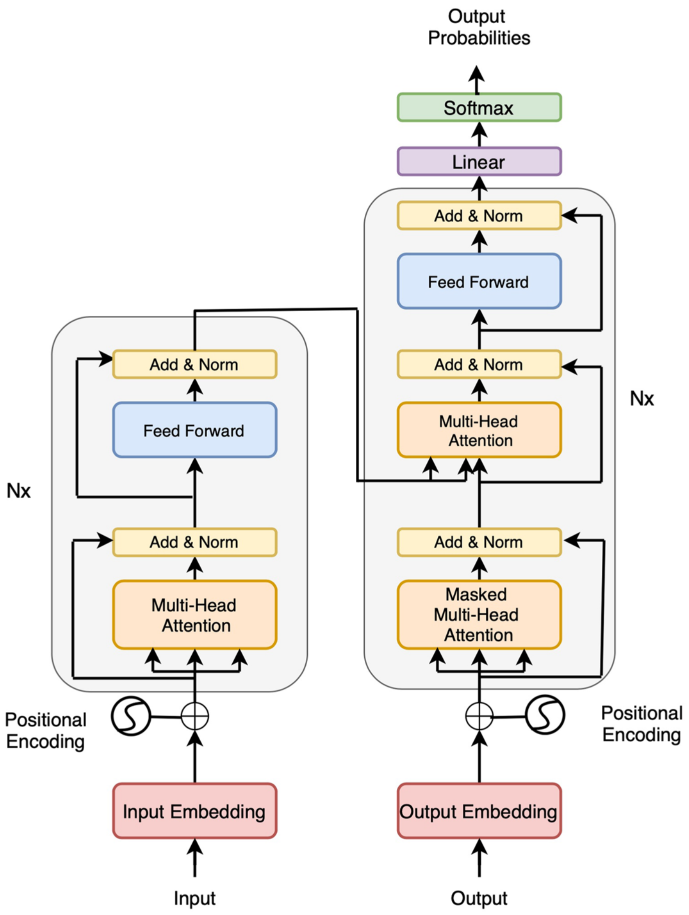

Fig. 3 The transformer architecture (Vaswani et al. 2017)

- **TSPLIB** (Reinelt 1991): Introduced by Gerhard Reinelt in 1991, TSPLIB is a comprehensive library of TSP and related problem instances ranging from small sets (e.g., 14 cities) to extremely large sets (e.g., 85,900 cities). It is recognized as a fundamental dataset for algorithm benchmarking.
- **Very large-scale integration (VLSI) instances** (Rohe 2013): Derived from the challenges in designing integrated circuits, these instances are typically complex and feature thousands to tens of thousands of nodes, making them suitable for testing the scalability of TSP algorithms (Liu et al. 2023).
- **National instances** (Cook 2022): Based on real-world geographic data, these instances model the practical challenges of routing and logistics across actual city locations within a specific country.

3 Related work

This section reviews state-of-the-art ML approaches for solving the TSP. From a model-based perspective, ML approaches are categorized into: traditional ML-based approaches and DL-based approaches. The DL-based approaches are further divided into Ptr-Net-based, GNN-based, Transformer-based, Multi-model DL-based, Hybrid DL-Heuristic-based, and Other DL-based approaches. **Ptr-Net-based approaches** use sequence-to-sequence models with attention mechanisms to generate tour permutations. **GNN-based approaches** represent the TSP as a graph and learn from node and edge interactions using message passing. **Transformer-based approaches** apply self-attention to model global dependencies and capture complex relationships among nodes. **Multi-model DL-based approaches** combine different architectures, such as GNNs and Transformers, to benefit from their complementary capabilities. **Hybrid DL-Heuristic approaches** integrate DL with classical heuristics like LKH or GLS to guide or enhance the search process. **Other DL-based approaches**, including CNN-based models, reframe the TSP as a spatial or image-based task and apply convolutional architectures to learn tour construction patterns. Figure 4 illustrates this classification.

Additionally, ML-based approaches can be classified based on how ML is integrated with traditional TSP algorithms into:

- **End-to-end approaches** rely entirely on ML, where a TSP instance is provided as input, and the model directly generates the final tour. These approaches, such as Pointer Network-based and Transformer-based models, are particularly suitable for real-time applications due to their fast inference and minimal reliance on problem-specific adjustments. However, their scalability is limited, as larger problem instances significantly increase memory consumption and computational overhead (Bello et al. 2017; Deudon et al. 2018; Khalil et al. 2017; Sui et al. 2021; Vinyals et al. 2015).
- **Two-stage approaches** use the power of ML to learn and the search efficiency of heuristic algorithms (such as beam search, GLS, MCTS, and LKH) to work together. This combination makes it easier for these models to handle larger TSP instances more effectively by iteratively refining initial solutions. Despite their success, they require longer computation time and careful tuning of search operators, demanding more computational resources and expert knowledge (Joshi et al. 2022; Nowak et al. 2017; Hudson et al. 2022; Xin et al. 2021b; Zheng et al. 2023).
- **Decomposition-based approaches** tackle large-scale TSP instances by breaking them

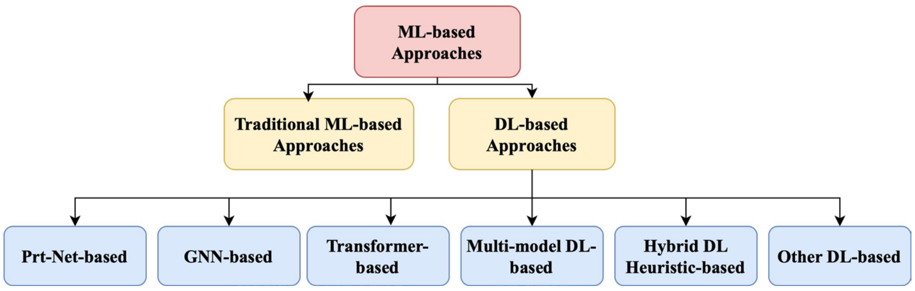

Fig. 4 A classification of ML-based approaches to the TSP

into smaller subproblems, solving each using ML-based solver independently, and then merging them into a complete tour. This decomposition strategy significantly improves scalability, making it particularly effective for large-scale TSP instances. (Pan et al. 2023; Fu et al. 2021; Cheng et al. 2023; Chen et al. 2023; Zheng et al. 2024).

3.1 DL-based approaches

Recent advancements in DL have led to promising developments in solving combinatorial optimization problems such as the TSP. This section categorizes DL-based approaches into six approaches: Ptr-Net-based approaches, GNN-based approaches, Transformer-based approaches, Multi-model DL-based approaches, Hybrid DL-Heuristic-based approaches, and Other DL-based approaches. Each of these methods leverages different neural architectures to learn problem structures and optimize solutions.

3.1.1 Ptr-net-based approaches

Vinyals et al. (2015) introduced Ptr-Nets, a neural architecture designed for sequence-to-sequence tasks where output tokens correspond to positions in the input sequence. They applied Ptr-Net to the TSP, demonstrating its ability to generate near-optimal tours for instances with up to 50 nodes. However, performance deteriorated as the problem size increased, particularly beyond 20 nodes. Bello et al. (2017) extended Ptr-Net by applying an RL approach to the TSP. Their model was optimized using a policy gradient technique with the negative tour length as the reward signal. It outperformed the original SL approach of Ptr-Net and surpassed traditional heuristics, including Christofides' heuristic Christofides (2022), for instances with up to 100 nodes.

da Costa et al. (2021) proposed a DRL-based local search heuristic for TSP using 2-opt operators. The model, trained via policy gradient, learns to refine solutions by incorporating search history. It achieves near-optimal solutions even from poor initial tours. Unlike traditional heuristics, it requires only random initialization and sampling. It outperformed 2-Opt First Improvement (FI) and Best Improvement (BI), even with restart strategies. However, it requires a large number of samples for effective training. Barro (2023) proposed the Pointer Q-Network (PQN) that integrates Ptr-Nets with model-free Q-learning. By combining attention scores with Q-values, PQN balances immediate accuracy with long-term planning. It outperformed Ptr-Nets on TSP20 and TSP50. However, its scalability to larger instances remains limited due to computational costs.

Table 2 provides a summary of the Ptr-Net-based approaches, including their solutions, performance measures, advantages, and disadvantages.

3.1.2 GNN-based approaches

Khalil et al. (2017) proposed S2V-DQN, an early approach that combined GNNs with RL to solve the TSP. The model integrated Structure2Vec (Dai et al. 2016), a GNN-based embedding method, with Deep Q-Networks (DQN) to sequentially construct TSP tours by learning optimal node insertion. Trained on graphs with 50-100 nodes and tested on instances up to 1200 nodes, it demonstrated strong generalization across different problem sizes and outper-

Table 2 Summary of Ptr-net-based approaches

References	Solution	Performance measure	Advantages	Disadvantages
Vinyals et al. (2015)	Ptr-Net: seq2seq model with input attention	**RUE instances (up to 50 nodes):** For $n < 20$: results are close to optimal solutions. For $n > 40$: The performance decreases. **Tour length:** TSP20: 3.88 TSP50: 6.09	It overcomes limitations of standard seq2seq models	Focus on small problem sizes. Performance degrades as the problem size increases significantly. SL needs optimal labels which are difficult to obtain for large instances
Bello et al. (2017)	Ptr-Net trained using the Actor-Critic RL algorithm	**RUE instances (up to 100 nodes):** Tour length: TSP20: 3.82 TSP50: 5.70 TSP100: 7.83	Training the Ptr-Net with RL significantly improves performance	Evaluation is conducted on small instances with up to 100 nodes
da Costa et al. (2021)	Used Ptr-Nets to learn 2-opt heuristic via DRL	**Sample size: 2,000** TSP20: 0.00% TSP50: 0.12% TSP100: 0.87% **TSPLIB (up to 280 cities):** Avg optimality gap 4.56%	It can be easily adapted to general k-opt and is more sample-efficient	Requires more computational time compared to the 2-opt approach. Evaluation is limited to relatively small TSP instances (up to 100 nodes)
Barro (2023)	Ptr-Net + Q-learning	**RUE instances (up to 50 nodes):** Tour length: TSP20: 3.8563 TSP50: 6.3441	Combines short-term accuracy (attention scores) with long-term reward planning (Q-values)	Limited scalability due to computational cost. Evaluated only on small-scale TSP instances

formed previous DL-based methods. This work pioneered the use of GNNs in combinatorial optimization, influencing later ML-based TSP solvers.

In contrast to autoregressive approaches, Nowak et al. (2017) proposed a GNN with a SL approach to generate TSP tours as heat map (adjacency matrices) which represents the likelihood of edges being part of the TSP tour. The model uses beam search to construct feasible solutions in a single step. It consists of 40 GNN layers with over 10 million parameters, enabling pattern memorization in small TSP instances. However, it does not generalize well to larger problems and underperforms compared to autoregressive models. On the top of (Nowak et al. 2017), Joshi et al. (2019) developed a non-autoregressive model using 30 layers of graph convolution networks to extract node and edge features. It predicts edge probabilities, forming a heatmap converted into a tour via beam search. Its parallelized structure enables faster evaluations than RL approaches. The model surpasses autoregressive DL approaches in solution quality, inference speed, and sample efficiency for small problems (50-100 nodes). However, it tends to memorize patterns rather than learning adaptable strategies.

Drori et al. (2020) introduced a unified framework to solve combinatorial optimization problems on graphs, such the TSP, using RL with GNN representations. For a variety of instances, this method yields linear running times with optimality gaps near to 1%, demonstrating notable speed advantages over traditional methods. A GAT is used as the encoder and an attention model is used as the decoder in their encoder-decoder architecture. Experimental results show that this method generalizes well from training on TSP100 to running on large graphs for TSP with 250 nodes. Also shows that it generalizes well from random graphs to TSPLIB graphs (up to 225 nodes) with a mean optimality gap of 1.032%.

Hu et al. (2021) developed a Bidirectional GNN (BGNN) for TSP on arbitrary symmetric graphs. The model uses Bidirectional Message Passing Layers (BMPL) to extract decision-making features from graph edges. When combined with a best-first search algorithm, it explores a broader set of nodes, producing shorter tours. Trained on instances with 70 to 75 cities and tested on graphs with up to 200 cities, the BGNN demonstrated strong generalization. It achieved faster solutions than Gurobi (Bixby et al. 2008) and produced more optimal results than ORTools.

Scalability is a key challenge in neural TSP solvers, particularly when handling large instances with thousands of nodes. Recent studies have focused on designing models that efficiently scale while maintaining high solution quality. Ouyang et al. (2021) proposed eMAGIC, a DRL framework for TSP, designed to enhance generalization across different instance sizes. The model uses a GNN and MLP encoder with an attention-based decoder. It integrates three training techniques: interleaved local search, smoothed policy gradient, and stochastic curriculum learning, improving solution quality. eMAGIC demonstrated strong generalization, solving instances up to 1000 nodes after training on small-scale TSP (< 50 nodes). Qiu et al. (2022) introduced the DIfferentiable MEta Solver (DIMES), a compact continuous space for parameterizing the underlying distribution of candidate solutions and proposed a meta-learning framework for CO problem instances. This framework is designed to effectively initialize model parameters during the fine-tuning stage. DIMES outperformed strong baselines in DRL-based solvers and scaled effectively to TSP instances up to 10,000 nodes, though at the cost of increased computational time. Sun and Yang (2023) proposed DIFUSCO, which stands for the graph-based DIFfUsion Solvers for Combinatorial Optimization. This model combines the strengths of probabilistic modeling and GNNs to address NP-complete problems, including TSP. The core of DIFUSCO lies in its application of denoising diffusion models, which operate through a two-stage process. The forward process gradually introduces noise, transforming the solution from a near-optimal state into one of high entropy. Conversely, the reverse process systematically reduces this noise, thereby refining the solution towards optimality. This model iteratively adjusts the edge inclusion probabilities based on the denoised data, effectively enhancing the solution quality. Experimental results validate that DIFUSCO markedly surpasses existing state-of-the-art neural solvers.

Graph sparsity is a crucial factor in optimizing GNN performance, Lischka et al. (2023) explored two data preprocessing methods that enhance the efficiency of GNNs by simplifying the TSP graph structure: a nearest neighbors heuristic and a minimum spanning tree method called 1-Trees, which make the corresponding TSP instances sparse by deleting unpromising edges. Performance was tested with GCNs and GATs on different sizes and distributions of TSP instances. Experiments with GCNs and GATs showed that GCNs perform best on highly sparse graphs, retaining only the top three edges per node in 100-node instances. GATs performed optimally when about half of the possible edges were retained.

Table 3 provide a summary of the GNN-based approaches, including their solutions, performance measures, advantages, and disadvantages.

Table 3 Summary of GNN-based approaches and their advantages and disadvantages

References	Solution	Performance measure	Advantages	Disadvantages
Khalil et al. (2017)	a Structure2Vec graph embedding network architecture trained with deep Q-learning	**S2V-DQN trained on TSP50-TSP100:** TSP 50-100: 1.0730 TSP400-500: 1.0944 TSP1000-1200: 1.1065 **TSPLIB (from 51 to 318 cities):** Avg opt. gap: 1.0475	Novel application of GNNs for TSP	Performance drop on larger instances
Nowak et al. (2017)	A non-autoregressive GNN with beam search	TSP20: 2.46%	Novel use of GNNs for QAP	Performance depends on high-quality training data. Underperforms for small TSP instances
Joshi et al. (2019)	Graph ConvNet and beam search	**Beam search (b = 1280) and shortest tour heuristic:** TSP20: 0.01% TSP50: 0.01% TSP100: 1.39%	Highly parallelized GCN implementation	SL model struggles with size invariance, poor generalization
Drori et al. (2020)	Graph attention network (GAT) as encoder and attention model as the decoder with RL	TSP100: 1.074 **TSPLIB (up to 255 cities, trained on TSP100):** Avg. opt. gap: 1.032	Linear running time	No comprehensive evaluation of training efficiency
Hu et al. (2021)	Bidirectional GNN (BGNN)	**Random mixed graph sizes:** Gap = $L_{\text{tour}}/L_{\text{best}}$: 1 is the better. $n \in [20, 25]$: 1.015 $n \in [30, 50]$: 1.021 $n \in [50, 100]$: 1.034	BMPL captures decision-related features from edges and partial solutions, enhancing quality	Solves small instances only
Ouyang et al. (2021)	GNN, MLP, and Attention mechanism + combined local search technique	TSP20: 0% TSP50: 0.01% TSP100: 0.02% TSP200: 0.50% TSP500: 2.92% TSP1000: 4.36% **TSPLIB (different ranges of instances):** (Size range): Avg. opt. gap (50-199): 0.46% (200-399): 1.37% (400-1002): 3.40%	Utilizes equivariance and invariance techniques to standardize inputs and improve generalization. Introduces combined local search heuristics to escape local optima	Limited generalization capabilities
Qiu et al. (2022)	DIMES: anisotropic GNN +MLP trained using DRL	**RL+AS+MCTS:** TSP500: 1.76% (2.15h) TSP1000: 2.46% (4.62h) TSP10,000: 3.19% (3.5 7h)	Robust performance among DRL-based solvers (up to 10,000)	Limitation in reasoning power compared to iterative methods
Sun and Yang (2023)	Graph-based diffusion model (combines probabilistic modeling and GNNs)	**DIFUSCO (16×SAMPLING)**: TSP50: −0.01% TSP100: −0.01% TSP500: 0.46 TSP1000: 1.17 TSP10000: 2.58	Handles large problems well. Achieves state-of-the-art results on TSP10k	High resource requirements

Table 3 (continued)

References	Solution	Performance measure	Advantages	Disadvantages
Lischka et al. (2023)	GCN and GAT with reduction techniques	**For GAT:** TSP100: 0.72% **For GCN:** TSP100: 1.4%	Effective graph reduction	Weak evaluation (on TSP100 only), unclear generalization

3.1.3 Transformer-based approaches

Instead of relying on the LSTM architecture, Deudon et al. (2018) introduced the first transformer-based model for TSP. Their approach improved solution quality by integrating a 2-opt local search heuristic. Kool et al. (2019) later refined this idea by eliminating recurrence, enabling parallelization, and introducing a rollout baseline for training. Their model achieved near-optimal solutions on instances up to 100 nodes. These works demonstrated the potential of transformer-based architectures in solving combinatorial optimization problems like TSP.

Building upon the approach introduced by Kool et al. (2019), Kwon et al. (2020) introduced Policy Optimization with Multiple Optima (POMO), a RL-based framework. The model leverages the symmetry in TSP solutions by generating multiple rollouts from different starting cities. This reduces training variance and improves exploration. POMO uses a shared baseline to enhance training stability and prevent convergence to local optima. It also applies multiple greedy rollouts during inference, improving solution quality and efficiency. Additionally, Bresson and Laurent (2021) extended transformer-based models by modifying the decoder architecture. They introduced a token city to optimize the starting point and used self-attention to construct queries from all cities in a partial tour. The model was trained with RL and decoded with beam search, achieving high accuracy with 0.004% optimality gap for TSP50 and 0.39% for TSP100.

Enhancing transformer-based TSP approaches Several studies have focused on improving the effectiveness of transformer-based solvers by refining solution quality, embedding strategies, and exploration methods. Wu et al. (2022) introduced a RL framework that integrates local search heuristics (2-opt, node swap) into a self-attention mechanism, refining solutions through iterative improvement. Xin et al. (2021a) proposed the Multi-Decoder Attention Model (MDAM), which enhances solution diversity using multiple construction policies and a customized beam search method, ensuring more robust exploration. In addition, they integrated an Embedding Glimpse (EG) layer into the model. This layer dynamically updates node embeddings by selectively re-embedding only the unvisited nodes, achieved by masking out the visited nodes in the top attention layer of the encoder. This process of selective attention ensures that the decoder concentrates exclusively on relevant nodes, thereby improving the accuracy of decisions at each construction step, enhancing the overall quality of each individual solution. To further improve embedding quality, Wang et al. (2023) developed BDRL, a framework combining BERT (Devlin et al. 2019) and RL, leveraging ReZero-enhanced transformers (Bachlechner et al. 2021) for deeper network training, though at the cost of increased model complexity. Ma et al. (2021) introduced the Dual-Aspect Collaborative Transformer (DACT), which independently learns positional and node embeddings to avoid noise and incompatible correlations. Also, they designed a novel Cyclic Positional Encoding (CPE) method that captures the circularity and symmetry inherent in the TSP. This method enables the transformer to handle cyclic inputs effectively and enhances its generalization performance. The model is trained with proximal policy optimization and applies curriculum learning to enhance sample efficiency. Similarly, Xu et al. (2022) introduced an RL-based attention model, incorporating batch normalization reordering and gated aggregation to refine node embeddings dynamically. While effective, it performed worse than DACT and MDAM on benchmark instances.

Several approaches further refined training and solution selection. Sun et al. (2024) introduced LCH-Regret, a regret-based encoding mechanism designed to enhance learning in existing transformer-based solvers (e.g., AM (Kool et al. 2019), POMO (Kwon et al. 2020), and MatNet (Kwon et al. 2021)), reducing optimality gaps through data augmentation techniques. Meanwhile, Grinsztajn et al. (2023) proposed Poppy, a population-based method that trains multiple agents with diverse policies, improving exploration across different problem distributions.

Memory-efficient transformer-based approaches Recent works have explored different strategies to reduce memory overhead. Yang et al. (2023) proposed a memory-efficient transformer-based network, termed Tspformer, which is designed to reduce the time and space complexities of the transformer-based models for handling large-scale TSP. They incorporated a sampled scaled dot-product attention mechanism that reduces both the time and space complexity from $O(n^2)$ to $O(n \log n)$, where n represents the length of input sequences. By removing the masked MHA layer in the decoder, the model further reduces parameter count and memory usage. Experiments on instances up to 1000 nodes showed significant reductions in memory usage and training time, though with a slight performance drop. Similarly, Jin et al. (2023) proposed Pointerformer, a novel end-to-end DRL approach tailored for the TSP. This method utilizes a reversible residual network in the encoder to significantly reduce memory consumption, diverging from traditional residual networks. It encodes TSP nodes with several residual attention layers and decodes solutions using a multi-pointer network, which relies on queries enhanced by comprehensive context embeddings. To further refine solution quality, Pointerformer incorporates feature augmentation to explore TSP symmetries during both training and inference stages. The model demonstrates competitive performance on TSP500 and generalizes effectively to TSPLIB instances.

Scalability-focused transformer-based approaches Several studies have tackled this challenge by introducing destroy-and-repair mechanisms, iterative refinement techniques, and decomposition strategies. Cheng et al. (2023) proposed an iterative transformer-based framework that selectively samples and optimizes subproblems in parallel. During each iterative step, several sub-problems are sampled from the current solution and processed in parallel by the neural model. The sub-problem that shows the largest improvement is then selected for optimization to update the current solution. The framework uses Kool et al. (2019)'s solver to solve sub-problems. Their approach integrates a destroy-and-repair mechanism to escape local optima, achieving superior speed and solution quality on instances up to 20,000 nodes. Similarly, Li et al. (2025) introduced DRHG, a Destroy-and-Repair using Hyper-Graphs framework that reduces problem complexity by representing solutions as hyper-graphs. A light encoder and heavy decoder predict how to reconnect isolated nodes, and an iterative destroy-and-repair process refines the solution. DRHG demonstrated competitive performance on TSP instances up to 10,000 nodes.

Wen et al. (2025) proposed LocalEscaper, a neural solver for large-scale TSP instances. Their method used a weakly-supervised learning (Weakly-SL) framework, combining SL and RL to improve low-quality training labels over time. To solve a TSP instance, LocalEscaper first generated an initial solution using a lightweight neural model with linear attention. It then applied an iterative improvement process consisting of subsequence reconstruction, 2-opt, and regional reconstruction. Subsequence reconstruction refined small segments of the tour, while regional reconstruction removed and reconnected edges in larger regions to escape local optima. The model progressively improved the solution while maintaining efficiency on large-scale instances (up to 50,000 nodes).

Generalization-focused transformer-based approaches Ensuring robust generalization in neural TSP solvers is crucial for handling unseen distributions, varying problem complexities, and different instance scales. Recent works have tackled generalization through state space reduction, leveraging problem symmetries, data augmentation, meta-learning, curriculum learning, adaptive training, and diffusion models. Drakulic et al. (2023) formulated TSP as a Markov Decision Process (MDP) and introduced Bisimulation Quotienting (BQ-MDP) to reduce the state space. BQ maps partial solutions to induced subproblems, identifying equivalent states and improving policy learning efficiency. This approach leverages TSP symmetries to enhance out-of-distribution robustness. The model generalized to TSP instances up to 1000 nodes. Kim et al. (2023) introduced Sym-NCO, a regularization-based training scheme that leverages rotational problem symmetricity in combinatorial optimization. This approach ensures solution invariance under transformations, improving generalization and convergence. Sym-NCO is model-agnostic, making it compatible with any encoder-decoder-based NCO framework. Implemented on POMO (Kwon et al. 2020), it outperformed NCO baselines in both greedy rollout and multi-start settings while achieving the fastest inference speed. On TSPLIB instances (< 250 cities), Sym-NCO surpassed POMO, achieving an average optimality gap of 1.62%, making it highly competitive among NCO solvers.

To tackle large-scale TSP instances, Luo et al. (2023) proposed a novel Light Encoder and Heavy Decoder (LEHD) model with a significant generalization capabilities. Unlike traditional transformer-based models that use a heavy encoder and light decoder, LEHD inverts this approach to dynamically capture the relationships between the current partial solution and all available nodes at each construction step through the use of a heavy decoder. Moreover, in order to create a more effective and reliable training process, they developed a training mechanism known as "learning to construct partial solutions," which teaches the LEHD how to construct optimal partial solutions of various sizes and random directions. This method acts as a form of data augmentation, thereby enriching the training set and leading to robust model performance. Furthermore, they introduced "learning to construct partial solutions", a training mechanism that improves generalization by teaching the model to build optimal sub-solutions in different directions. To enhance inference, they developed Random Re-Construct (RRC), which iteratively refines solutions within a computational budget. Zhang et al. (2022) introduced a hardness-adaptive generator, capable of creating TSP instances with increasing difficulty. The model was trained on progressively harder instances, improving generalization to complex distributions such as Gaussian mixture models. Zhou et al. (2023) introduced a generic meta-learning framework aimed at enhancing the generalization capabilities across different sizes and distributions of problems. Their approach leverages a second-order technique to enable effective learning of an initialized model with the capability of efficient adaptation to new tasks only using limited data during inference. By meta-training POMO (Kwon et al. 2020), the model achieved strong zero-shot generalization but suffered from high computational costs. Wang et al.

(2024) introduced Adaptive Staircase Policy (ASP), a universal neural solver designed to improve generalization across different TSP distributions and scales. ASP is model-agnostic and integrates game theory with curriculum learning for adaptive training. It consists of two key components: Distributional Exploration, which uses Policy Space Response Oracles (PSROs) to adapt to varying problem distributions, and Persistent Scale Adaption (PSA), which gradually increases problem complexity to enhance scalability. Comparative analysis reveals that ASP-enhanced models, specifically AM(ASP) and POMO(ASP), exhibited substantial improvements in performance of 90.9% and 49.4% reductions in the optimality gap compared to their respective base models, AM (Kool et al. 2019) and POMO (Kwon et al. 2020), without a notable increase in computation time. Notably, POMO(ASP) achieved the best results among all solvers.

Built on top of POMO, Zhou et al. (2024) proposed the Instance-Conditioned Adaptation Model (ICAM), an RL-based model designed to generalize across TSP instances of varying scales. The model incorporates instance-conditioned information (e.g., instance size and pairwise node distances) through a learnable adaptation function. This function is embedded in both the encoder and decoder via an Adaptation Attention-Free Module (AAFM), which replaces traditional attention for efficiency. A three-stage training scheme enables the model to progressively learn from small to large-scale instances without labeled data. ICAM achieved SOTA performance among RL-based constructive methods on TSP instances with up to 1,000 nodes. However, on large-scale TSP instances (>1000 nodes), its performance was worse than the SL-based methods, BQ (Drakulic et al. 2023) and LEHD (Luo et al. 2023).

Luo et al. (2024) proposed Self-Improved Learning (SIL) to enhance the scalability of Transformer-based approaches for solving large-scale TSP instances. Unlike existing methods that require labeled data or struggle with high computational complexity, SIL introduces an iterative self-improvement mechanism that enables direct training on large-scale instances. A local reconstruction approach continuously refines solutions, generating high-quality pseudo-labels to guide training. Additionally, SIL incorporates a linear complexity attention mechanism, significantly reducing computational costs compared to traditional Transformer-based models. Experimental results on TSP instances up to 100K nodes demonstrate that SIL outperforms state-of-the-art learning-based methods in both efficiency and solution quality. However, as problem sizes increase, the iterative local reconstruction process requires multiple decoding passes, making SIL computationally expensive for extremely large instances.

Wang et al. (2025) developed DEITSP, a diffusion-based non-autoregressive solver that generates initial TSP solutions by denoising noisy representations. By iteratively adjusting noise schedules, the model balances exploration and exploitation. While evaluated on real-world maps (USA, Japan, Burma), its reliance on SL-based training makes it dependent on high-quality labeled solutions which could limit its ability to generalize to unseen distributions and increase training data costs.

Table 4 provide a summary of the Transformer-based approaches with the advantages and disadvantages of the reviewed papers in this section.

Table 4 Summary of Transformer-based approaches and their advantages and disadvantages

References	Solution	Performance measure	Advantages	Disadvantages
Deudon et al. (2018)	Transformer-based network trained by Actor-Critic RL+ refined with 2-Opt heuristic	**Model+2-opt:** TSP20: 0% TSP50: 1.58% TSP100 (trained on TSP50): 5.02%	Proposes attention-based models based on the Transformer architecture	Small TSP instances (up to TSP100). Performance depends on the 2-opt heuristic
Kool et al. (2019)	An attention-based RL model with transformer-based encoder and self-attention-based	TSP20: 0.08% TSP50: 0.52% TSP100: 2.26%	Substitutes RNN structures with attention modules; improves performance on TSP and CVRP	Small TSP instances (up to 100 nodes); poor generalization ability
Kwon et al. (2020)	A policy optimization with multiple optima framework using a modified REINFORCE algorithm	TSP20: 0.0% (3 s) TSP50: 0.03% (16 s) TSP100: 0.14% (1 m)	Excellent runtime and energy performance	Limited scalability to small-sized problems
Bresson and Laurent (2021)	Transformer-based architecture trained by RL	TSP50: 0.004% TSP100: 0.39%	Uses batch normalization for stability and performance	Evaluated only on small instances (TSP50, TSP100)
Wu et al. (2022)	Deep RL framework to learn improvement heuristics	**Step limit T = 5,000:** TSP20: 0% (1 h) TSP50: 0.20% (1.5h) TSP100: 1.42% (2 h) TSP200: 6.86% (30 m) **TSPLIB (up to 300 cities):** Avg. opt. gap: 4.70%	Learns 2-opt strategies via DRL	Works only for small instances
Xin et al. (2021a)	Multi-Decoder Attention Model with Embedding Glimps	**MDAM (beam search 50):** TSP20: 0% (3 m) TSP50: 0.03% (14 m) TSP100: 0.38% (44 m)	Uses multiple decoders for diversity; enhances embeddings	Poor scalability to larger problems
Wang et al. (2023)	BERT-based deep RL for graph embeddings	TSP20: 0% (0 s) TSP50: 0.53% (1 s) TSP100: 0.64% (3 s)	Effective representation of graphs with BERT	High complexity and parameter count
Ma et al. (2021)	DACT: Dual-Aspect Transformer trained with proximal policy optimization	**DACT×4 augment:** TSP20: 0% (10 m) TSP50: 0% (1 h) TSP100: 0.09% (2.5h) **TSPLIB (up to 200 cities):** Avg. opt. gap: 2.07%	Avoids noise with separate embeddings; circularity and symmetry capture	Resource-intensive, limited to small instances

Table 4 (continued)

References	Solution	Performance measure	Advantages	Disadvantages
Xu et al. (2022)	Attention-based RL model with dynamic-aware context embedding	TSP20: 0.26% (0.37s) TSP50: 1.23% (0.91s) TSP100: 3.74% (2 s)	Utilizes batch normalization reordering and gate aggregation to enhance node embeddings, leading to better context embedding and improved performance. Demonstrates better sample efficiency and faster learning compared to existing attention-based RL models	Exhibits poor solution quality even for small instances
Sun et al. (2024)	LCH-Regret method	**POMO-Regret-8 aug:** TSP50: 0.02% (15 s) TSP100: 0.11% (88 s) **TSPLIB instances:** Avg. opt. gap: 4.45%	Enhances generalization ability	LCH-Regret model slightly increases running time due to the maintenance of the Regret mask
Grinsztajn et al. (2023)	Poppy: A population-based RL approach based on the AM model (Kool et al. 2019)	**Poppy (population size: 16):** TSP100: 0.07% (1 m) TSP125: 0.14% (10 s) TSP150: 0.27% (20 s)	Trains a population of complementary policies, improving the exploration of the solution space and overall performance	Primarily evaluated on instances with up to 100 nodes for TSP
Yang et al. (2023)	Memory-efficient Transformer with scaled dot-product attention	**Tsformer (beam search):** TSP20: 0.26% TSP1000: 16.87% **TSPLIB (up to 280 cities):** Avg. opt. gap: 4.71%	Reduces complexity; effective on larger TSPs (up to 1000 nodes)	Minor performance drop, particularly in optimality gap
Jin et al. (2023)	Pointerformer: DRL approach based on multi-pointer transformer	TSP100: 0.16% (52.34s) TSP200: 0.68% (5.54s) TSP500: 3.56% (59.35s) **TSPLIB:** (100 cities): 1.33% (20 s) (101-500 cities): 5.43% (0.46s) (501-1002 cities): 18.20% (5.14s)	Utilizes a reversible residual network in the encoder to significantly reduce memory consumption. Employs feature augmentation and an enhanced context embedding approach to improve solution quality. Capable of handling TSP instances with up to 500 nodes	It can scale up to 500 nodes, but may struggle with instances larger than this, limiting its applicability to very large TSP problems
Cheng et al. (2023)	Selector-Optimizer Transformer for sub-problems	**Ours-mixed:** TSP200: 0.6362% TSP2000: 2.7577%	Scales to large problems (20,000 nodes)	Prone to local minima, sub-problem reliance

Table 4 (continued)

References	Solution	Performance measure	Advantages	Disadvantages
Li et al. (2025)	Hyper-Graph Representation, where consecutive edges are grouped into hyper-edges. A Transformer-based model predicts how to reconnect isolated nodes	TSP10K: 1.33% (3.70m)	The hyper-graph representation reduces computational complexity, making it suitable for large-scale routing problems	The destruction phase relies on clustering-based removal, which may restrict solution diversity and limit exploration capability
Wen et al. (2025)	Uses a lightweight NN with Linear Attention to construct initial solutions. Applies Subsequence Reconstruction, 2-opt, and Regional Reconstruction to refine solutions iteratively	TSP10K: 1.69% (16.07m) TSP50K 4.46% (3.19h)	Reduces reliance on high-quality labels with weakly-SL. Efficient and memory-friendly due to linear attention	Iterative improvement may slow down inference for real-time applications
Drakulic et al. (2023)	Transformer-based policy network for the BQ-MDPs	**BQ-transformer bs16:** TSP100: 0.01% (32 m) TSP200: 0.09% (3 m) TSP500: 0.55% (15 m) TSP1000: 1.38% (38 m)	Utilizes Bisimulation Quotienting to reduce the state space, leading to more efficient learning and generalization	Relies on imitation learning from (near) optimal solutions for small instances, which can be resource-intensive to generate

Kim et al. (2023)	Sym-NCO: A regularization-based training scheme leveraging symmetricities, trained with REINFORCE	**Sym-NCO (sampling 100, with multi-start rollout):**
		TSP100: 0.39% (13 s)
		TSPLIB ($n < 250$ cities, pre-trained model POMO TSP100):
		Avg. opt. gap: 1.62%
			Introduces a regularizer-based training scheme, enhancing the performance of existing DRL-NCO methods without needing expert domain knowledge	Performance evaluation is mainly conducted on problems with up to 100 nodes

Luo et al. (2023)	LEHD model with Random Re-Construct (RRC)	**LEHD + RRC (1000 iterations):** TSP100: 0.0016% (2.2h) TSP500: 0.167% (1.2h) **TSPLIB (up to 4461 nodes):** Avg. opt. gap: 1.529%	Dynamic decoder captures node relationships; iterative improvements	High computational cost, relies on SL
Zhou et al. (2023)	Meta-learning framework leveraging a second-order technique	**TSPLIB (100-1002 nodes):** Avg. opt. gap: 6.55%	Improves the generalization capabilities of DL methods, allowing models to adapt to new tasks with limited data during inference. The second-order technique offers rapid and effective learning	The initial use of second-order derivatives introduces significant computational overhead, particularly for training on large instance sizes

Table 4 (continued)

References	Solution	Performance measure	Advantages	Disadvantages
Wang et al. (2024)	ASP: PSRO-based training + Adaptive Staircase-based curriculum	**POMO(ASP):** TSP20: 0.09% (0.45s) TSP50: 0.59% (1.44s) TSP100: 1.67% (5.67s) **TSPLIB (up to 299 nodes):** Avg. opt. gap: 3.23%	Incorporates game theory and curriculum learning. Introduces Adaptive Staircase Policy Space Response Oracle (ASP) to address generalization issues and Persistent Scale Adaption for performance across various problem scales	Performance evaluation is mainly conducted on problems with up to 100 nodes
Zhou et al. (2024)	Transformer-based RL (POMO) with instance adaptation	TSP200: 0.326% (3 s) TSP500: 0.771% (38 s) TSP1000: 1.581% (3.8m)	Lightweight AAFM improves training and inference speed. Combines instance scale and distance for better decision-making	Outperformed by SL-based models on large TSP instances/
Luo et al. (2024)	Iterative local reconstruction method with Linear complexity attention mechanism, trained using a combination of RL and self-improvement	TSP10K: 2.00% TSP100K: 2.95%	Linear attention reduces computational overhead. Self-improved learning does not require labeled data	Heavily depends on iterative improvement cycles to refine the solution, which can slow convergence
Wang et al. (2025)	A dual-modality graph transformer extracts node and edge features. Iterative noise adjustment and scheduling refine the solution	TSP1000: 3.68% (41.83m). **Real-world instances:** Japan (100 cities): 0.85% (4.03m) Burma (100 cities): 2.30% (3.99m) USA (100 cities): 1.89% (4.11m) TSBLIB (<200 cities): 0.78% (8.51s)	Fast inference with one-step diffusion. Strong generalization to real-world instances	Dependence on high-quality labels

3.1.4 Multi-model DL approaches

Multi-model DL-based approaches combine different architectures, such as GNNs and Transformers, to benefit from their complementary capabilities. Sui et al. (2021) introduced Neural-3-OPT, a DRL-based 3-OPT heuristic for TSP. The model follows an encoder-decoder architecture with an actor-critic training process. The encoder combines a sparse graph convolution network and a bidirectional LSTM to generate tour embeddings. The decoder consists of a pointer network for selecting three links to remove and a Feature-wise Linear Modulation Network (FiLM-Net) for determining the optimal reconnection among seven possible types. This structured decision-making process improves tour quality with each iteration. Neural-3-OPT outperformed previous neural 2-OPT heuristics (da Costa et al. 2021). However, it requires higher computational time due to its increased complexity in link selection and reconnection. Similarly, Ma et al. (2019) introduced the Graph Pointer Network (GPN), which extends pointer networks with graph embedding layers to improve generalization and convergence for large-scale TSP. It achieved shorter tours and faster computation than previous RL-based methods and explored hierarchical RL for complex variants like TSP with time windows. Building on this, Wang and Chen (2022) proposed DGCM, incorporating GNNs, Conv-LSTM, and Dynamic Positional Encoding (DPE) to enhance node selection and translation invariance. Trained on TSP50, it generalized to larger instances, surpassing GPN in solution quality. Stohy et al. (2021) further advanced this line of research with Hybrid Pointer Network (HPN), integrating GPN and transformer encoders to improve efficiency and effectiveness across small and large TSP instances.

To tackle large-scale TSP instances, Pan et al. (2023) proposed H-TSP, a hierarchical RL framework. The model consists of two levels: an upper-level policy, which selects a subset of nodes (<200), and a lower-level policy, which connects them to an existing partial tour. The upper-level model is trained with Proximal Policy Optimization (PPO) (Schulman et al. 2017), while the lower-level model uses REINFORCE with a shared baseline. H-TSP achieved competitive results on instances up to 10,000 nodes with computation times under 4 s. Performance improves when replacing the lower-level model with LKH-3 during inference, though at the cost of higher computational time. Ye et al. (2024) proposed GLOP (Global and Local Optimization Policies), a neural solver for large-scale routing problems. GLOP partitions large instances into smaller TSPs and further decomposes them into shortest Hamiltonian path problems. It combines non-autoregressive global partitioning with autoregressive local construction, integrating GNNs and attention mechanisms in an end-to-end NCO framework. GLOP achieved state-of-the-art performance on large-scale TSP, becoming the first neural solver to scale to TSP100K, with an optimality gap of 5.1% and a 174× speed-up over LKH3.

Zheng et al. (2024) proposed a Unified neural Divide-and-Conquer framework (UDC) to address large-scale combinatorial optimization problems. The framework employs a GNN-based approach to divide the TSP instance into sub-problems and utilizes a constructive transformer-based solver (Zhou et al. 2024) to optimize these sub-problems iteratively. A key contribution is the Divide-Conquer-Reunion (DCR) training method, which refines the dividing policy by considering its impact on the final solution. Unlike previous methods that train the dividing and conquering policies separately, UDC jointly optimizes both, preventing the model from adapting to sub-optimal sub-solutions. Extensive experiments on large-scale TSP instances demonstrated that UDC significantly improves both efficiency and solution quality compared to existing DL-based solvers.

Jung et al. (2024) introduced a lightweight CNN-Transformer model for TSP, designed to reduce computational complexity and GPU memory usage in transformer-based approaches. The model integrates a CNN embedding layer to extract local spatial features and employs partial self-attention, limiting attention to recently visited nodes. This modification reduces the overhead of fully connected transformer architectures. Experiments showed that the model consumes 20% less GPU memory than other state-of-the-art transformer-based solvers, while maintaining competitive performance.

Table 5 provide a summary of the Multi-model DL-based approaches, with the advantages and disadvantages of the reviewed papers in this section.

Table 5 Summary of multi-model DL-based approaches and their advantages and disadvantages

References	Solution	Performance measure	Advantages	Disadvantages
Sui et al. (2021)	Ptr-Net + GCN to Learn 3-opt Heuristic via Deep RL	**Sample size: 2,000** TSP50: 0.08% TSP100: 0.74% **TSPLIB (up to 280 cities):** Avg optimality gap 2.93%	It is more powerful and efficient than the learned 2-opt heuristic	The running time is higher compared to Learned 2-OPT, especially for larger instances
Ma et al. (2019)	A graph pointer network (GPN) based on the pointer network trained using RL	**TSPLIB (up to 1500 cities):** Avg optimality gap $9.35 \pm 3.45\%$	Employs hierarchical GPNs to handle constrained TSP problems, improving solution quality under constraints. Demonstrates strong generalization from small-scale TSP problems to larger problems	May produce infeasible solutions
Wang and Chen (2022)	Dynamic Graph Convolutional-LSTM (Conv-LSTM) model with Dynamic Positional Encoding (DPE)	**DGCM (2-opt):** TSP100: 0.38% (16 s) TSP500: 8.1450% (847 s) TSP1000: 9.9091% (3964 s) **TSPLIB (50-200 cities, trained on TSP50):** Avg. opt. gap: 0.76%	Combines Conv-LSTM with GNNs to capture both spatial and temporal features, enhancing the encoding of node information	Poor generalization ability
Stohy et al. (2021)	Based on GPN: Graph embedding layer + transformer encoder layer	**HPN+2opt:** TSP500: 8.5115% (1460 s) TSP1000: 9.0869% (6480 s)	Outperforms the original Graph Pointer Network (GPN)	Struggles to find optimal solutions for large instances, indicating limits in scalability
Pan et al. (2023)	The upper-level model: CNN, and The lower-level model: a transformer	TSP1K: 6.62% (0.33s) TSP2K: 7.39% (0.72s) TSP5K: 7.10% (1.66s) TSP10K: 7.32% (3.32s)	It can efficiently handle large-scale TSP instances with up to 10,000 nodes. It reduces computation time compared to state-of-the-art (SOTA) methods	H-TSP's solution quality has a gap compared to some SOTA methods, particularly for larger instances
Ye et al. (2024)	GNN and Attention-based approach: Hybridizing NAR and AR end-to-end NCO paradigms	TSP1K: 3.11% (3.0m) TSP10K: 4.90% (1.8m) TSP100K: 5.10% (2.8m) **TSPLIB (up to 1000 cities):** Avg. opt. gap: 0.69%	The first neural solver to adeptly scale to TSP100K, achieving an optimality gap of 5.1% and a speed-up factor of 174 times compared to LKH3 algorithm. Combines non-autoregressive global partitioning with autoregressive local construction, leveraging the strengths of both paradigms	Decomposition strategies might overlook the relationships between sub-problems, which affect the overall solution quality

Table 5 (continued)

References	Solution	Performance measure	Advantages	Disadvantages
Zheng et al. (2024)	Dividing policy: A heatmap-based solver with Anisotropic GNN (Sun and Yang 2023). Conquering policy: ICAM (Zhou et al. 2024)	TSP1K: 1.78% (8 m) TSP10K: 4.03% (1 m) TSP100K: 4.05% (11.5m) TSPLib,(500,1,000] **TSPLIB (500-1000):** Avg. opt. gap: 6.0% **TSPLIB (1000-5000):** Avg. opt. gap: 7.6%	Reunion stage mitigates issues that arise from sub-optimal division	Less Effective on TSPLIB instances
Jung et al. (2024)	A lightweight CNN-Transformer model based on a CNN embedding layer and partial self-attention	With Beam search (B=2500): TSP100: 1.11% **TSPLIB (51 to 150 cities):** Avg. opt. gap: 3.81%	Substantially lowers GPU memory usage and shortens inference time through the use of partial self-attention in the decoder	Solves only small size instances

3.1.5 Hybrid DL and heuristic-based approaches

Hybrid DL-Heuristic approaches integrate DL with classical heuristics like LKH or GLS to guide or enhance the search process.

GNN + heuristic search integration: These approaches integrate GNNs with heuristic search algorithms such as Guided Local Search (GLS) (Voudouris et al. 2010a), Monte Carlo Tree Search (MCTS) (Chaslot et al. 2008), LKH (Helsgaun 2000), and Ising models (Cipra 1987). These methods leverage GNNs to enhance heuristic search with enriched data insights or iteratively employ neural networks to refine decision-making within search strategies (Sui et al. 2023).

Xing and Tu (2020) combined the GNN with MCTS. The GNN identifies graph motifs and interactions between vertices to generate prior probabilities for vertex selection. These probabilities inform the MCTS, guiding it towards more promising solutions. Experimental results show that the GNN-MCTS framework generalizes well to larger instances (up to 1000 nodes) despite being trained on small instances.

Xin et al. (2021b) introduced NeuroLKH, which integrates a Sparse Graph Network (SGN) with LKH to enhance efficiency in large-scale TSP instances. It combines SL for edge scores and UL for node penalties. Trained on instances with up to 500 nodes, it outperformed LKH and VSR-LKH (Zheng et al. 2023), achieving minimal optimality gaps. It demonstrated strong generalization to instances with up to 5000 nodes and performed well on 72 TSPLIB instances. The NeuroLKH_M variant, trained on diverse node distributions, further improved generalization, surpassing LKH while slightly underperforming VSR-LKH. Wang et al. (2023) proposed a Self-Supervised RL (SSRL) framework that integrates self-supervised learning with the LKH heuristic. The approach combines the advantages of VSR-LKH (Zheng et al. 2021) and NeuroLKH (Xin et al. 2021b) while addressing their limitations. The model employs a GNN with a dual-decoder structure. The edge decoder learns edge scores using self-SL, while the node decoder refines node penalties through RL. This method replicates and enhances key LKH components without expert-designed rules. The authors introduce Self-Supervised Q-Learning (SQ) and Self-Supervised Actor-Critic

(ACS) to improve learning stability and efficiency. The results show that SSRL outperforms LKH, VSR-LKH, and NeuroLKH on large-scale instances, achieving higher solution quality, better generalization, and improved computational efficiency.

Li et al. (2023) proposed a method that enhances the generalization of pre-trained GCN models by combining them with the LKH heuristic. They tackled large instances by dividing them into smaller sub-problems, each containing a fixed number of cities corresponding to the GCN's training size, and then integrating the sub-solutions using an attention-based merging mechanism. The merged probability heatmap was used to construct an edge candidate set for the LKH algorithm. Experiments demonstrated that their approach significantly enhances the generalization ability of the GCN, outperforming LKH, NeuroLKH, VSR-LKH, and Att-GCN+MCTS (Fu et al. 2021), across problem sizes ranging from 20 to 10K cities, all within the same time constraints.

Hudson et al. (2022) combined GAT and GLS to solve the TSP. The model uses GATs to estimate global regret for each edge, guiding GLS toward high-quality solutions. It learns a regret approximation model that distinguishes between costly and optimal edges. The approach significantly reduces optimality gaps for 50 and 100-node instances and demonstrates strong generalization to larger and TSPLIB instances. This generalization is attributed to using the line graph as input, which focuses on edge weights rather than node positions. However, it requires high GPU memory and relies on SL, suggesting potential improvements with end-to-end training. Sui et al. (2023) extended this work by proposing NeuralGLS, which replaces GAT with a self-adaptive GCN. Their method operates on a sparse k-nearest neighbor graph, reducing computational overhead while maintaining high solution quality. Both methods enhance traditional local search with learned heuristics.

GNN + meta-heuristics: Ye et al. (2023) introduced DeepACO, a GNN-based metaheuristic that integrates learned heuristics into the Ant Colony Optimization (ACO). Their framework consists of two stages: a GNN learns problem-specific heuristic maps across instances; and ACO uses these learned measures to guide search and local optimization. DeepACO can also incorporate Neural-guided Local Search (NLS) to escape local optima by alternating between conventional and heuristic-driven refinements. Their model outperformed classical ACO variants and performed competitively with DL-based approaches such as (Sun and Yang 2023) and (Cheng et al. 2023) on RUE instances with up to 1000 nodes.

GNN + DP: Kool et al. (2022) proposed Deep Policy Dynamic Programming (DPDP), a hybrid framework that integrates DP with GNN guidance to solve routing problems including the TSP. DPDP uses the GNN to generate a heatmap that scores edges and guides a beam-based DP algorithm which incrementally builds partial solutions. It enhances DP by using the learned scores to restrict the state space, selecting only the top-scoring partial solutions and pruning dominated ones. The model combines heat values with a potential function to estimate the quality of incomplete solutions. DPDP achieved near-optimal performance on TSP instances with 100 nodes.

Decomposition-based hybrid approaches: Sanyal and Roy (2022) proposed Neuro-Ising, a GNN-based framework that integrates the Ising model to solve large-scale TSP. The model clusters TSP instances into smaller sub-problems, which are solved separately using Ising solvers. These solvers optimize path inclusion to minimize total travel distance. A GNN aggregates sub-solutions into a near-optimal global tour. Neuro-Ising achieved a

396x speedup over Concorde on 15 TSPLIB benchmarks, with a 20% average degradation in solution quality.

Fu et al. (2021) proposed an Attention Graph Convolutional Residual Network (Att-GCRN) to efficiently generalize small-scale TSP models to large instances. The model generates scalable heatmaps using graph sampling, where the TSP graph is divided into subgraphs of fixed size (m=20 or 50) before merging. A MCTS module refines the solutions, maintaining the advantages of SL without requiring retraining for different problem sizes. The approach achieved near-optimal solutions across various TSP sizes, with an optimality gap of 4.39% for 10,000 nodes, producing results comparable to LKH3.

Chen et al. (2023) proposed a fine-grained divide-and-conquer approach for Extending NCO (ExtNCO) to solve large-scale TSP. Their approach consists of three procedures: dividing, solving and merging. First, a noval Locality-based K-Means (LocKMeans) algorithm is proposed to divide the large TSP instance into smaller non-overlapping sub-problems. This algorithm operates with linear time complexity and is based on node density distribution. Then, each sub-problem is solved independently using a NCO solver like POMO model (Kwon et al. 2020), which was pre-trained on TSP-100 instances. This allows for parallel processing to speed up the solution time without compromising on the quality of the sub-tour solutions. Finally, the sub-tours are merged into a complete tour using a minimum spanning tree (MST-based) strategy. Enhancements such as the K-OPT heuristic are incorporated to locally optimize the merged tour which improve the overall solution quality. Experiments on random euclidean, TSPLIB, and VLSI instances (1K to 1 M nodes) showed that ExtNCO outperformed state-of-the-art baselines like H-TSP (Pan et al. 2023), achieving an optimality gap below 6% for TSP-1 M while maintaining near-linear time scalability.

Liu et al. (2025) proposed UNiCS, a neural-guided solver that combined local search (LKH) and population-based search (EAX) in a two-stage framework. The first stage used LKH to find high-quality solutions quickly. The second stage applied EAX to improve the solutions further. A SGN guided both stages through the Unified Neural Guidance (UNG) module. UNG predicted edge scores to help the search process and decided when to switch from LKH to EAX based on problem size. It showed strong generalization capabilities to challenging TSP benchmarks (TSBLib, VLSI, and National) containing large instances (10,000- 71,009 nodes) with diverse node distributions across different runtime budgets.

Transformer + heuristic search integration: Tian et al. (2024) proposed a DRL-assisted improvement-based approach that enhanced the Guided Fast Local Search (GFLS) (Voudouris et al. 2010b) with a Transformer model. GFLS improves 2-opt by penalizing overused edges to escape local optima. The Transformer replaced the hand-crafted rule for selecting 2-opt endpoints and was trained using an actor-critic RL algorithm. This reduced the number of unnecessary trials by directly predicting promising city pairs. Their method achieved near-zero optimality gaps on TSP instances up to 100 nodes and generalizes well to TSPLIB instances with up to 300 nodes.

Table 6 provide a summary of the hybrid DL and heuristic-based approaches, with the advantages and disadvantages of the reviewed papers in this section.

3.1.6 Other DL-based approaches

Miki et al. (2018) proposed algorithms that use CNNs to learn the optimal TSP tour represented as an image, termed the Good-Edge Distribution. This distribution represents the

Table 6 Summary of hybrid DL and heuristic-based approaches and their advantages and disadvantages

References	Solution	Performance measure	Advantages	Disadvantages
Xing and Tu (2020)	GNN and Monte Carlo Tree Search	**RUE instances:** TSP20: 0.01% TSP50: 0.20% TSP100: 1.44% **Clustered instances:** TSP20: 0.03% TSP50: 0.44% TSP100: 1.44%	Good generalization to larger instances even when trained on smaller ones. MCTS refines decisions with scouting exploration, enhancing reliability of the policy	Slower due to MCTS simulation phase
Xin et al. (2021b)	Sparse Graph Network (SGN) and Lin-Kernighan-Helsgaun (LKH) Heuristic	TSP100: 0 TSP200: 0 TSP500: 0.010 **Gaps with respect to the best value (LKH, VSR-LKH):** TSP1000: 0% (938 s) TSP2000: 0% (25613 s) TSP5000: 0.010% (103885 s)	Captures a general pattern across a class of instances. Demonstrates outstanding generalization capabilities. Significantly improves solution quality over traditional LKH	Expensive ground truth required for SL
Wang et al. (2023)	GNN-based model, SSRL with LKH	**Objective Values:** TSP100: 7.753 TSP200: 10.701 TSP500: 16.541837 **8 TSPLIB instances (150- 5934 cities):** finds optimal solutions in an average of 7.5 out of 10 runs	Enhances LKH by learning node penalties and edge scores	High computational needs
Li et al. (2023)	Partition-based GCN prediction + attention-guided merging + LKH	TSP2K: 0.0005%. TSP5K: 0.0009%. TSP10K: 0.0023%	Outperforms the LKH, NeuroLKH, VSR-LKH for problem sizes ranging from 20- 10000 under the same time limit	The computation of the optimality gap relies on the shortest tour generated by all tested methods rather than the actual optimal solution
Hudson et al. (2022)	Graph Attention Network (GAT) and Guided Local Search (GLS)	TSP20: 0% TSP50: 0.009±0.069% TSP100: 0.698±0.801% **TSPLIB (50-200 cities, trained on TSP100):** Mean opt. gap: 1.529±1.328%	Combining GNN with GLS achieves a significant reduction in the optimality gap. The global regret measure to guide local search allows for high-quality solutions to be found quickly	Requires increased GPU memory
Sui et al. (2023)	Self-Adaptive GCN and GLS	**RUE instances:** TSP20: 0% TSP50: 0.003% TSP100: 0.470% **TSPLIB (50-200 cities, trained on TSP100):** Mean opt. gap: 1.529±1.328%	Superior generalization on both real-world and larger-scale TSP instances	Cumbersome for large instances

Table 6 (continued)

References	Solution	Performance measure	Advantages	Disadvantages
Ye et al. (2023)	GNN + ACO (meta-heuristic)	**DeepACO (T = 10):** TSP500: 1.84% (10 s). TSP1000: 3.16% (32 s)	Improves ACO efficiency and solution quality	Fails to produce high-quality solutions without local search refinements
Kool et al. (2022)	GNN + DP	TSP100: 0.004% (10 m heatmap + 2h35m DP)	Enhances DP by learning to restrict the state space	Limited to 100 nodes
Sanyal and Roy (2022)	GNN with LKH heuristic	**TSPLIB (200-5000 cities):** 396× Speedup vs. Concorde	Achieves significant speedup over Concorde for large-scale TSPs, handling up to 5000 cities. Utilizes parallel execution of Ising solvers for clusters	20% degradation in solution quality on average vs. Concorde
Fu et al. (2021)	Att-GCN+MCTS: Attention Graph Convolutional Residual Network and RL approach: Monte Carlo tree search	TSP20: 0.0% TSP50: 0.0145% TSP100: 0.0370% TSP200: 0.8844% TSP500: 2.5365% TSP1000: 3.2238% TSP10,000: 4.3902%	Significantly improves the generalization ability of a small pre-trained model to handle large-scale TSP instances. Effective on instances with up to 10,000 vertices	Generating and merging heat maps, along with running MCTS, introduces additional computational overhead
Chen et al. (2023)	Dividing: K-means-based dividing algorithm + NCO Solver: POMO pre-trained model + Merging: MST-based merging strategy	**For ExtNCO-Qlt: REI:** TSP10K: 5.62% TSP20K: 5.77% TSP50K: 5.84% TSP100K up to TSP1M: avg. opt. gap: 5.96% **VLSI (100K to 744K nodes):** Avg. opt. gap: 8.325%	Highlights the efficiency of the divide-and-conquer strategy when combined with advanced NCO techniques. Handles TSP instances ranging from 1,000 to 1 million nodes	Requires significant computational resources, particularly GPU capabilities
Liu et al. (2025)	Uses SGN to guide local search (LKH) and population-based search (EAX)	Outperformed LKH-based and hybrid methods across different runtime budgets (1800s, 3600 s, 7200 s)	Leverages LKH's fast convergence in early stages and EAX's strong exploration in later stages. Demonstrates strong real-world applicability beyond synthetic benchmarks	Phase transition policy (which is based on instance size) may not always be optimal
Tian et al. (2024)	Transformer+ GFLS	TSP50: 0.00% (2.5h). TSP100: 0.01% (5 h) **TSPLIB (up to 300):** Avg: 0.61%	Near-optimal performance on RUE instances	it is computationally intensive

likelihood of each edge being part of the optimal tour and used instead of the traditional distance measures to construct tours and guide the selection of edges with high Good-Edge Values. The approach is implemented in two heuristic strategies: EV-greedy and EV-2opt, which are based on greedy algorithm and the conventional 2-opt search, respectively. Their model treats the problem and its solution as images which allow the CNN to learn patterns that might not be evident through traditional data formats. The experimental results demonstrate that the EV-2opt+2opt method, which combines the predictive power of DL with the refinement capabilities of the 2-opt local search, showed notably lower optimality gaps compared to other heuristic methods such as EV-greedy and basic 2-opt. However, for large problem sizes where the image has a high density of vertices, the CNN often outputs large values even for non-optimal edges, reducing the model's accuracy. Additionally, the accuracy of the solution heavily relies on the search algorithm that utilizes these evaluations.

To improve the performance of CNN, Miki and Ebara (2019) proposed a model using CNN for solving TSP, called Pixel-mapped Classification Network (PCN). Rather than outputting a complete tour at once, this method makes a tour by repeatedly selecting a vertex and adding it to the end of the partial path. This iterative approach involves feeding the current state of the solution (partially completed path) back into the CNN, the CNN evaluates all possible next vertices and suggests the most suitable one to add based on the learned features. This selection process continues until the tour is complete. The beam search is applied to improve the solution quality. The model was trained on small instances (10-100 nodes) and tested on instances up to 400 nodes. Results showed that the error ratio increases with problem size, but higher beam search thresholds (β) improve accuracy at the cost of increased computation time.

Mele et al. (2021) introduced an ML-Constructive heuristic for large-scale TSP, combining a ResNet model (He et al. 2016) with the Clarke-Wright (CW) heuristic (Clarke and Wright 1964). The approach reduces the search space by using ML to initialize Candidate Lists (CLs), improving generalization. It follows a two-phase process: the ResNet model constructs partial solutions by identifying patterns in CLs, while the CW heuristic completes the tour by connecting remaining vertices. Trained on small instances (up to 300 nodes) and tested on larger instances (up to 1748 cities), the model outperformed traditional heuristics like multi-fragment, CW, and furthest insertion. Despite its high-quality solutions, the approach requires significant computation time due to the DL model's complexity.

Ling et al. (2023) proposed a DRL-based model using deep CNNs (DCNNs) to solve the TSP in real time. The model transforms each TSP instance into a sequence of image-based sub-problems, where each step predicts the next city to visit using an image classification process. Each sub-problem is represented as a 40×40 image, which is processed by a modified VGG16 DCNN trained through RL without labeled data. A rule-based filtering mechanism is used to remove infeasible actions and ensure tour validity. The approach achieves near-optimal and fast solution. However, its scalability is limited by the fixed image resolution and sparse pixel encoding.

Table 7 provides a summary of the other DL-based approaches, including their performance measures, along with the advantages and disadvantages of the reviewed papers in this section.

3.2 Traditional ML-based approaches

Vitali et al. (2022) improved the ML-Constructive heuristic of Mele et al. (2021) by reducing computational cost and execution time. They replaced the ResNet model with simpler ML models and integrated Delaunay triangulation for faster candidate list creation. A third phase, a 2-opt local search, was introduced to refine solutions by optimizing edges selected in the heuristic phase. Experiments on TSPLIB instances (up to 1748 cities) showed that 2-opt heuristic significantly improved solution quality. The SVM+LS model outperformed

Table 7 Summary of other DL approaches and their advantages and disadvantages

References	Solution	Performance measure	Advantages	Disadvantages
Miki et al. (2018)	CNN of Good-Edge Distribution trained in a SL manner	**TSPLIB (51 to 200 cities trained on random instances of 20-100 nodes):** Avg. opt. gap: 1.568	The introduction of Good-Edge Distribution helps in identifying promising edges that could be part of the optimal tour	Training the CNN with SL requires a large number of instances and significant computational resources
Miki and Ebara (2019)	CNN-based: Pixel-mapped Classification Network (PCN)	**Random instances (20 to 400 nodes, trained on instances of 10-100 nodes):** Avg. opt. gap: 0.979%	Utilizes CNNs to treat the TSP as an image classification problem, enabling the model to capture spatial features effectively. Achieves high classification accuracy (over 95%) with SL	The CNN tends to output higher values for non-optimal edges in areas with a high density of vertices, which can reduce solution accuracy
Mele et al. (2021)	Simple ResNet model and Clarke-Wright heuristic	**TSPLIB (100 to 1748 cities):** Avg. opt. gap: 4.374%	The use of Candidate Lists significantly reduces the computational burden by restricting the search space during solution creation	The heuristic exhibits an expensive constant computation time due to the training phase of the ML model
Ling et al. (2023)	DRL + VGG16 Deep CNN	TSP50: 3.97%. TSP100: 6.33%	Real-time inference, suitable for real-time applications	Fixed image size and sparse input may limit scalability

the original ML-Constructive heuristic, improving 48 out of 54 instances in solution quality and efficiency.

By integrating RL with the LKH algorithm, Zheng et al. (2023) introduced two novel algorithms: the Variable Strategy Reinforced LKH (VSR-LKH) and VSR-LKH-3. These algorithms are aimed to address the inflexible traversal operation of LKH algorithms by learning to adaptively choose the appropriate edges to be added in the k-opt process based on RL.T VSR-LKH integrates Q-learning, Sarsa, and Monte Carlo methods with LKH, using a Q-value based on city distance and α-value for candidate selection. It outperformed LKH and NeuroLKH across 111 TSPLIB instances, including super large cases (85,900 cities). VSR-LKH-3 extends this framework to LKH-3, enabling solutions for TSP variants with constraints, such as TSP with time windows and Colored TSP (CTSP). It outperformed LKH-3 and state-of-the-art CTSP heuristics, yielding 12 new best solutions across 65 CTSP instances.

Fitzpatrick et al. (2021) integrated ML techniques as pre-processing step for graph sparsification, which was then followed by an exact Integer Programming method to address the TSP. This sparsification process utilized a binary classification model designed to identify and remove edges unlikely to be part of the optimal tour, thus significantly cutting down the number of decision variables. The features selected for this model were drawn from various sources, such as linear programming relaxation, cutting planes, Minimum-weight Spanning Tree (MST) heuristics, and other local and statistical characteristics of the graph. The results demonstrate that this approach can effectively prune about 85% of the variables in TSP instances from TSPLIB/MATILDA[1]while maintaining most of the edges that form the optimal tour, thereby reducing computational complexity without greatly affecting the quality of the solution.

Table 8 provides a summary of the traditional ML approaches, with the advantages and disadvantages of the reviewed papers in this section.

3.3 Summary

Recent advancements in ML have introduced a variety of approaches to solve the TSP, each with its own architecture, strengths, and challenges. This section summarizes these approaches and outlines their main strengths and limitations.

Ptr-Nets (Vinyals et al. 2015) are neural architectures that use an encoder-decoder structure with attention mechanisms. Ptr-Nets select elements directly from the input sequence rather than producing outputs of a fixed size. Ptr-Net-based approaches achieved strong performance on small to medium TSP instances, particularly those with up to 200 nodes. However, their performance declines rapidly as the problem size increases.

GNN-based approaches represent a significant shift from Ptr-Nets by treating TSP instances as unordered graphs rather than sequences. GNNs have been used both to predict feasible solutions directly (Joshi et al. 2022) and to guide local search heuristics (Hudson et al. 2022). For small to medium-sized instances, GNNs work well because they excel at managing unordered data and capturing inter-node interactions. However, training GNNs on large graphs remains computationally demanding and time-consuming. To improve scalability, graph sparsification methods, such as k-nearest neighbors (Joshi et al. 2022) and 1-Tree heuristics (Lischka et al. 2023), can be used to reduce the graph size while preserving solution quality. Furthermore, anisotropic and attention-based GNNs, which can produce both node and edge embeddings, outperform isotropic GCNs in tasks requiring edge prediction (Sun and Yang 2023).

Transformer-based approaches offer a powerful alternative to recurrent and convolutional-based approaches for solving TSP. Their architecture relies on a multi-head self-attention mechanism within an encoder-decoder framework to model interactions between

[1] https://matilda.unimelb.edu.au/matilda/problems/opt/tt#tt.

Table 8 Summary of traditional ML approaches and their advantages and disadvantages

References	Solution	Performance measure	Advantages	Disadvantages
Vitali et al. (2022)	SVM + Clarke-Wright heuristic + 2-opt local search	**TSPLIB (100 to 1748 cities):** Avg. opt. gap: 5.56%	Replacing the ResNet architecture with simpler ML models reduces computational costs significantly. The SVM model improves solution quality, leading to better tour lengths on average	Limited scalability: evaluated on small to medium-scale TSP instances. Overfitting risks
Zheng et al. (2023)	Combination of three RL methods and LKH heuristic algorithm	**TSPLIB (14 to 85,900 cities):** **VSR-LKH with theα-measure:** Easy instances: Avg. opt. gap: 0.0161% Hard instances: Avg. opt. gap: 0.0218% **VSR-LKH with the POPMUSIC:** Easy instances: Avg. opt. gap: 0.0042% Hard instances: Avg. opt. gap: 0.0139%	Combines RL (Q-learning, Sarsa, Monte Carlo) with the LKH algorithm, enhancing the ability to adaptively select candidate edges. Shows significant improvements over standard LKH on benchmark instances, including large instances with up to 85,900 cities	VSR-LKH uses RL to adapt and learn during the solving process of each TSP instance. The Q-values that guide the search are updated dynamically based on the outcomes of the k-opt moves performed during the search
Fitzpatrick et al. (2021)	Binary classification model for sparsification followed by IP approach	**TSPLIB (up to 1000 cities):** Avg. opt. gap $\left(\frac{l_{\text{opt}}}{l_{\text{opt}}}\right)$: 1.0053	The method can prune over 85% of the edges in TSP instances while preserving most of the optimal tour edges, decreasing computational effort	Performance is influenced by the distribution of the training data. Instances significantly different from the training set may not be pruned as effectively

nodes in a permutation-invariant manner. This allows for parallel processing and more effective handling of unordered node sets, a key characteristic of TSP instances. Approaches such as the AM (Kool et al. 2019), POMO (Kwon et al. 2020), LEHD (Luo et al. 2023), and ICAM (Zhou et al. 2024) have demonstrated high performance on small to medium-sized instances, often achieving near-optimal solutions. They are also used effectively in decomposition-based frameworks to quickly solve subproblems in large-scale TSPs (Zheng et al. 2024; Ye et al. 2024; Cheng et al. 2023). Recent advancements, such as Pointerformer (Jin et al. 2023) with its reversible residual network and enriched context embeddings, have reduced memory consumption and improved generalization. Similarly, models like Tspformer (Yang et al. 2023) integrate sampled scaled dot-product attention, lowering the complexity of self-attention from $O(L^2)$ to $O(L \log L)$, enabling scalability to larger node counts. However, Transformer-based approaches face notable limitations. The standard self-attention mechanism introduces quadratic time and memory complexity, leading to high computational costs and memory overflow when scaling to large instances. While recent improvements have extended capacity to larger instances, training time remains significant, and generalization beyond seen sizes is still an ongoing challenge.

Multi-model DL-based approaches combine different DL models, such as GNNs, Transformers, and CNNs, to leverage their complementary strengths in solving TSP. These methods aim to improve solution quality, scalability, and generalization beyond what single-architecture models can achieve. For example, Neural-3-OPT (Sui et al. 2021) integrates sparse GNNs and LSTMs within a pointer network framework to enhance 3-opt local search operations. At a larger scale, approaches like GLOP (Ye et al. 2024), H-TSP (Pan et al. 2023), and UDC (Zheng et al. 2024) extend this idea using hierarchical learning or unified divide-and-conquer frameworks to partition and solve very large problems efficiently. Additionally, models like the CNN-Transformer (Jung et al. 2024) reduce GPU memory consumption while preserving solution quality, making them more resource-efficient compared to classical Transformer models. Despite their advantages, multi-model approaches involve higher design complexity and increased training overhead due to the coordination of heterogeneous components.

Hybrid DL-heuristic approaches combine DL models with classical search algorithms to leverage the learning ability of neural networks and the powerful exploration capability of heuristic algorithms. These approaches use models such as GNNs or Transformers to guide or enhance heuristic techniques like LKH, MCTS, GLS, or Ising solvers. These hybrid methods offer strong generalization and often surpass end-to-end DL-based approaches in terms of solution quality (Sui et al. 2025). GNN+Heuristic integration methods (e.g., NeuroLKH (Xin et al. 2021b), SSRL (Wang et al. 2023), NeuralGLS (Sui et al. 2023)) improve traditional heuristics like LKH and GLS by learning edge scores, penalties, or regret estimates. This leads to enhanced generalization and lower optimality gaps, even for instances with thousands of nodes. Similarly, approaches like GNN-MCTS (Fu et al. 2021) demonstrate how learned models can guide search strategies. The ability of these approaches to embed domain knowledge into search procedures allows them to handle diverse node distributions and maintain performance across problem sizes. However, hybrid approaches often require careful design and depend on existing heuristic frameworks, which may reduce flexibility for new TSP variants. Their integration can also increase computational requirements, especially when large neural models are involved.

Other DL-based approaches explore alternative DL models, particularly CNNs to solve the TSP. These approaches often reframe TSP instances as 2D grids or images, allowing CNNs or ResNets to learn spatial patterns and predict promising tour segments. They benefit from CNNs' ability to extract local features, making them effective for edge selection in small or structured instances. However, their scalability and accuracy diminish on large or dense graphs, where they tend to produce noisy outputs and less reliable edge predictions (Ling et al. 2023).

Traditional ML-based approaches apply classical techniques such as SVMs, reinforcement learning, and binary classification to support heuristic solvers in solving the TSP. These methods are often used for tasks like edge selection (Zheng et al. 2023), graph sparsification (Fitzpatrick et al. 2021), or candidate list generation (Vitali et al. 2022), and are typically integrated with local search or exact solvers. Rather than solving TSP directly, they enhance specific components of traditional algorithms to improve efficiency and solution quality. However, these approaches have limited scalability and adaptability when used alone. Their effectiveness often depends on feature engineering and integration with advanced solvers. They lack the representational power of DL models and are rarely used as standalone solutions for large TSP instances.

Table 9 summarizes the main strengths and limitations of each approach discussed in this section.

Table 9 Summary of strengths and limitations of ML-based approaches for solving TSP

Approach	Strengths	Limitations
Pointer network-based	Effective on small/medium TSP instances	Poor scalability beyond 200 nodes
GNN-based	Captures graph structure; strong generalization; guides heuristics; scalable with sparsification	Training is resource-intensive; performance drops on very large graphs without heuristics
Transformer-based	Parallel processing; adaptable; near-optimal on medium TSP	Quadratic time/memory cost; struggles with very large instances; long training times
Multi-model DL-based	Combines strengths of multiple models; scalable to large TSP instances; robust generalization	High model complexity; difficult tuning; increased training instability
Hybrid DL-heuristic-based	Balances learning and search; efficient on large TSPs; domain-aware; high solution quality	Relies on existing heuristics; complex integration; limited flexibility for new TSP variants
CNN-based	Leverages spatial patterns; improves with heuristics; fast inference in small cases	Poor scalability; noisy in dense graphs; depends on heuristics; training sensitivity
Traditional ML-based	Lightweight; useful for preprocessing and edge filtering	Limited scalability; weak as standalone solver for large TSPs

4 Discussion

This section presents the findings of this review, highlighting advancements, challenges, and future research directions in ML-based TSP solutions.

4.1 Advancements and challenges in ML-based TSP solutions

The analysis of ML-based approaches from 2015 to 2025 underscores the growing prominence of GNNs and Transformers, driven by advancements in graph processing technologies that enhance TSP solving performance. In contrast, Ptr-Nets and traditional ML approaches have seen limited adoption due to scalability challenges, reflecting a shift toward more advanced architectures capable of addressing larger and more complex problems.

Prt-Nets are recognized for their sequential data processing capabilities, but face scalability issues due to their reliance on LSTM units. Their performance is often limited to smaller TSP instances (up to 200 nodes). Hybrid models that integrate Ptr-Nets with graph-based structures have shown promise in expanding their applicability to larger instances, addressing some of the scalability challenges. **GNNs** represent a significant shift toward graph-based processing that aligns more naturally with the structure of the TSP. For small to medium-sized instances, GNNs work well because they excel at managing unordered data and capturing inter-node interactions. However, significant computational costs make it challenging to scale GNNs for large-scale, real-world challenges. Techniques such as *graph sparsification* (e.g., k-nearest neighbors) have helped reduce computational demands while maintaining solution quality, marking a potential direction for improving scalability. **Transformers** have shown strong performance through self-attention and parallelism, but their memory and time complexity limit their use on large problems. Advanced approaches like Pointerformer (Jin et al. 2023) and Tspformer (Yang et al. 2023) aim to address this. **CNN-based approaches** offer spatial reasoning capabilities but face challenges in dense graphs and typically require heuristic enhancements. **Traditional ML** methods are lightweight and often support pre-processing tasks but lack scalability as standalone solvers. **Multi-Model DL-based approaches** have emerged as a potent strategy, combining strengths from different ML architectures to tackle complex, large-scale TSP instances. Integrations of GNNs with Transformers or CNNs with Transformer architectures have shown success in managing up to 100,000 node TSPs, demonstrating improved adaptability and robustness (Ye et al. 2024). The hybridization of DL and heuristic techniques has become a prominent approach for solving TSP instances (Tian et al. 2024). **Hybrid DL and Heuristic-based approaches** combine DL with heuristic methods have shown strong potential in solving large-scale TSP instances. These methods leverage the learning ability of DL to provide guidance and insights, while relying on classical heuristics to ensure solution quality and scalability.

End-to-end constructive approaches, such as Ptr-Net-based and Transformer-based approaches, achieve high-quality solutions efficiently for small instances but suffer from high computational complexity and their memory requirements grow significantly with the number of nodes which limits their practical use in medium to large-scale instances. **Two-stage approaches** improve solution quality by integrating DL-based models with heuristic search techniques. However, they require multiple search iterations, which leads to high computational overhead and long execution times. **Decomposition-based approaches** scale better to large TSP instances. Most of these approaches use the best-performing constructive DL-based solvers, such as POMO (Kwon et al. 2020), LEHD (Luo et al. 2023), and ICAM (Zhou et al. 2024), to optimize subproblems quickly, while others leverage pre-trained GNNs, as seen in works like (Li et al. 2024; Fu et al. 2021). However, decomposition-based approaches face several challenges. Initial solutions are typically generated using weak heuristics like random insertion or greedy algorithms, requiring several refinements and multiple iterations to reach high-quality solutions. Additionally, most partitioning methods rely on fixed-sized sub-problems (Zheng et al. 2024; Fu et al. 2021; Luo et al. 2024), which do not adapt dynamically to problem complexity and lead to fragmented solutions with limited global optimization. The lack of adaptive learning prevents models from prioritizing high-impact regions. Furthermore, local refinements operate independently without global awareness, and merging strategies often fail to capture optimal inter-subtour connections, reducing overall solution quality. Inefficient parallelization also limits scalability, as many existing approaches still rely on sequential learning and inference.

A range of strategies has been employed to enhance ML solutions for the TSP. **Hard instance generation** trains models on challenging problem instances to improve robustness and generalization. **Curriculum learning** starts with easier instances and progressively introduces harder ones, boosting learning efficiency and accuracy. **Leveraging symmetries** reduces computational complexity by exploiting structural patterns, while **enhancing**

node encodings improves the model's understanding of node relationships, leading to more precise initial decisions and reducing the need for complex decoding. Advanced decoding strategies like **beam search** and **2-OPT local search** significantly narrow the optimality gap by exploring multiple potential solutions or iteratively refining segments of a route. Approaches such as **instance augmentation** apply coordinate transformations to reduce the optimality gap, while **multi-decoder strategies** employ cooperative decoders to expand exploration of the solution space. **Generalization across distributions**, particularly **cross-size and cross-distribution generalization**, ensures the model can handle variations in problem size and structure, aligning with real-world TSP distributions. Additionally, strategies like **active search**, which refines predictions during decoding, and **MCTS**, which narrows the solution gap for larger problems, further enhance performance. These targeted techniques collectively address the challenges of TSP, enabling more efficient and scalable ML solutions.

Additionally, the choice of **learning paradigm** plays a critical role in model performance. Recent trends show a growing preference for RL since 2021 due to its ability to learn without labeled data. RL adapts well to diverse TSP instances, unlike SL, which struggles to generalize across instance sizes. While SL performs well with sufficient data, RL's flexibility makes it better suited for real-world problems. However, RL faces challenges like sparse rewards and slow convergence. To address these limitations, several learning paradigms have emerged. SSRL which integrates self-supervised learning with RL to provide additional learning signals and avoid sparse rewards, as demonstrated by SSRL-based approaches that enhanced the LKH heuristic (Wang et al. 2023). Self-Improved Learning refines initial low-quality solutions through iterative learning, progressively enhancing label quality over time without requiring high-quality supervision (Luo et al. 2024). Similarly, Weakly-SL combines SL and RL to train models on noisy or incomplete labels, making learning more data-efficient while ensuring scalability to large instances (Wen et al. 2025). These approaches significantly improve generalization, reduce reliance on high-quality labeled data, and accelerate convergence. Future research should explore further hybridization of these paradigms to improve solution quality, scalability, and computational efficiency in ML-based approaches.

Scalability remains a key challenge for ML-based solutions. One promising approach is to train the model on small instances and then generalize it to solve large-scale instances. Transformer-based models, in their standard form, struggle to scale beyond 200 nodes due to the quadratic complexity of self-attention and high memory requirements. Recent innovations, such as SIL (Luo et al. 2024) and Pointerformer (Jin et al. 2023), have introduced memory-efficient architectures and optimized attention mechanisms, extending the scalability of Transformers to handle instances with up to 5,000 nodes. However, further advancements are still required to enable their application to much larger problems.

In contrast, scalability improvements in GNN-based solutions often rely on sparse message-passing or hierarchical frameworks. For example, Sun and Yang (2023) introduced a graph-based diffusion model that successfully solved TSP instances with up to 10,000 nodes by using sparse graph representations and efficient propagation mechanisms. Similarly, Xin et al. (2021b) combined a sparse graph network with the LKH heuristic to solve instances of up to 5,000 nodes, demonstrating the potential of GNN-guided heuristics for scaling to larger problems.

Decomposition strategies offer another promising avenue for scaling ML-based solutions. By leveraging divide-and-conquer principles, large-scale TSP instances are broken into smaller, more manageable sub-problems, which are solved independently using specialized ML solvers before recombination. This strategy has shown promising results in several studies (Chen et al. 2023; Cheng et al. 2023; Pan et al. 2023; Ye et al. 2024; Zheng et al. 2024). Employing this principle, Ye et al. (2024) and Zheng et al. (2024) developed neural solvers that effectively manage TSP instances up to 100,000 nodes, achieving an optimality gap of less than 5.10% and 4.05%, respectively. However, while decomposition improves scalability and computational efficiency, ensuring global consistency when merging sub-solutions remains a challenge, as local optimizations do not always align with the overall optimal tour structure.

A key remaining challenge is benchmarking ML-based solvers effectively for real-world applications. Most evaluations rely on randomly generated instances of up to 100,000 nodes due to the limited availability of real-world problem datasets. While these synthetic datasets demonstrate scalability, they may not reflect the complexity of real-world problems. Real-world benchmarks, including TSPLIB, provide more practical insights. However, reliance on TSPLIB and synthetic data is prevalent, with limited use of diverse benchmarks like National (Cook 2022) and VLSI (Rohe 2013), restricting the evaluation of ML methods across varied real-world scenarios. Future research should focus on developing representative datasets and improving model generalization to enhance the practical relevance of ML-based solvers for real-world applications.

4.2 Choosing between ML-based and traditional solvers

Comparing traditional solvers with ML-based solvers for the TSP highlights significant differences in scalability, performance, and practical applications. Traditional solvers include both exact algorithms, such as **Concorde**, and heuristic methods like **LKH** and **EAX**. Exact algorithms guarantee optimal solutions but face scalability limitations due to their exponential computational demands, making them impractical for large-scale problems. Heuristic methods provide near-optimal solutions with lower computational costs, but their efficiency and solution quality decline as problem size and complexity increase.

ML-based solvers present a promising alternative, particularly for large-scale and time-critical applications. While these models do not guarantee optimality, they achieve high-quality solutions with significantly faster inference times compared to traditional solvers. Moreover, ML-based models leverage parallel computation on GPUs, enabling them to scale efficiently and handle large instances that would be infeasible for exact solvers. Studies have shown that ML-based approaches can outperform traditional methods under the same time constraints, making them effective in real-time and dynamic environments. Furthermore, ML models reduce the need for manual feature engineering and benefit from adaptive learning capabilities, bypassing the extensive fine-tuning and expert knowledge often required by traditional heuristics.

Each type of solver has its niche in **specific applications**. Exact solvers are essential for high-precision tasks where exact solutions are critical, and extended computation times are acceptable, such as circuit design. Heuristic methods are well-suited for logistics, planning, and scheduling, where achieving good-quality solutions within a reasonable timeframe is the priority. ML-based solvers are particularly advantageous in real-time, dynamic, and resource-constrained environments. They are especially effective in operational settings requiring frequent and rapid resolution of similar optimization problems, such as daily vehicle routing with new destinations. These models perform intensive computations during training, enabling rapid inference for subsequent problem instances and making them highly efficient for time-sensitive tasks.

4.3 Open research problems and future research directions

This section outlines the key challenges and promising directions for future research in applying ML to the TSP.

- **Scalability of ML Approaches:** Current DL models perform well on small TSP instances but struggle with larger instances due to increased memory consumption and computation time. Future research should focus on developing scalable models that can handle large-scale TSP instances efficiently. Strategies, including divide-and-conquer, graph sparsification, and adaptive learning approaches, should be explored to improve scalability.
- **Generalization Capabilities:** Existing models often fail to generalize well to unseen instances that differ from the training data in terms of size or distribution. Research should focus on methods to improve generalization abilities. Approaches like curriculum learning, active search, and mixed training sets can be explored.
- **Handling Structural Problem Instances:** Clustered TSP instances and other structural data are difficult for current approaches to handle. The development of architectures that effectively handle structural complexity should be the focus of future study.
- **Exploration of Learning Paradigms:** There are limitations to the generalization of traditional SL and RL learning paradigms. Examining novel learning paradigms such as meta-learning can help train model parameters for rapid adaptation and fine-tuning to different data distributions and problem sizes.
- **Graph Reduction Techniques:** Developing new graph reduction techniques that are better than the fundamental k-nearest neighbors will improve the scalability and efficiency of GNN-based models, thus increasing their effectiveness for larger TSP instances.
- **Dependency on Heuristics:** Combining ML with traditional heuristics requires deep domain knowledge to choose and integrate effective heuristics; therefore, it restricts their generalizing capacity and broader applicability. Future research should strive to develop efficient models that incorporate the strengths of both methods.
- **Parameter Tuning of Traditional Solvers:** Combining parameter tuning with DL methods to achieve comprehensive control over traditional solver behaviors is still an open problem.
- **Extended Problem Symmetries:** Employing additional symmetries, such as scaling and translating can improve models and make them more robust and effective for solving TSP.
- **Population-Based Approaches:** Training a larger number of agents can lead to performance breakdowns and higher computational demands. Future research should inves-

tigate new ways to improve population-based approaches, which will reduce performance drops.

- **Explore Graph Transformers:** Investigating the use of recent scalable graph transformers, such as (Wu et al. 2023, 2022, 2023), to solve TSP can significantly enhance performance and applicability.

5 Conclusion

This review provides a comprehensive analysis of ML-based approaches for solving the TSP, examining key architectures such as Ptr-Nets, GNNs, Transformers, multi-model and hybrid DL-heuristic models. Our findings highlight the strengths and limitations of these methods, particularly in terms of scalability, computational efficiency, and adaptability. While ML-based solvers do not guarantee optimality, they offer significant advantages over traditional exact and heuristic methods, especially in large-scale and real-time applications where inference speed is critical.

Recent advancements, including optimized attention mechanisms, graph sparsification, and decomposition strategies, have improved the scalability of ML-based approaches, allowing them to handle increasingly complex instances. Hybrid approaches that combine ML with traditional heuristics and multi-model approaches that integrate multiple DL architectures have demonstrated further improvements in solution quality and efficiency, with some methods scaling to 100,000-node instances. Despite these advancements, challenges remain in generalization across problem distributions, maintaining solution quality in large-scale instances, and efficiently handling diverse real-world constraints.

Future research should focus on improving the generalization capabilities of ML models across various TSP instances, enhancing scalability to handle larger problems efficiently, developing adaptive learning frameworks that can dynamically adjust to the complexity of the problem, and expanding benchmarking efforts with real-world datasets. Furthermore, advancing ML-guided heuristics that integrate traditional heuristics with ML can optimize scalability and solution quality. By addressing these open challenges, the field can move towards more scalable, high-quality solutions for TSP and other combinatorial optimization problems.

Author contributions E.A. conducted the literature review and its analysis and wrote the main manuscript text. M.M. suggested the study, supervised it, and provided critical feedback. Both authors reviewed and approved the final version for submission.

Data availability No datasets were generated or analysed during the current study.

Declarations

Competing interests The authors declare no competing interests.

References

Agatz N, Bouman P, Schmidt M (2018) Optimization approaches for the traveling salesman problem with drone. Transp Sci 52(4):965–981. https://doi.org/10.1287/trsc.2017.0791

Andrew B, Richard SS (2018) Reinforcement learning: an introduction

Anson A (2024) Enhanced dynamic programming approaches aor efficient solutions to the traveling salesman problem. J Comput Sci Appl Eng 2:24–28. https://doi.org/10.70356/josapen.v2i2.32

Applegate DL, Bixby RE, Chvatál V, Cook WJ (2006) The traveling salesman problem: a computational study. Princeton University Press, Princeton

Applegate DL, Bixby RE, Chvátal V, Cook W, Espinoza DG, Goycoolea M, Helsgaun K (2009) Certification of an optimal TSP tour through 85,900 cities. Oper Res Lett 37(1):11–15. https://doi.org/10.1016/j.orl.2008.09.006

Applegate D, Bixby R, Chvatal V, Cook W (2006) Concorde Tsp Solver http://www.math.uwaterloo.ca/tsp/concorde

Bachlechner T, Majumder BP, Mao H, Cottrell G, McAuley J (2021) Rezero is all you need: fast convergence at large depth. In: Uncertainty in artificial intelligence 1352–1361. PMLR

Balas E, Toth, P: Branch and bound methods for the traveling salesman problem:. Defense Technical Information Center, Fort Belvoir, VA (1983) https://doi.org/10.21236/ADA126957

Barrena E, Canca D, Coelho LC, Laporte G (2023) Analysis of the selective traveling salesman problem with time-dependent profits. TOP 31(1):165–193. https://doi.org/10.1007/s11750-022-00632-6

Barro A (2023) Pointer networks with q-learning for combinatorial optimization. ArXiv preprint arXiv:2311.02629

Bello I, Pham H, Le, QV, Norouzi M, Bengio S (2017) Neural combinatorial optimization with reinforcement learning. arXiv:1611.09940

Bengio Y, Lodi A, Prouvost A (2021) Machine learning for combinatorial optimization: a methodological tour d'horizon. Eur J Oper Res 290(2):405–421. https://doi.org/10.1016/j.ejor.2020.07.063

Bixby RE, Gu Z, Rothberg E (2008) Gurobi Optimizer. https://www.gurobi.com/

Bogyrbayeva A, Meraliyev M, Mustakhov T, Dauletbayev B (2022) Learning to solve vehicle routing problems: a Surv. arXiv:2205.02453

Borchate R, Jibhe S, Bhandane T, Pandit PD (2024) Memory optimization using dynamic programming: a comprehensive. Int J Res Appl Sci Eng 12(11):1208–1212. https://doi.org/10.22214/ijraset.2024.65070

Bresson X, Laurent T (2021) The transformer network for the traveling salesman problem. arXiv:2103.03012

Chaslot G, Bakkes S, Szita I, Spronck P (2008) Monte-Carlo tree search: a new framework for game AI. Proc AAAI Conf Artif Intell Interact Digital Entertain 4(1):216–217. https://doi.org/10.1609/aiide.v4i1.18700

Chen X, Li Y, Yang Y, Zhang L, Li S, Pan G (2023) Extnco: a fine-grained divide-and-conquer approach for extending Nco to solve large-scale traveling salesman problem. Rochester, NY

Cheng H, Zheng H, Cong Y, Jiang W, Pu, S: Select and optimize: learning to solve large-scale tsp instances. In: International conference on artificial intelligence and statistics, 1219–1231 (2023) PMLR

Christofides N (2022) Worst-case analysis of a new heuristic for the travelling salesman problem. Ops Res Forum 3(1):20. https://doi.org/10.1007/s43069-021-00101-z

Cipra BA (1987) An introduction to the Ising model. Am Math Mon. https://doi.org/10.2307/2322600

Clarke G, Wright JW (1964) Scheduling of vehicles from a central depot to a number of delivery Points. Oper Res. https://doi.org/10.1287/opre.12.4.568

Cook W (2022) National traveling salesman problems. https://www.math.uwaterloo.ca/tsp/world/countries.html

Cook W, Held S, Helsgaun K (2024) Constrained local search for last-mile routing. Transp Sci 58(1):12–26. https://doi.org/10.1287/trsc.2022.1185

da Costa P, Rhuggenaath J, Zhang Y, Akcay A, Kaymak U (2021) Learning 2-opt heuristics for routing problems via deep reinforcement learning. SN Comp Sci 2(5):388. https://doi.org/10.1007/s42979-021-00779-2

Dai H, Dai B, Song L (2016) Discriminative embeddings of latent variable models for structured data. In: International conference on machine learning, 2702–2711, PMLR

Davendra D (2010) Traveling salesman problem: theory and applications. IntechOpen, UK

Deudon M, Cournut P, Lacoste A, Adulyasak Y, Rousseau L-M (2018) Learning heuristics for the TSP by policy gradient. In: van Hoeve W-J (ed) Integration of constraint programming, artificial intelligence, and operations research. Springer, Cham, pp 170–181

Devlin J, Chang M-W, Lee K, Toutanova, K (2019) BERT: Pre-training of deep bidirectional transformers for language understanding. arXiv https://doi.org/10.48550/arXiv.1810.04805

Drakulic D, Michel S, Mai F, Sors A, Andreoli J-M (2023) BQ-NCO: bisimulation quotienting for efficient neural combinatorial optimization. ArXiv https://doi.org/10.48550/arXiv.2301.03313

Drori I, Kharkar A, Sickinger, WR, Kates B, Ma Q, Ge S, Dolev E, Dietrich B, Williamson, DP, Udell M (2020) Learning to solve combinatorial optimization problems on real-world graphs in linear time. In: 2020 19th ieee international conference on machine learning and applications (ICMLA), pp. 19–24 https://doi.org/10.1109/ICMLA51294.2020.00013

Fang S, Luo, Z: Multi-Traveling salesman algorithm for UAV swarms based on improved genetic algorithm. In: 2024 IEEE 6th international conference on power, intelligent computing and systems (ICPICS), pp. 1027–1032 (2024)https://doi.org/10.1109/ICPICS62053.2024.10796048. ISSN: 2834-8567. https://ieeexplore.ieee.org/abstract/document/10796048 Accessed 2025-03-17

Fitzpatrick J, Ajwani D, Carroll P (2021) Learning to sparsify travelling salesman problem instances. In: Stuckey PJ (ed) Integration of constraint programming, artificial intelligence, and operations research. Springer, Cham, pp 410–426

Fu, Z-H, Sun S, Ren J, Yu T, Zhang H, Liu Y, Huang L, Yan X, Lu P (2023) A aierarchical destroy and repair approach for solving very large-scale travelling dalesman problem. ArXiv https://doi.org/10.48550/arXiv.2308.04639

Fu Z-H, Qiu K-B, Zha H (2021) Generalize a small pre-trained model to arbitrarily large TSP instances. Proc AAAI Con Artif Intell 35(8):7474–7482. https://doi.org/10.1609/aaai.v35i8.16916

Grinsztajn N, Furelos-Blanco D, Surana S, Bonnet C, Barrett T (2023) Winner takes it all: training performant RL populations for combinatorial optimization. Adv Neural Inf Process Syst 36:48485–48509

Held M, Karp RM (1962) A dynamic programming approach to sequencing problems. J Soc Ind Appl Math 10(1):196–210. https://doi.org/10.1137/0110015

Helsgaun K (2017) An extension of the Lin-Kernighan-Helsgaun TSP solver for constrained traveling salesman and vehicle routing problems. Roskilde University, Roskilde, pp 966–980

Helsgaun K (2000) An effective implementation of the Lin-Kernighan traveling salesman heuristic. Eur J Oper Res 126(1):106–130. https://doi.org/10.1016/S0377-2217(99)00284-2

Helsgaun K (2009) General k-opt submoves for the Lin-Kernighan TSP heuristic. Math Program Comput 1(2–3):119–163. https://doi.org/10.1007/s12532-009-0004-6

He K, Zhang X, Ren S, Sun, J: Deep residual learning for image recognition. In: Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 770–778 (2016)

Hu Y, Zhang Z, Yao Y, Huyan X, Zhou X, Lee WS (2021) A bidirectional graph neural network for traveling salesman problems on arbitrary symmetric graphs. Eng Appl Artif Intell 97:104061. https://doi.org/10.1016/j.engappai.2020.104061

Hudson B, Li Q, Malencia M, Prorok A (2022) Graph neural network guided local search for the traveling salesperson problem. ArXiv https://doi.org/10.48550/arXiv.2110.05291

Ilavarasi K, Joseph KS (2014) Variants of travelling salesman problem: A survey. In: International conference on information communication and embedded systems (ICICES2014), pp. 1–7 https://doi.org/10.1109/ICICES.2014.7033850

Jiang Y, Weise T, Lässig J, Chiong R, Athauda, R: Comparing a hybrid branch and bound algorithm with evolutionary computation methods, local search and their hybrids on the TSP. In: 2014 IEEE symposium on computational intelligence in production and logistics systems (CIPLS), 148–155 (2014) https://doi.org/10.1109/CIPLS.2014.7007174. https://ieeexplore.ieee.org/abstract/document/7007174 Accessed 2025-03-02

Jin Y, Ding Y, Pan X, He K, Zhao L, Qin T, Song L, Bian J (2023) Pointerformer: deep reinforced multi-pointer transformer for the traveling salesman problem. Proc AAAI Conf Artif Intell 37(7):8132–8140. https://doi.org/10.1609/aaai.v37i7.25982

Joshi CK, Laurent T, Bresson X (2019) An efficient graph convolutional network technique for the travelling salesman problem. ArXiv

Joshi CK, Cappart Q, Rousseau L-M, Laurent T (2022) Learning the travelling salesperson problem requires rethinking generalization. Constr 27(1):70–98. https://doi.org/10.1007/s10601-022-09327-y

Jung M, Lee J, Kim J (2024) A lightweight CNN-transformer model for learning traveling salesman problems. ArXiv https://doi.org/10.48550/arXiv.2305.01883

Junior Mele U, Maria Gambardella L, Montemanni, R (2021) Machine learning approaches for the traveling salesman problem: a survey. In: Proceedings of the 2021 8th international conference on industrial engineering and applications (Europe) ICIEA 2021-Europe, pp. 182–186. Association for computing machinery, New York, NY, USA https://doi.org/10.1145/3463858.3463869

Karp RM (1972) Reducibility among combinatorial problems. In: Miller RE, Thatcher JW, Bohlinger JD (eds) Complexity of computer computations: proceedings of a symposium on the complexity of computer computations. Held March 20–22, 1972, at the IBM Thomas J. Watson Research Center, Yorktown Heights, New York, and sponsored by the office of naval research, Mathematics program, IBM world trade corporation, and the IBM research mathematical sciences department, 85–103. Springer, Boston, MA https://doi.org/10.1007/978-1-4684-2001-2_9

Khalil E, Dai H, Zhang Y, Dilkina B, Song L (2017) Learning combinatorial optimization algorithms over graphs. Adv Neural Inf Process Syst 30:1

Khemani B, Patil S, Kotecha K, Tanwar S (2024) A review of graph neural networks: concepts, architectures, techniques, challenges, datasets, applications, and future directions. J Big Data 11(1):18. https://doi.org/10.1186/s40537-023-00876-4

Kim M, Park J, Park J (2023) Sym-NCO: Leveraging symmetricity for neural combinatorial optimization. ArXiv https://doi.org/10.48550/arXiv.2205.13209

Kool W, Hoof H, Gromicho J, Welling M (2022) Deep policy dynamic programming for vehicle routing problems. International conference on integration of constraint programming, artificial intelligence, and operations research. Springer, New York, pp 190–213

Kool W, van Hoof H, Welling M (2019) Attention, learn to solve routing problems! ArXiv https://doi.org/10.48550/arXiv.1803.08475

Kumar M, Panwar K, Deep K (2024) Discrete marine predators algorithm for symmetric travelling salesman problem. Evol Intel. https://doi.org/10.1007/s12065-024-00960-5

Kwon, Y.-D, Choo J, Yoon I, Park M, Park D, Gwon Y (2021) Matrix encoding networks for neural combinatorial optimization. ArXivhttps://doi.org/10.48550/arXiv.2106.11113

Kwon Y-D, Choo J, Kim B, Yoon I, Gwon Y, Min S (2020) Pomo: policy optimization with multiple optima for reinforcement learning. Adv Neural Inf Process Syst 33:21188–21198

Larni-Fooeik AM, Ghasemi N, Mohammadi E (2024) Insights into the application of the traveling salesman problem to logistics without considering financial risk: a bibliometric study. Manag Sci Lett 14(3):189–200. https://doi.org/10.5267/j.msl.2023.11.002

Li M, Tu S, Xu L (2023) Generalizing graph network models for the traveling salesman problem with Lin-Kernighan-Helsgaun heuristics. International conference on neural information processing. Springer, New York, pp 528–539

Li K, Liu F, Wang Z, Zhang Q (2025) Destroy and repair using hyper graphs for routing. arXiv preprint arXiv:2502.16170

Lin S, Kernighan BW (1973) An effective heuristic algorithm for the traveling-salesman problem. Oper Res 21(2):498–516. https://doi.org/10.1287/opre.21.2.498

Ling Z, Zhang Y, Chen X (2023) A deep reinforcement learning based real-time solution policy for the traveling salesman problem. IEEE Trans Intell Transp Syst 24(6):5871–5882

Lischka A, Wu J, Basso R, Chehreghani, M.H, Kulcsar B (2023) Travelling salesman problem goes sparse with graph neural networks

Li M, Tu S, Xu L (2024) Generalizing graph network models for the traveling salesman problem with Lin-Kernighan-Helsgaun Heuristics. In: Luo B, Cheng L, Wu Z-G, Li H, Li C (eds) Neural information processing. Springer, Singapore, pp 528–539

Liu S, Zhang Y, Tang K, Yao X (2023) How good is neural combinatorial optimization? A systematic evaluation on the traveling salesman problem. IEEE Comput Intell Mag 18(3):14–28. https://doi.org/10.1109/MCI.2023.3277768

Liu S, Lv H, Wang Z, Tang K (2025) Cascaded large-scale tsp solving with unified neural guidance: Bridging local and population-based search. arXiv preprint arXiv:2501.14285

Luo F, Lin X, Liu F, Zhang Q, Wang Z (2023) Neural combinatorial optimization with heavy decoder: toward large scale generalization. Adv Neural Inf Process Syst 36:8845–8864

Luo F, Lin X, Wang Z, Tong X, Yuan M, Zhang Q (2024) Self-improved learning for scalable neural combinatorial optimization. ArXiv https://doi.org/10.48550/arXiv.2403.19561

Ma Y, Li J, Cao Z, Song W, Zhang L, Chen Z, Tang J (2021) Learning to iteratively solve routing problems with dual-aspect collaborative transformer. Adv Neural Inf Process Syst 34:11096–11107

Ma Q, Ge S, He D, Thaker D, Drori I (2019) Combinatorial optimization by graph pointer networks and hierarchical reinforcement learning. ArXiv https://doi.org/10.48550/arXiv.1911.04936

Mazyavkina N, Sviridov S, Ivanov S, Burnaev E (2021) Reinforcement learning for combinatorial optimization: a survey. Comput Oper Res 134:105400. https://doi.org/10.1016/j.cor.2021.105400

Mele UJ, Gambardella LM, Montemanni R (2021) A new constructive Heuristic driven by machine learning for the traveling salesman problem. Algoritm 14(9):267. https://doi.org/10.3390/a14090267

Miki S, Ebara H (2019) Solving traveling salesman problem with image-based classification. In: 2019 IEEE 31st international conference on tools with artificial intelligence (ICTAI), pp. 1118–1123 https://doi.org/10.1109/ICTAI.2019.00156

Miki S, Yamamoto D, Ebara H (2018) Applying deep learning and reinforcement learning to traveling salesman problem. In: 2018 International conference on computing, electronics & communications engineering (iCCECE), pp. 65–70 https://doi.org/10.1109/iCCECOME.2018.8659266

Mitchell TM, Mitchell TM (1997) Machine learning, vol 1. McGraw-hill, New York

Mostafa SM, Habashy SM, Salem SA (2023) A new framework for multi-objective route planning in smart cities. In: Hassanien AE, Snášel V, Tang M, Sung T-W, Chang K-C (eds) Proceedings of the 8th international conference on advanced intelligent systems and informatics 2022. Springer, New York, pp 824–837

Nagata Y, Kobayashi S (2013) A powerful genetic algorithm using edge assembly crossover for the traveling salesman problem. Informs J Comput 25(2):346–363. https://doi.org/10.1287/ijoc.1120.0506

Nowak A, Villar S, Bandeira A, Bruna J (2017) A note on learning algorithms for quadratic assignment with graph neural networks

Ouyang W, Wang Y, Weng P, Han S (2021) Generalization in deep RL for TSP problems via equivariance and local search. ArXiv https://doi.org/10.48550/arXiv.2110.03595

Pan X, Jin Y, Ding Y, Feng M, Zhao L, Song L, Bian J (2023) H-TSP: hierarchically solving the large-scale traveling salesman problem. Proc AAAI Conf Artif Intell 37(8):9345–9353. https://doi.org/10.1609/aaai.v37i8.26120

Pop PC, Cosma O, Sabo C, Sitar CP (2024) A comprehensive survey on the generalized traveling salesman problem. Eur J Oper Res 314(3):819–835. https://doi.org/10.1016/j.ejor.2023.07.022

Purkayastha R, Chakraborty T, Saha A, Mukhopadhyay D (2020) Study and analysis of various heuristic algorithms for solving travelling salesman problem—a survey. In: Mandal JK, Mukhopadhyay S (eds) Proceedings of the global AI congress 2019. Springer, Singapore, pp 61–70

Qiu R, Sun Z, Yang Y (2022) DIMES: a differentiable meta solver for combinatorial optimization problems. ArXiv

Reinelt G (1991) TSPLIB-A traveling salesman problem library. ORSA J Comput 3(4):376–384. https://doi.org/10.1287/ijoc.3.4.376

Rohe A (2013) VLSI data sets. https://www.math.uwaterloo.ca/tsp/vlsi/index.html

Sanyal S, Roy K (2022) Neuro-ising: accelerating large-scale traveling salesman problems via graph neural network guided localized ising solvers. IEEE Trans Comput Aided Des Integr Circ Syst 41(12):5408–5420. https://doi.org/10.1109/TCAD.2022.3164330

Schrijver A (2005) On the history of combinatorial optimization (till 1960). Handb Oper Res Manag Sci 12:1–68

Schulman J, Wolski F, Dhariwal P, Radford A, Klimov O (2017) Proximal policy optimization algorithms. arXiv. https://doi.org/10.48550/arXiv.1707.06347

Shi Y, Zhang Y (2022) The neural network methods for solving traveling salesman problem. Procedia Comput Sci 199:681–686. https://doi.org/10.1016/j.procs.2022.01.084

Stohy A, Abdelhakam H-T, Ali S, Elhenawy M, Hassan AA, Masoud M, Glaser S, Rakotonirainy A (2021) Hybrid pointer networks for traveling salesman problems optimization. PLoS ONE 16(12):0260995. https://doi.org/10.1371/journal.pone.0260995

Sui J, Ding S, Xia B, Liu R, Bu D (2023) NeuralGLS: Learning to guide local search with graph convolutional network for the traveling salesman problem. Neural Comput Appl. https://doi.org/10.1007/s00521-023-09042-6

Sui J, Ding S, Huang X, Yu Y, Liu R, Xia B, Ding Z, Xu L, Zhang H, Yu C (2025) A survey on deep learning-based algorithms for the traveling salesman problem. Front Comp Sci 19(6):1–30

Sui J, Ding S, Liu R, Xu L, Bu D (2021) Learning 3-opt heuristics for traveling salesman problem via deep reinforcement learning. In: Asian conference on machine learning, pp. 1301–1316 PMLR

Sun Z, Yang Y (2023) DIFUSCO: Graph-based diffusion solvers for combinatorial optimization. Adv Neural Inf Process Syst 36:3706–3731

Sun R, Zheng Z, Wang Z (2024) Learning encodings for constructive neural combinatorial optimization needs to Regret. Proc AAAI Conf Artif Intell 38(18):20803–20811. https://doi.org/10.1609/aaai.v38i18.30069

Tian Y, Zhu Q, Shao S, Si L, Zhang, X: A deep reinforcement learning assisted heuristic for solving traveling salesman problems. In: 2024 IEEE congress on evolutionary computation (CEC), IEEE, pp. 1–8 (2024)

Vaswani A, Shazeer N, Parmar N, Uszkoreit J, Jones L, Gomez AN, Kaiser Ł, Polosukhin I (2017) Attention is all you need. Adv Neural Info Proc Syst 30:1

Veličković P, Cucurull G, Casanova A, Romero A, Liò P, Bengio Y (2018) Graph attention networks arXiv. https://doi.org/10.48550/arXiv.1710.10903

Vinyals O, Fortunato M, Jaitly N (2015) Pointer networks. Adv Neural Info Proc Sys 28:1
Vitali T, Mele UJ, Gambardella LM, Montemanni R (2022) Machine learning constructives and local searches for the travelling salesman problem. In: Trautmann N, Gnägi M (eds) Operations research proceedings 2021. Springer, New York, pp 59–65
Voudouris C, Tsang EP, Alsheddy A (2010) Guided local search. Handbook of metaheuristics. Springer, Cham, pp 321–361
Voudouris C, Tsang EPK, Alsheddy A (2010) Guided local search. In: Gendreau M, Potvin J-Y (eds) Handbook of metaheuristics. Springer, Boston, pp 321–361
Wang Y, Chen Z (2022) Dynamic graph Conv-LSTM model with dynamic positional encoding for the large-scale traveling salesman problem. Math Biosci Eng 19(10):9730–9748. https://doi.org/10.3934/mbe.2022452
Wang Q, Lai KH, Tang C (2023) Solving combinatorial optimization problems over graphs with BERT-Based Deep Reinforcement Learning. Inf Sci 619:930–946. https://doi.org/10.1016/j.ins.2022.11.073
Wang Q, Zhang C, Tang C (2023) Discovering Lin-Kernighan-Helsgaun heuristic for routing optimization using self-supervised reinforcement learning. J King Saud Univ Comput Inf Sci 35(8):101723. https://doi.org/10.1016/j.jksuci.2023.101723
Wang Z-C, Liang K, Bao X-G, Wu T-H (2024) A novel algorithm for solving the prize collecting traveling salesman problem based on DNA computing. IEEE Trans Nanobiosci 23(2):220–232. https://doi.org/10.1109/TNB.2023.3307458
Wang C, Yu Z, McAleer S, Yu T, Yang Y (2024) ASP: Learn a universal neural solver. IEEE Trans Pattern Anal Mach Intell. https://doi.org/10.1109/TPAMI.2024.3352096
Wang M, Zhou Y, Cao Z, Xiao Y, Wu X, Pang W, Jiang Y, Yang H, Zhao P, Li Y (2025) An efficient diffusion-based non-autoregressive solver for traveling salesman problem. arXiv preprint arXiv:2501.13767
Wen J, Li Y, Selman B, He, K: Localescaper: A weakly-supervised framework with regional reconstruction for scalable neural tsp solvers. arXiv preprint arXiv:2502.12484 (2025)
Wu Y, Song W, Cao Z, Zhang J, Lim A (2022) Learning improvement heuristics for solving routing problems. IEEE Trans Neural Netw Learn Syst 33(9):5057–5069. https://doi.org/10.1109/TNNLS.2021.3068828
Wu Q, Zhao W, Li Z, Wipf DP, Yan J (2022) NodeFormer: A scalable graph structure learning transformer for node classification. Adv Neural Inf Process Syst 35:27387–27401
Wu Q, Zhao W, Yang C, Zhang H, Nie F, Jiang H, Bian Y, Yan J (2023) SGFormer: simplifying and empowering transformers for large-graph representations. Adv Neural Inf Process Syst 36:64753–64773
Wu Q, Yang C, Zhao W, He Y, Wipf D, Yan J (2023) DIFFormer: Scalable (Graph) Transformers induced by energy constrained diffusion. ArXiv https://doi.org/10.48550/arXiv.2301.09474
Xin L, Song W, Cao Z, Zhang J (2021) Multi-decoder attention model with embedding glimpse for solving vehicle routing problems. Proc AAAI Conf Artif Intell 35(13):12042–12049. https://doi.org/10.1609/aaai.v35i13.17430
Xin L, Song W, Cao Z, Zhang J (2021) Neurolkh: Combining deep learning model with Lin-Kernighan-Helsgaun heuristic for solving the traveling salesman problem. Adv Neural Inf Process Syst 34:7472–7483
Xing Z, Tu S (2020) A graph neural network assisted monte carlo tree search approach to traveling salesman problem. IEEE Access 8:108418–108428. https://doi.org/10.1109/ACCESS.2020.3000236
Xu Y, Fang M, Chen L, Xu G, Du Y, Zhang C (2022) Reinforcement learning with multiple relational attention for solving vehicle routing problems. IEEE Trans Cybern 52(10):11107–11120. https://doi.org/10.1109/TCYB.2021.3089179
Yang H, Zhao M, Yuan L, Yu Y, Li Z, Gu M (2023) Memory-efficient transformer-based network model for traveling salesman problem. Neural Netw 161:589–597. https://doi.org/10.1016/j.neunet.2023.02.014
Yang L, Wang X, He Z, Wang S, Lin J (2024) Review of traveling salesman problem solution methods. In: Pan L, Wang Y, Lin J (eds) Bio-inspired computing: theories and applications. Springer, Singapore, pp 3–16
Ye H, Wang J, Cao Z, Liang H, Li Y (2023) Deepaco: Neural-enhanced ant systems for combinatorial optimization. Adv Neural Inf Process Syst 36:43706–43728
Ye H, Wang J, Liang H, Cao Z, Li Y, Li F (2024) GLOP: learning global partition and local construction for solving large-scale routing problems in real-time. Proc AAAI Conf Artif Intell 38(18):20284–20292. https://doi.org/10.1609/aaai.v38i18.30009
Zabinsky ZB, Ho T-Y, Huang H (2019) Integrating heuristics and approximations into a branch and bound framework. In: 2019 IEEE 15th international conference on automation science and engineering (CASE), pp. 774–779 https://doi.org/10.1109/COASE.2019.8842982. ISSN: 2161-8089. https://ieeexplore.ieee.org/abstract/document/8842982 Accessed 2025-03-12
Zhang Z, Zhang Z, Wang X, Zhu W (2022) Learning to solve travelling salesman problem with hardness-adaptive curriculum. Proc AAAI Conf Artif Intell 36(8):9136–9144. https://doi.org/10.1609/aaai.v36i8.20899

Zheng J, He K, Zhou J, Jin Y, Li C-M (2021) Combining reinforcement learning with Lin-Kernighan-Helsgaun algorithm for the traveling salesman problem. Proc AAAI Conf Artif Intell 35(14):12445–12452. https://doi.org/10.1609/aaai.v35i14.17476

Zheng J, He K, Zhou J, Jin Y, Li C-M (2023) Reinforced Lin-Kernighan-Helsgaun algorithms for the traveling salesman problems. Knowl-Based Syst 260:110144. https://doi.org/10.1016/j.knosys.2022.110144

Zheng Z, Zhou C, Xialiang T, Yuan M, Wang Z (2024) UDC: A unified neural divide-and-conquer framework for large-scale combinatorial optimization problems. arXiv https://doi.org/10.48550/arXiv.2407.00312

Zhou C, Lin X, Wang Z, Tong X, Yuan M, Zhang Q (2024) Instance-conditioned adaptation for large-scale generalization of neural combinatorial optimization. arXiv. arXiv:2405.01906 [cs] https://doi.org/10.48550/arXiv.2405.01906. Accessed 2025-02-20

Zhou J, Wu Y, Song W, Cao Z, Zhang J (2023) Towards omni-generalizable neural methods for vehicle routing problems. In: International conference on machine learning, pp. 42769–42789 PMLR