eng

MDPI

Article

Combinatorial Route Optimization Using Near-Training-Free Foundation Models

Nguyen Gia Hien Vu [1], Yifan Tang [1], Rey Lim [2], Yifan Yang [3], Hang Ma [3], Ke Wang [1] and G. Gary Wang [1,*]

[1] Product Design and Optimization Laboratory, Simon Fraser University, Surrey, BC V3T 0A3, Canada; hien_vu@sfu.ca (N.G.H.V.); yta88@sfu.ca (Y.T.); kwa83@sfu.ca (K.W.)
[2] Alumnus, Simon Fraser University, Burnaby, BC V5A 1S6, Canada; reysteven@gmail.com
[3] School of Computing Science, Simon Fraser University, Burnaby, BC V5A 1S6, Canada; yya305@sfu.ca (Y.Y.); hangma@sfu.ca (H.M.)
* Correspondence: gary_wang@sfu.ca; Tel.: +1-778-782-8495

Abstract

Combinatorial Route Optimization (CRO) problems, such as the Vehicle Routing Problem (VRP) or the Travelling Salesman Problem (TSP), are commonly seen in scheduling, logistics, and transportation. While current machine learning (ML) methods can overcome certain limitations of traditional approaches, including exact and heuristic algorithms, they typically require substantial computational resources, large training datasets, and carefully designed models, thereby limiting their scalability and practical deployment. In this paper, we develop a method to address such concerns in a data-efficient and near-training-free manner using foundation models. We select TSP, one of the most well-known combinatorial optimization problems, to solve in our experiments and employ the Tabular Prior-Data Fitted Network (TabPFN), one of the newly designed foundation models. Specifically, we develop a node-based formulation that converts TSP into a sequence of localized prediction tasks and constructs a complete route through in-context learning provided by TabPFN. The proposed method enables TabPFN, a model developed for regression and classification, to be applied to CRO problems with only one TSP sample for fine-tuning. We evaluate the proposed method across varying TSP instance sizes and demonstrate that our approach generalizes effectively without retraining, maintains competitive solution quality, and exhibits promising scalability. These findings suggest that CRO problems can be approached through foundation models, enabling scalability as well as generating rapidly deployable solutions with near-training-free adaptation.

Keywords: combinatorial optimization; foundation model; graph-based problems; tabular prior-data fitted network; travelling salesman problem; routing

Academic Editors: Asela K. Kulatunga and M. Vijaya Manupati

Received: 25 June 2026
Revised: 24 July 2026
Accepted: 29 July 2026
Published: 1 August 2026

1. Introduction

Combinatorial Optimization (CO) is a branch of mathematical optimization that aims at optimizing discrete variables with a nonpolynomial increase in possible solutions with a given number of variables [1]. Many CO problems are hence considered NP-hard [1]. In the real world, CO problems are commonly seen both in research and industries, such as logistics, production scheduling, or transportation [2–4]. As an example, finding the optimal route for the Travelling Salesman Problem (TSP) is one of the most well-known problems in the CO domain [5,6], arising from routing and scheduling applications [7]. Conceptually, TSP is a graph-based problem defined by a set of nodes, and the goal is to visit each node exactly once, then return to the starting point, with the minimum total route

costs, such as route length or time [8]. TSP has been studied in thousands of publications each year [6], and adopted in a wide range of applications such as those in engineering, biology, or economics, just to name a few [9–11].

Traditional algorithms for TSP include two main branches, namely exact algorithms and heuristic algorithms [5]. Exact algorithms try to find global optimal solutions by exploring the entire solution space exhaustively [5]. There are different methods for exact algorithms, such as Dynamic Programming (DP) or Branch-and-Bound (BB). The DP method, such as the Held & Karp algorithms [12], tries to break TSP instances into smaller subproblems to avoid redundant computation. Although this method ensures the optimality of the final solution, it requires exponential time complexity and high resource requirements, resulting in impracticality for large-size TSP instances [13]. Different from the DP method, the BB method [14] explores the solution space using branching to search the corresponding space and eliminate any suboptimal branches. However, similar to the DP method, this method still has relatively high time complexity [5]. Furthermore, Concorde [15], an advanced exact TSP solver frequently used as a benchmark in various studies [2–4,16–19], continues to struggle with large-size TSP instances, as highlighted by [20].

To resolve time complexity issues, heuristic and metaheuristic methods have been developed [21]. One of the most famous SOTA solvers in this category is the LKH algorithm [22], which employs k-opt strategies to conduct a local search process. Notably, while heuristic methods are significantly faster than exact algorithms, they are developed for specific TSP instances. For instance, in Helsgaun's report [22], only a limited number of solvers are introduced for specific TSP variants, which makes it difficult to adapt these solvers when new constraints are introduced. Furthermore, for large-sized TSP instances, the search process might still take hours on a multicore CPU [23]. Similar to exact algorithms, heuristics and metaheuristics might fail to return optimal solutions within reasonable time frames, especially for new TSP variants [21]. Therefore, there should be a better TSP-solving algorithm that can both meet timeframe requirements and maintain the quality of the final solution.

In this trend, Machine Learning (ML) models, especially Deep Learning (DL) ones, have gained significant attention in recent years [2–4,16–19]. Different research and publications evidence the success of this approach in solving TSP instances rapidly once the training has been completed without sacrificing the quality of the solution, revealing considerable potential to apply ML techniques such as transformers, attention mechanisms, or diffusion models [2–4,16–19]. However, this trend often meets four barriers as follows:

- All these models require extremely long training time, as well as dedicated specialized GPUs [2–4,16–19]. For example, the authors of [4] spend over 20 days using a specialized A100 GPU to train their specialized diffusion model. Similarly, according to [17], their training time can take up to one week for full convergence.
- Many current models employ probabilistic methods and often become untrainable on TSP instances that have more than 100 nodes [3]. Some researchers have attempted training their models on larger instance sizes, but each of the varying TSP instance sizes requires a respective specialized model, or else the performance suffers [4,24].
- A large quantity of training samples is required. For instance, ref. [17] generates up to 200 million samples for the whole training process, while the model by [19] requires over one million samples.
- Much of the current research encodes the entire problem, i.e., whole-instance encoding, regardless of the input instance size [19]. This limits the efficiency of encoding TSP instances of varying sizes with different characteristics, particularly if the ML model size is intended to remain relatively constant.

In this context, foundation models have emerged as potential candidates to alleviate or resolve those barriers. Foundation models can be defined as extensive Artificial Intelligence (AI) systems that are pre-trained on massive datasets of general information with different learning techniques and adapted to various downstream tasks [25]. In this sense, foundation models offer a promising alternative for many downstream applications, including TSP and other routing or CRO problems, because their large-scale pre-training enables greater data efficiency, rapid adaptation, and reduced task-specific training. However, foundation models often have a large number of parameters, such as LLaMA with up to 65 billion parameters [26], TimesFM with 200 million parameters [27], or GPT-3 with 175 billion parameters [28]. These models can be customized for particular tasks by further adapting them to domain-specific data, such as fine-tuning pre-trained models [29]. In other words, this is similar to transfer learning, where the foundational knowledge learned during the pre-training is transferred to new tasks with much smaller amounts of data [30]. This enables foundation models to use domain-specific data more efficiently and adapt to new tasks faster compared to training a model from scratch. In fact, different foundation models have been designed for a range of applications, such as Generative Pre-Training (GPT) [31], MOMENT [32], or TabPFN-v2 [33]. Although many of them have only been developed recently, foundation models have found many applications in different fields, such as engineering [34], medicine [35], or education [36].

Among foundational models, the Tabular Prior-Data Fitted Network (TabPFN) is a newly designed model for small to medium-sized tabular data, requiring a small training data size and short training time [33,37]. Indeed, TabPFN-v2 is advantageously seen as outperforming current leading models by a substantial margin across various benchmark tests [33]. It should be highlighted that, although they possess such prominent advantages, TabPFN in general and TabPFN-v2 in particular have considerably fewer parameters compared to other existing foundation models [37,38], making them well-suited particularly for resource-constrained scenarios. In reality, TabPFN-v2 has seen its applications in different fields, such as time series forecasting [38], geotechnical [39], and medical applications [40]. However, despite such merits, TabPFN is developed for regression and classification and has not been employed in tackling TSP and other combinatorial route optimization (CRO) problems yet. Therefore, there remain open questions such as how to leverage foundation models, in this case TabPFN-v2, to enable AI-based, scalable, and near-training-free solutions for CRO.

To address this gap, this work develops a data-efficient and near-training-free method to solve CRO problems with foundation models by introducing a node-based formulation in combination with TabPFN-v2, using TSP as a representative case of CRO problems. Via testing and comparison with other methods, we demonstrate that our approach can be directly applied to problems of varying sizes without additional training, large training datasets, or costly optimization procedures. Accordingly, this paper aims to achieve the following goals:

- To develop a near-training-free adaptation strategy using only one sample that allows TabPFN-v2 to predict the next node in a TSP route. This strategy aims to minimize training time, hardware resources, and data requirements, thereby offering a faster and more resource-efficient solution than traditional methods.
- To develop a node-based encoding approach for better scalability. This method is to avoid expressiveness issues inherent in whole-instance encoding methods [4], and handle large TSP instances without the need for extensive retraining, thus making it scalable for different problem sizes in real-world applications.

- To develop the first application of TabPFN-v2 to CRO problems. To the best of the authors' knowledge, TabPFN-v2, as a supervised learning model for regression and classification, has never been used to solve CRO problems.
- To benchmark and evaluate our approach with varying TSP instance sizes, and compare the performance of the proposed approach against state-of-the-art (SOTA) algorithms.

Within such a scope, the remainder of this paper is structured as follows: Section 2 briefly reviews TSP and TabPFN with a focus on TabPFN-v2. Based on such background, Section 3 discusses our proposed approach and experimental setup. Section 4 presents and analyzes our results. Finally, Section 5 summarizes the findings and recommends possible future research.

2. Background

2.1. Travelling Salesman Problem (TSP)

TSP is a relatively old problem and has been mentioned as early as the 18th century [41]. TSP has garnered sustained attention and interest from researchers and industry [9–11], owing to its substantial theoretical significance and extensive practical applications, thereby justifying it as a representative CRO problem. To name a few [41]:

- TSP as an optimization problem is easy to describe, yet an NP-hard problem.
- TSP has significantly broad applicability which goes beyond normal daily routing, as it is also applicable in areas such as scheduling, planning, or even cryptoanalysis.
- TSP has become a test problem, especially to test the usefulness of a new CO or CRO method.

TSP involves a set of nodes, where the goal is to determine the lowest-cost route that visits each node exactly once and returns to the starting node (c) [8,42]. In formal terms, TSP is defined on an undirected or edge-directed graph $G = (V, E)$, where V represents the set of n nodes and E is the set of edges connecting those nodes ($E = \{e_{ij}\}$ for $(i, j = 1, 2, 3, \ldots, n; i \neq j)$. Each edge has a corresponding cost c_{ij}. The set of feasible solutions F includes all edge subsets that form a Hamilton cycle, and the objective function to be minimized is the sum of edge weights $w(e)$ for all edges in the solution $f \in F$ [8,42]. TSP can be formulated using binary variables x_{ij} where $x_{ij} = 1$ if edge e_{ij} is used in the optimal solution $(i \neq j)$, or else $x_{ij} = 0$. Altogether, the TSP formulation can be stated as follows [42]:

Minimize

$$w(e) = \sum_{i \neq j} c_{ij} * x_{ij} \quad (1)$$

S.T.

$$\sum_{i=1}^{n} x_{ij} = 1, \ \forall \mathrm{j} \in \mathrm{V} \quad (2)$$

$$\sum_{j=1}^{n} x_{ij} = 1, \ \forall \mathrm{i} \in \mathrm{V} \quad (3)$$

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

$$x_{ij} \in \{0, 1\}, \ \forall \mathrm{i}, \mathrm{j} \in \mathrm{V}, \mathrm{i} \neq \mathrm{j} \quad (5)$$

In this formulation, the objective (Equation (1)) describes the cost of the optimal tour. Equations (2) and (3) guarantee the flow of the route, which means each node should be entered exactly once (Equation (2)) and left exactly once (Equation (3)). Equation (4) eliminates a potential subtour, i.e., a full Hamilton cycle on a subset that does not include all n nodes. Finally, Equation (5) imposes a binary constraint on decision variables [42].

Throughout the years, more specialized variants have also been researched to meet different requirements, including Symmetric TSP (STSP) [43], Asymmetric TSP (ATSP) [44], and Multiple TSP (MTSP) [45].

2.2. Tabular Prior-Data Fitted Network (TabPFN)

TabPFN is a powerful foundation model specialized in tabular data [33,37], being developed using the Prior-Data Fitted Network (PFN) architecture [33,37,46]. In essence, as proposed by Müller et al. [46], PFN is a modified Transformer encoder designed to perform Bayesian inference. The architecture removes positional encodings from the Transformer to ensure permutation invariance in the dataset. In principle, PFN is trained across a large number of artificial datasets drawn from a prior distribution $p(\mathcal{D})$, learning to approximate the posterior predictive distribution $p(y|x, D)$. Methodically, the model is trained using Prior-Data Negative Log-Likelihood (NLL) loss, which can be defined as

$$l_\theta = E_{D\cup(x,y)\sim p(\mathcal{D})}[-log(q_\theta(y|x, D))] \quad (6)$$

In this equation, l denotes the loss function to be minimized, θ represents the parameters of the neural network, E denotes the expectation operator, q_θ is the parameterized model, D represents the dataset, x denotes the input features, and y corresponds to the true label associated with each input x. In addition, $D \cup (x, y)$ is a dataset sampled from $p(\mathcal{D})$ [46]. Equation (6) drives the model to match its predictive distribution to the true Bayesian posterior averaged over priors [46].

Based on the PFN architecture, ref. [37] leverages in-context learning, the same mechanism behind the success of large language models (LLMs), to develop a fully learned tabular prediction mechanism as TabPFN. Next, ref. [33] improves TabPFN and introduces TabPFN-v2, which can handle larger datasets and support up to 10,000 examples for tasks like regression and classification. Instead of learning from one dataset through gradient updates as in conventional supervised learning, TabPFN-v2 is trained across millions of synthetic datasets, allowing it to infer new relationships directly from context without further optimization. It should be emphasized that this method enables the model to approximate a wide range of learning algorithms, including complex Bayesian ones, through a single forward pass. The core idea is to generate massive synthetic tabular datasets using structural causal models that encode realistic dependencies, noise, categorical variables, and missing values. This generative framework allows TabPFN-v2 to observe a rich variety of data patterns—linear, nonlinear, correlated, or noisy—and therefore enables it to generalize across a broad spectrum of tabular problems [33].

Figure 1 illustrates the overall architecture of TabPFN-v2. As shown in the upper section of Figure 1, the entire tabular dataset, consisting of labeled training samples and unlabeled test samples, is provided as a single input. The input table is first transformed into cell embeddings, where each table cell is represented by a learnable embedding vector. These embeddings are then processed by a stack of twelve 2D transformer layers, each composed of feature attention, sample attention, and a feed-forward MLP. Feature attention models relationships among features within the same sample (row-wise), while sample attention captures dependencies among samples for the same feature (column-wise). Finally, a prediction head produces either class probabilities or predictive distributions for the target variables. Using this architecture, TabPFN-v2 is trained to minimize the loss on a synthetic dataset and can then be used on an actual dataset during the inference phase, as depicted in the lower section in Figure 1.

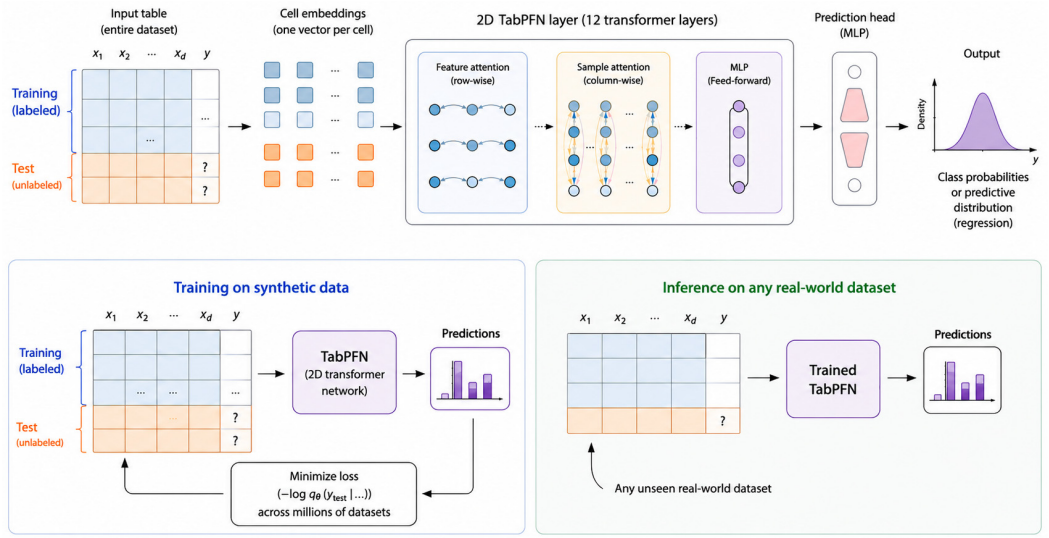

Figure 1. TabPFN-v2 architecture overview, adapted from [33].

When applied to a new, unseen dataset, TabPFN-v2 receives all training and test samples at once and outputs predictions through in-context learning—essentially performing training and inference in one step. It should be noted that its two-way attention architecture allows each cell in a table to attend both to other features in the same row and to the same feature across different rows, ensuring permutation invariance and efficient handling of tabular structure. Theoretically, similar to PFN, TabPFN-v2 approximates Bayesian posterior prediction, learning to predict $P(Y_{test}|X_{test}, Y_{train}, X_{train})$ [33]. Overall, TabPFN-v2 transforms algorithm design into a data-driven meta-learning problem, replacing explicit programming and parameter tuning with a model that learns to learn directly from diverse, causally grounded synthetic examples, resulting in a universal and efficient tabular prediction engine.

TabPFN-v2 has shown promising performance across multiple test datasets [33]. In just a few seconds, TabPFN-v2 can easily outperform even SOTA models in different tasks without requiring further adjustments [33,37]. Furthermore, TabPFN-v2, as a foundation model, allows users to perform fine-tuning for enhanced performance [33] and to tackle a wide range of tasks such as security [47], geotechnical [39], or medical applications [40]. These successes inspire us to employ it in combination with our node-based formulation to tackle CRO problems with TSP as an illustration in this paper.

3. Method

This section presents the proposed method for solving the Travelling Salesman Problem (TSP) using the proposed node-based formulation and TabPFN-v2 to construct an undirected Hamiltonian tour. As illustrated in Figure 2, the proposed workflow consists of data preparation, input processing, rapid adaptation from one sample, output decoding, and optional post-processing. This section also describes the benchmarking and evaluation protocol used to assess the performance of the proposed method.

3.1. Data Preparation

In this study, it is stressed that only a single sample is generated for the adaptation and fine-tuning process for all the TSP instances of varying sizes. The fine-tuned TabPFN is then applied to other TSP instances with direct inference. In our procedure, this single sample is generated by creating 500 points randomly inside a 2D unit square to represent a 500-city TSP. Each point is represented by two coordinates $x \in [0, 1]$ and $y \in [0, 1]$. It is

important to note that this is also a standard procedure adopted in previous studies such as [4].

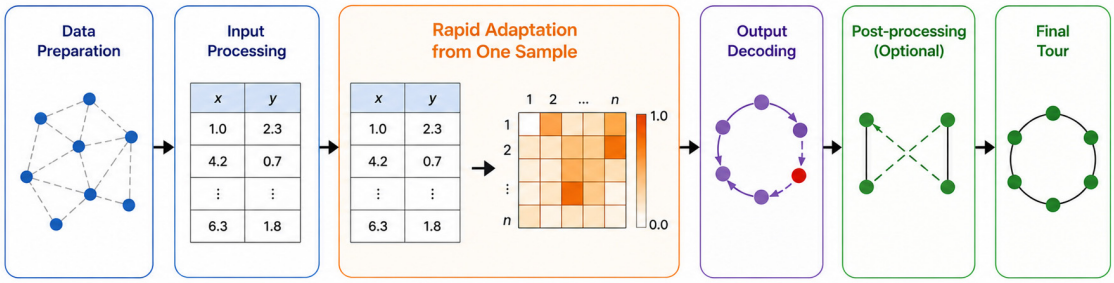

Figure 2. Overview of the proposed method.

Next, Google OR-Tools, or OR-Tools, is selected as the TSP solver for the data generation process. Methodologically, OR-Tools is chosen since it is a highly trusted and publicly available software, considered to be fast, portable, and designed to solve both simple and complex CO problems using SOTA algorithms [48,49]. Moreover, unlike other specialized solvers designed exclusively for routing problems like TSP, such as LKH-3 [22], OR-Tools provides a general-purpose optimization platform that can effectively address a diverse set of CO problems, including the Bin Packing Problem (BPP), Job Shop Scheduling Problem (JSSP), and Knapsack Problem (KP) [49]. In our designed procedure, this OR-Tools runs for 30 min to solve this single 500-point sample, and the best-found solution is recorded for input processing.

3.2. Training and Solution Decoding

In this study, training and solution decoding are designed as a three-step process, including input processing, model adaptation, and output decoding.

3.2.1. Input Processing

As the adaptation and fine-tuning process aims to learn from the OR-Tools solver's optimized solution, the input should meaningfully capture information from the optimized tour. The input for TabPFN-v2 is based on the best-found solution generated by OR-Tools, which produces a single optimized tour for one input sample. In theory, if the entire route of a sample is used as the input, i.e., the whole-instance method, the number of columns representing such a sample must be large enough to accommodate all node information and might vary with the size of the corresponding TSP instances. For this reason, this whole-instance method requires separate adapted models for TSP of different sizes.

To address the above-mentioned limitation, this study adopts a node-based formulation that appends nodes to the solution one by one. For clarification, the proposed method involves multiple node predictions within a single TSP sample, with each prediction corresponding to the model output for that specific node. To enable meaningful learning, the output must adapt to each new node input, necessitating multiple distinct outputs for each node prediction. However, since there is only one optimal solution for the single generated TSP instance, the fully optimized tour cannot be used directly as the output data for training or adaptation.

Instead of predicting the entire tour at once, our node-based method can be reframed as a sequence prediction problem, where the model predicts the potential location to visit at each step. As a result, the input for the next node prediction is set to be the information of the current node position. At this stage, there are two critical considerations. First, raw 2-D coordinates and edge-distance data alone lack contextual relationships and combinatorial information of TSP instances that guides real-world route construction. As discussed

by [19], the spatial information of the current node and its surrounding neighbours is extremely important in finding an optimal tour. Indeed, in a TSP sample, nodes in proximity are more likely to be connected in the optimal solution [19]. This suggests that the information about the closest neighbours can be meaningful for adaptation and fine-tuning, and therefore should be included in the input. Second, the information about previously visited nodes of the solution can be another important input. However, incorporating such information requires sequential inference steps to build the TSP route because each node depends on the current route, which prevents the use of parallel processing and increases inference time.

With such considerations, to enable the model to construct optimal tours, the input layer is carefully designed to encode both the local neighbourhood structure and the current state of traversal. In short, the input information includes the following:

- The current node in the path is represented by its two-dimensional coordinates. This feature provides an explicit spatial reference for the model, allowing it to evaluate where the tour is presently located and to plan the next move accordingly.
- The k-closest neighbours are represented. During the adaptation process, the closest neighbours are selected based on their distance from the next node location. In the testing process, as there is no information on the location of the next node, the distances are calculated based on their respective distances from the current node in the path. However, as mentioned by [19], such a difference in the distance calculation has minimal impact since nearby nodes are more likely to be connected in the optimal solution.

In the next step, the output of the model predicts the potential location of the next node to be visited, which effectively transforms TSP into a regression task, as the model estimates the continuous two-dimensional coordinates of the next node rather than selecting a node from a discrete set of candidates.

To illustrate the structure of the proposed input representation during the inference stage, the node-based embedding components are visualized in Figure 3. The current node (circle shape) serves as the focal point of prediction, while the predicted next node (triangle shape) and the actual next node (square shape) indicate the model's output and ground truth in the optimal solution, respectively. The five closest neighbor nodes (cross shape) to the current node (circle shape) are labelled from the closest node (Node 1) to the furthest node (Node 5). These nodes represent the local spatial context encoded in the input features of the model. Figure 3 also demonstrates that the next node to be visited in the optimal solution is Node 2, instead of the closest node (Node 1). This strategy bypasses the local greedy choice (Node 1) and selects the less favorable option (Node 2) when Node 2 exhibits higher spatial connectivity (i.e., closer to multiple nodes). By prioritizing nodes that offer greater future branching opportunities, the algorithm increases the likelihood of improved path construction in subsequent decision steps.

By adopting this node-based approach, the proposed method aims to provide the following benefits:

- Scalability across different instance sizes: Each prediction step relies on a fixed number of nearby neighbors, preventing the input size from growing with the full graph. This significantly simplifies the model architecture and removes the need for specialized designs when testing on new instance sizes. Moreover, this method may help mitigate the generalization issue across different TSP instance sizes, a challenge discussed by [24].
- Improved emphasis on local spatial structure: By directly observing local neighbors, the model can adapt to local geometric patterns which are often crucial in TSP, as argued by [19].

- Reduced computational costs: Since each step only processes a small local neighborhood, the inference for each node is lightweight, enabling more efficient computation.

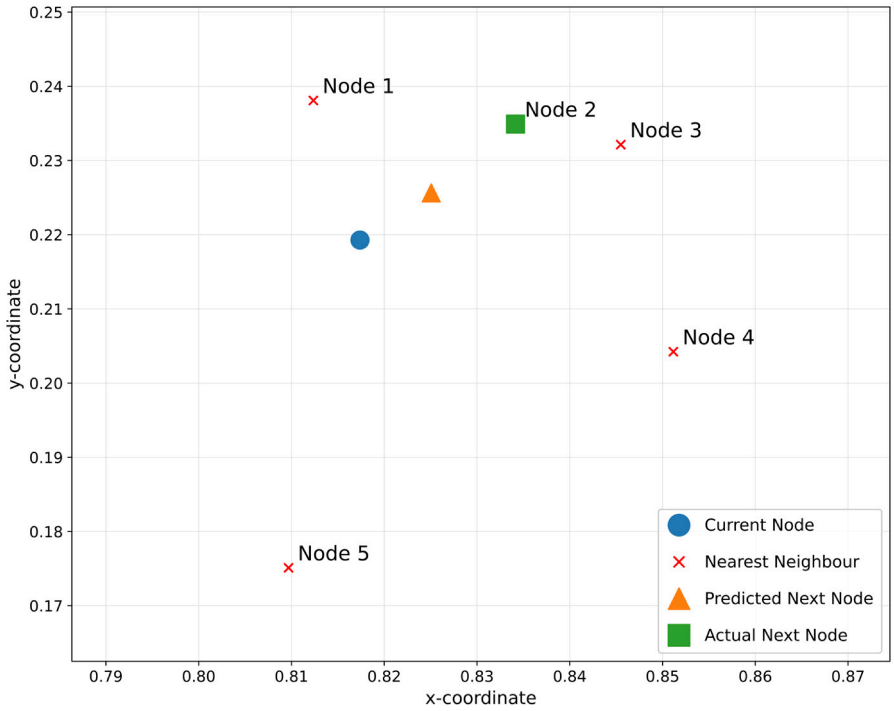

Figure 3. Visualization of node-based input embedded in a TSP sample.

3.2.2. Model Adaptation

The model adaptation process, or fine-tuning process, adjusts the pretrained model to the target dataset. During this process, the original and publicly available adaptation and fine-tuning code provided by the TabPFN authors [33] is used to update model parameters and align internal components for prediction, resulting in the adapted model. In principle, the adapted model is created with a two-step process as follows:

- First, the number of closest neighbours used as the input is optimized by testing various configurations to determine the optimal value. This step applies the input processing procedure in Section 3.2.1 and the above-mentioned adaptation and fine-tuning code [33]. In our design, the number of neighbours in the k-nearest neighbours method ranges from 1 to 40, using the same 500-node sample as the input for all evaluations.
- Second, the best-adapted TabPFN-v2 regressor model, optimized with the optimal number of closest neighbors, is saved. Notably, since TabPFN-v2 does not support multiple outputs, two distinct models are created separately to predict the x-coordinate and y-coordinate independently.

A key advantage of the proposed method in general and this two-step framework is that it is near-training-free, requiring a minimal adaptation stage (approximately two minutes using a single TSP instance), rather than training a model from the very first step. It should be noted that the offline generation of adaptation and fine-tuning solutions is part of the data preparation process and is therefore not considered part of the model training time cost.

During the inference for evaluation, the same type of input created for the adaptation process (i.e., the location of each node and its k-closest neighbour) is given to the adapted model, resulting in one row for each node in each TSP evaluation sample. The adapted model then uses this same-type input to predict the potential location of the next node for each node in parallel. As these input rows are independent, they can be processed

simultaneously by TabPFN-v2 in a single forward pass, enabling parallel computation on the GPU. This prediction can subsequently be used to assess the potential of all nodes to infer the next possible stop for each node, using the decoding method outlined in Section 3.2.3.

3.2.3. Output Decoding

As described in Section 3.2.1, the adapted model is designed to output the potential location $\left(x_i^{po}, y_i^{po}\right)$ of the next node for the current node i, and each node i is processed independently. Since each node is predicted exactly once, the complexity of this process is $O(n)$. However, since different test samples might have different node locations, a probabilistic mapping is required rather than rigidly matching a fixed set of locations. Hence, it is necessary to design a decoding mechanism with due reference to the decoding method used by [4].

To be more specific, the adapted model first examines the list of nodes $\{(x_1, y_1), (x_2, y_2), (x_3, y_3), \ldots, (x_n, y_n)\}$ and then assigns an unnormalized score for the current node i and the other nodes using the Euclidean distance

$$D_{ij} = \sqrt{\left(x_i^{po} - x_j\right)^2 + \left(y_i^{po} - y_j\right)^2} \tag{7}$$

This calculation results in a matrix D of size n^2. It should be noted that, since each node is visited exactly once, D_{ij} is manually set to 0 if $i = j$, indicating the probability of the same node edge will be 0. After that, any value that is infinite or equal to 0 is filtered out, and the median τ of the remaining values is calculated. The array D is then normalized by dividing it by the median τ, or

$$D_{normalized} = \frac{D}{\tau} \tag{8}$$

The next decoding step is to translate this matrix into a probability function by applying the softmax function, resulting in the probability matrix P. In this matrix, each entry $P(i, j)$ represents the likelihood that point i is connected to point j. Diagonal entries are manually set to 0 to avoid self-loops during the matrix construction process. In this notation, it is important to emphasize that the whole matrix is stacked together, resulting in a n^2 vector before the softmax function is applied. Overall, each component of the matrix is calculated using the following Equation (9):

$$P(i,j) = \begin{cases} softmax(-D_{normalized}(i,j)) & if\ i \neq j \\ 0 & if\ i = j \end{cases} \tag{9}$$

Then, two types of candidate edges are generated using the same Equation (9) as follows:

- Spatial k-NN edges: Edges are generated between two nodes if either node identifies the other as one of its k nearest spatial neighbors.
- Full-matrix edges: Edges are generated between all pairs of nodes.

It should be noted that, since the number of close neighbors is predefined, the complexity to calculate the spatial k-NN is $O(kn)$, while the complexity to calculate the full-edge matrix is $O(n^2)$.

The final decoding step is to create the actual TSP route. Adapting from the method used in [4], this step sequentially adds unconnected edges (i.e., edges that have not yet been selected as part of the solution) in descending order of their predicted probabilities using the spatial k-NN edges while ensuring that no subtour is formed, i.e., there is no cycle that does not cover all nodes. In other words, the algorithm prevents the formation of

premature Hamiltonian cycles by rejecting any edge whose endpoints are already connected through the partially constructed route, since adding such an edge would create a smaller cycle that excludes some vertices. Such an edge is permitted only after all vertices have been incorporated into a single connected path, at which point the final edge closes the Hamiltonian cycle. Moreover, if the spatial k-NN edges cannot create a complete solution, the full-matrix edges can be used to create a feasible one. The actual decoding step is illustrated in Algorithm 1.

Algorithm 1. Output Decoding

1. If the number of points ≤ 1 then
2. Return the trivial route.
3. End If
4. Compute the spatial k-nearest neighbors.
5. Predict the next location for each point.
6. Generate the probability matrix.
7. Generate candidate edges from:
8. (a) Spatial k-NN edges.
9. (b) Probability-based full-matrix edges.
10. Compute the mutual probability score for each candidate edge.
11. Rank the candidate edges by mutual probability.
12. For each ranked candidate edge in Spatial k-NN edges, do
13. If adding the edge satisfies the degree and cycle constraints then
14. Add the edge to the solution.
15. End If
16. End For
17. If a complete tour is not obtained, then
18. Evaluate the remaining probability-based full-matrix edges.
19. Continue greedy edge selection.
20. Connect the remaining endpoints if necessary.
21. End If
22. Traverse the selected edges from the start node to construct the route.
23. Return the route and selected edges.

Algorithm 1 first handles the trivial case where the input contains zero or one point (Lines 1–3). It then computes the spatial k-nearest neighbors, predicts the next location for each point using the adapted TabPFN models, and constructs the probability matrix as described in Equations (7)–(9) (Lines 4–6). The resulting probability matrix is directed, where each entry $P(i, j)$ represents the probability of transitioning from node i to node j, and therefore $P(i, j)$ is not necessarily equal to $P(j, i)$. Candidate edges are subsequently generated from both the spatial k-NN edges and the probability-based full-matrix edges (Lines 7–9). Since TSP is defined on an undirected graph, the transition probabilities in both directions are combined using the product $P(i, j)P(j, i)$ to rank each candidate edge. The candidate edges are then sorted in descending order of the combined probability using the Spatial k-NN edges (Lines 10–11). The algorithm greedily selects candidate edges while ensuring that each node has at most two incident edges (i.e., the degree of every node does not exceed two) and that no partial cycle, i.e., premature Hamiltonian cycle, is formed before the final tour is completed (Lines 12–16). If the initial candidate set is insufficient to construct a complete tour, the remaining probability-based full-matrix edges are evaluated using the same ranking criterion as the Spatial k-NN edges, i.e., candidate edges are sorted in descending order of the probability, to repair the partial solution and

connect the remaining endpoints when necessary (Lines 17–21). Finally, the selected edges are traversed from the specified start node to reconstruct and return the complete TSP route (Lines 22–23).

3.3. Post-Processing

The solution created by the decoding phase (Section 3.2.3) can be further improved by a simple 2-opt heuristic. Until now, 2-opt has been the heuristic that is utilized in many studies to improve initial solutions, such as those in [3,4]. In our research, post-processing works by picking all possible pairs of nodes from a complete solution and reversing the order of the route between those two nodes, resulting in the worst complexity of $O(n^2)$ for the post-processing process. The new solution is accepted if it is better than the best-known solution of a given sample, or else that new solution is discarded. The process continues until the time limit is reached or an iteration cannot find a better solution.

3.4. Benchmarking and Evaluation

The benchmarking and evaluation process is based on TSP instances of different sizes. In this process, it is critical to ensure that only one adapted model using a single sample for adaptation is tested for all TSP instances, and any change in evaluating sizes must not require new adaptation. For fair comparison, the evaluation of small instance sizes (i.e., 100 nodes or fewer) is done on two different sizes: 50 nodes (TSP-50) and 100 nodes (TSP-100), using the dataset created by [24] (Dataset available at https://github.com/chaitjo/learning-tsp, accessed on 28 July 2026). For larger instance sizes (i.e., 500 nodes or more), the test is conducted on two sizes of 500 nodes (TSP-500) and 1000 nodes (TSP-1000), using the test dataset created by [50] (Dataset available at https://github.com/SaneLYX/TSP_Att-GCRN-MCTS, accessed on 28 July 2026). It should be noted that the solutions in all the above-mentioned test datasets [24,50] are created using the Concorde exact solver [15]. In addition to the Concorde exact solver, we also compare our results with solutions created with LKH-3 and OR-Tools, which are two popular SOTA TSP metaheuristic solvers [22,49]. It should be emphasized that the results of LKH-3 are often comparable to those of the Concorde exact solver, with nearly no performance gap, as shown in many publications [2–4,18,51]. Another important fact is that not all of these publications present test results on all test samples with different instance sizes (TSP-50, TSP-100, TSP-500, and TSP-1000).

To ensure consistency and comparability with prior work, this study adapts the same evaluation metrics and baseline as provided in other papers [3,4,16,18]. Furthermore, the training time and number of samples used for benchmarking models reported in these works and for our method are included in our comparative evaluation. For simplicity, all methods based on AI and ML, including TabPFN-v2 and those proposed by [3,4,16,18,51], are referred to as ML-based solvers or ML-based models in this study. Altogether, we adopt four evaluation metrics throughout our experiments for TSP across all four instance sizes. They are:

- Length: The average total distance of the optimized routes.
- Gap: The relative gap between the length of the best solver (i.e., the baseline) and each corresponding solver. To be more specific, the gap is computed as the absolute distance difference between the baseline and the corresponding solver, divided by the route length of the baseline. Similar to the method adopted in prior works, the Concorde exact solver [15] is selected as the baseline for all evaluation samples in this study.
- Training or adaptation time required for each model.

- Data usage: The number of TSP samples used for training for each method and each instance size.

It should be made clear that the latter two metrics are only applicable to ML-based solvers, including TabPFN-v2.

In our design, the inference time is not reported as a metric, as all ML-based solvers produce solutions within a few seconds per sample using simple greedy decoding. Thus, the difference in inference time between different methods and solvers is considered negligible.

It should be emphasized that, even when different papers can share the same evaluation metrics, their test results might vary subject to different test attempts due to:

- Differences in generated samples: Since samples are often generated randomly, the actual optimal length for each sample might be different from that of the others. Therefore, this paper uses the gap between each solver and the baseline for each instance as the main metric for performance comparison.
- Differences in hardware: Different research might use different hardware resources, especially the GPU, resulting in varied performance.
- Differences in test sizes: Different papers might use different sizes of test instances, resulting in different test times.
- Effects of parallelism: Most of the available public codes for published papers use parallel processing to improve inference time and make full use of available hardware. This can improve performance, especially solving time, when multiple GPUs are available.

Based on the above-mentioned concerns, to make our comparisons consistent and fair, the results of our method are calculated based on the average of 30 instances for each problem size. This is because the central limit theorem is applicable when the sample size is 30 or more, as suggested by [52]. Furthermore, to demonstrate the performance of a SOTA ML-based solver with limited training resources, we retrain the Fast-T2T model as proposed by [4] using 50,000 samples for two different instance sizes, i.e., 50 nodes (TSP-50) and 100 nodes (TSP-100), resulting in a total of 100,000 training samples used across these two instance sizes. The retrained models achieved for these two instance sizes are referred to in this paper as Fast-T2T 50-node and 100-node models, respectively. Rationally, this Fast-T2T model by [4] is selected for retraining as it is one of the most recent SOTA solvers for TSP, with the source code available online (The code associated with the study by [4] is available at https://github.com/Thinklab-SJTU/Fast-T2T, accessed on 28 July 2026). In addition, to ensure a fairer comparison, we present the test results of Fast-T2T models using pretrained models provided by the original authors [4], which we refer to as the original Fast-T2T model in this paper.

Another consideration in benchmarking and evaluation is related to the gap metric. Although it is widely adopted as an evaluation metric for TSP solvers, it does not fully characterize the robustness of an algorithm. First, benchmark instances generated from the same probability distribution may exhibit randomness, leading to natural variations in the observed optimality gap. Second, although the adaptation dataset and all hyperparameters remain unchanged, stochastic operations during the adaptation process may produce slight variations in the adapted model.

To address these considerations, two complementary statistical analyses are conducted in this study. First, the variability of the optimality gap across the evaluation instances for each benchmark instance is quantified by reporting descriptive statistics, including the mean, median, standard deviation, minimum, maximum, interquartile range (IQR), and 95% confidence interval (CI). Second, the reproducibility of the proposed framework is evaluated by repeating the adaptation process using different random seeds while maintaining the same adaptation sample, benchmark datasets, and experimental settings.

Both analyses are conducted on all four instance sizes, i.e., TSP-50, TSP-100, TSP-500, and TSP-1000. The resulting performance variation is then analyzed to assess the sensitivity of our proposed method to stochasticity during adaptation.

4. Results and Discussion

Presenting the experimental results, this section starts with experiments on small TSP instance sizes (i.e., TSP-50 and TSP-100) to identify the effect of the number of closest neighbors on the quality of output solutions in Section 4.1. Then, Section 4.2 presents the evaluation results on all four instance sizes (TSP-50, TSP-100, TSP-500, and TSP-1000). Next, Section 4.3 focuses on robustness and statistical variability analysis. Finally, Section 4.4 discusses the results presented in Sections 4.1–4.3. All experiments were performed on a system equipped with an Intel Xeon Gold 5218 CPU and an Nvidia A40 GPU, and the proposed framework was implemented in Python 3.11.13, using PyTorch 2.8.0 with CUDA 12.8. TabPFN-v2 version 2.1.2, NumPy 2.3.2, SciPy 1.16.1, and Google OR-Tools 9.14.6206 were also employed for benchmark generation and optimization. For simplicity, our proposed method, which combines the node-based formulation with TabPFN-v2, is referred to as TabPFN-v2 in this section.

4.1. Effects of the Number of Closest Neighbors

Based on the input processing and adaptation method presented in Section 3.2, Section 4.1 explores whether varying the number of closest neighbors can influence the model performance. It is important to note that this part tests the effect only on small instance sizes (i.e., TSP-50 and TSP-100), but not on larger instance sizes (e.g., TSP-500 and TSP-1000). The underlying reason is to reduce possible risks of bias caused by size-specific adjustments during evaluation. This is in contrast with methods like Fast-T2T [4] that require specialized models for each instance size. As specified in Section 3.2.2, the number of closest neighbors is varied from 1 to 40. Also, to ensure statistical reliability, results are evaluated by calculating the average gap across 30 instances randomly sampled from the dataset by [24].

As illustrated in Figure 4, the performance varies significantly with the number of closest neighbors, highlighting the need to optimize this hyperparameter. When the number of closest neighbors k is too small, the model does not have enough information to learn, resulting in poor performance. In contrast, when k is significantly large, the model might be affected negatively by less useful information, resulting in downgraded results. It can be inferred that the best performance range is achieved when the number of closest neighbors k is between four and eight, with little variation across this range, with five being the best. In other words, the optimal performance might be attainable when the number of closest neighbors is set to five. Accordingly, all subsequent experiments in this study adopt five closest neighbors to ensure consistency. It should be noted that this optimal number k may differ for specific cases.

4.2. Evaluation Results

Using the four evaluation metrics as stated in Section 3.4, Table 1 presents the comprehensive evaluation of Concorde exact solver, heuristic solvers, SOTA ML-based solvers, and TabPFN-v2 for smaller instance sizes (TSP-50 and TSP-100). It should be noted that the gaps presented in Section 4.2 are compared against the Concorde exact solver as the baseline.

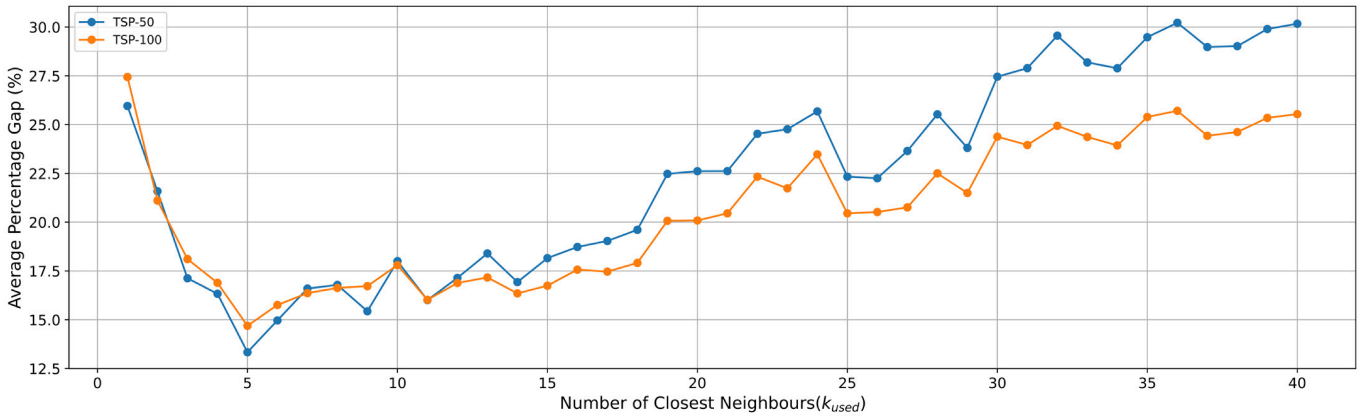

Figure 4. Effect of closest neighbour size on average percentage gap.

Table 1. Results with Greedy Decoding on TSP-50 and TSP-100. SL: Supervised Learning, RL: Reinforcement Learning, G: Greedy Decoding, T_s : Number of Sampling Steps, T_g: Number of Gradient Search Steps, x1: Greedy Inference. * denotes results that are quoted from previous works.

Method/Model Name	Type	TSP-50				TSP-100			
		Length	Gap (%)	Training or Adaptation Time	Data Used (Samples)	Length	Gap (%)	Training or Adaptation Time	Data Used (Samples)
Concorde (baseline) [15,24]	Exact	5.65	0.00	N/A	N/A	7.70	0.00	N/A	N/A
LKH-3 [22]	Heuristics	5.65	0.00	N/A	N/A	7.70	0.00	N/A	N/A
OR-Tools [49]	Heuristics	5.66	0.03	N/A	N/A	7.87	2.17	N/A	N/A
AM * [4,16]	RL+G	5.80	1.76	27 h 13 m	128,000 k	8.12	4.53	45 h 50 m	64,000 k
DIFUSCO * [3,4] $T_s = 1$	SL+G	6.42	12.84	N/A	1502 k	9.32	20.20	N/A	1502 k
DIFUSCO * [3,4] $T_s = 100$	SL+G	5.71	0.41	N/A	1502 k	7.84	1.16	N/A	1502 k
T2T * [4,51] $T_s = 1, T_g = 1$	SL+G	6.15	8.15	56 h 24 m	1502 k	9.00	16.09	206 h 20 m	1502 k
T2T * [4,51] $T_s = 50, T_g = 30$	SL+G	5.69	0.03	56 h 24 m	1502 k	7.76	0.11	206 h 20 m	1502 k
Fast-T2T [4] 50-node model	SL+G	5.91	4.52	6 h 40 m	50,000	8.78	13.96	10 h 25 m	50,000
Fast-T2T [4] 100-node model	SL+G	5.94	5.04	6 h 40 m	50,000	8.18	6.25	10 h 25 m	50,000
Fast-T2T [4] Original model	SL+G	5.69	0.31	112 h 45 m	1502 k	7.76	1.32	488 h 12 m	1502 k
Fast-T2T with 2-opt [4] 50-node model	SL+G+2-opt	5.70	1.03	6 h 40 m	50,000	7.95	3.1	10 h 25 m	50,000
Fast-T2T with 2-opt [4] 100-node model	SL+G+2-opt	5.71	1.19	6 h 40 m	50,000	7.83	1.62	10 h 25 m	50,000
Eformer * (x1) [18]	SL+G	5.70	0.13	N/A	51,000 k	7.79	0.32	N/A	101,000 k
TabPFN	SL+G	6.40	13.27	2 m	1	8.88	15.36	2 m	1
TabPFN with 2-opt	SL+G+2-opt	5.77	2.07	2 m	1	7.90	2.59	2 m	1

In general, Concorde and other heuristic solvers such as OR-Tools or LKH-3 can achieve relatively strong performance, and their performance differences are relatively insignificant (2.17% gap or less). At the same time, other learning methods, i.e., DIFUSCO [3],

T2T [51], Fast-T2T [4], Eformer [18], TabPFN-v2 [33], demonstrate the potential to achieve competitive performance in terms of solution quality, especially with the help of simple 2-opt post-processing, as evidenced by the gaps of approximately 3% or less between TabPFN-v2, Fast-T2T, and Concorde.

As shown in Table 1, for these smaller instance sizes (TSP-50 and TSP-100), TabPFN-v2's performance is comparable to that of other SOTA ML-based solvers, whether it is with or without the help of 2-opt post-processing. At this stage, it can be highlighted that, even without 2-opt post-processing, TabPFN-v2 outperforms DIFUSCO and T2T in a number of cases. For example, for TSP-100, DIFUSCO with $T_s = 1$ achieves a 20.2% gap, seeing TabPFN-v2 outperforming at 15.36%. In contrast, TabPFN-v2 exhibits lower performance compared to solvers like AM [16], Fast-T2T [4], and Eformer [18] on these smaller instance sizes. However, this performance gap can be narrowed significantly through the application of simple 2-opt post-processing. For example, TabPFN-v2 with 2-opt achieves performance similar to that of other ML-based models combined with 2-opt, as evidenced by gap differences of less than 1% between TabPFN-v2 and retrained Fast-T2T, and approximately 2% compared to the Concorde exact solver baseline.

With such performance, TabPFN-v2 visibly shows distinct advantages in terms of training time and data requirements. To be more specific, ML-based models, except for TabPFN-v2, require substantial amounts of data (from 25,600 samples to more than 100 million samples), and long training time (from 6 h up to more than 20 days). From our observation, when training resources are limited (i.e., fewer samples and less training time), the performance of current SOTA ML-based models deteriorates significantly. This is evidenced by performance gaps ranging from 5% to 10% between our retrained Fast-T2T models and the version provided by the original authors [4] on both TSP-50 and TSP-100. In contrast, TabPFN-v2 adapts to TSP in just two minutes using a single sample, and even outperforms SOTA ML-based models in some cases, as evidenced by the comparison between TabPFN-v2, T2T, and DIFUSCO in our experiments.

With the same four metrics provided in Section 3.4, Table 2 shows the evaluation results of TabPFN-v2 on larger instance sizes (TSP-500 and TSP-1000), still having the Concorde exact solver [15] as the baseline for all gap analysis. A trend similar to that of Table 1 is demonstrated in Table 2. From numerical perspectives, Concorde and heuristic solvers still perform the best, as performance gaps between Concorde, LKH-3, and OR-Tools are relatively small (4% or less). However, Concorde results are slightly worse than LKH-3, possibly due to numerical precision errors that might occur during the calculation process. Similar findings have also been reported in previous studies, such as TSP-10000 results in the research by [53]. Despite this, such an error is relatively small, as shown by the gap of almost 0% between these two solvers. In contrast, ML-based solvers produce significantly varied results. It can clearly be seen that performance gaps for these larger instance sizes for AM [16] and Eformer [18] increase significantly, being between 20 and 30% for TSP-500 and over 30% for TSP-1000 compared to the Concorde exact solver [15].

At this stage, TabPFN-v2 is also observed to be prominent for two distinguishable reasons, with performance gaps of less than 20% on TSP-500 and slightly over 20% on TSP-1000. Firstly, while TabPFN-v2 uses a single training sample across all four instance sizes, other SOTA models still rely on a relatively large number of training samples when such information is available. However, samples used by these SOTA models for TSP-500 and TSP-1000 are considerably fewer in number than those for TSP-50 and TSP-100, likely due to high costs of data collection. For example, DIFUSCO [3] and Fast-T2T [4] use 128,000 samples and 64,000 samples for TSP-500 and TSP-1000 training, respectively, compared to over 1.5 million on TSP-50 and TSP-100. The reduced amount of data might be one of the explanations for the degraded performance of other SOTA supervised learning

models, such as AM [16] and Eformer [18], as evidenced by the performance drop from less than 5% on smaller sizes to over 20% on larger ones. However, even with fewer training samples, the training time remains relatively high, as demonstrated by the nearly 15-day training duration for original Fast-T2T [4] on TSP-1000. Secondly, on TSP-500, using only a single sample for adaptation, TabPFN-v2, with a gap of 19.66%, keeps performing better than the others, especially compared to AM (20.99%) and Eformer (28.81%). On TSP-1000, TabPFN-v2 (22.90%) outperforms AM (34.75%) by more than 10 percentage points, while ref. [18] does not report the results of Eformer for this instance size.

Table 2. Results with Greedy Decoding on TSP-500 and TSP-1000. SL: Supervised Learning, RL: Reinforcement Learning, G: Greedy Decoding, T_s : Number of Sampling Steps, x1: Greedy Inference. * denotes results that are quoted from previous works.

Method/Model Name	Type	TSP-500				TSP-1000			
		Length	Gap (%)	Training or Adaptation Time	Data Used (Samples)	Length	Gap (%)	Training or Adaptation Time	Data Used (Samples)
Concorde (baseline) [15,50]	Exact	16.58	0.00	N/A	N/A	23.23	0.00	N/A	N/A
LKH-3 [22]	Heuristics	16.55	0.00	N/A	N/A	23.13	0.00	N/A	N/A
OR-Tools [49]	Heuristics	17.08	3.02	N/A	N/A	24.17	3.72	N/A	N/A
AM * [4,16]	RL+G	20.02	20.99	N/A	N/A	31.15	34.75	N/A	N/A
DIFUSCO * [3,4] $T_s = 100$	SL+G	18.17	9.82	N/A	128,000	25.74	11.36	N/A	64,000
Fast-T2T [4] Original model	SL+G	17.55	6.10	142 h 17 m	128,000	24.63	6.53	324 h 43 m	64,000
Eformer * (x1) [18]	SL+G	21.28	28.81	N/A	N/A	N/A	N/A	N/A	N/A
TabPFN	SL+G	19.84	19.66	2 m	1	28.55	22.90	2 m	1
TabPFN with 2-opt	SL+G+2-opt	17.12	3.27	2 m	1	24.36	5.31	2 m	1

4.3. Robustness and Statistical Variability Analysis

Based on the method provided in Section 3.4, Section 4.3 examines the variability of the optimality gap across 30 independent instances for each problem size, and the reproducibility of the proposed framework on all four sizes, i.e., TSP-50, TSP-100, TSP-500, and TSP-1000, without 2-opt post-processing.

4.3.1. Variability Across Benchmark Instances

Since the benchmark datasets consist of randomly generated TSP instances, the observed optimality gap naturally varies from one instance to another. To characterize this variability, Table 3 summarizes the mean, median, standard deviation, minimum, maximum, IQR, and 95% CI of the gaps between TabPFN-v2 without 2-opt and the baseline, i.e., the Concorde exact solver. Furthermore, to increase the accuracy, the number of random instances has also increased from 30 to 100 instances for each TSP size.

Table 3 complements the benchmark results presented in Tables 1 and 2 by providing a statistical characterization of the optimality gap across a larger set of benchmark instances. It should be noted that the mean gaps reported in Table 3 differ slightly from those in Tables 1 and 2 because the latter are computed using 30 randomly generated instances, whereas Table 3 uses 100 random instances for each benchmark size to increase the accuracy.

Table 3. Descriptive statistics of the optimality gap across 100 instances of each TSP size.

Instance Sizes	Mean (%)	Median (%)	Minimum (%)	Maximum (%)	Standard Deviation (%)	IQR (%)	Lower 95% CI (%)	Upper 95% CI (%)
TSP-50	19.71	18.14	6.22	39.75	7.24	10.29	18.28	21.15
TSP-100	20.02	19.53	9.44	31.79	4.41	5.44	19.14	20.89
TSP-500	21.07	21.00	15.46	29.06	2.98	3.89	20.47	21.66
TSP-1000	23.41	23.35	18.84	29.96	1.98	2.30	23.02	23.81

Since TSP instances are generated randomly, individual instances may exhibit substantially different geometric characteristics and intrinsic difficulties, especially for smaller instances with sparser density. Consequently, the average gap obtained from a smaller subset is more susceptible to sampling variability, while a larger evaluation set provides a more reliable estimate of the expected performance over the benchmark distribution. Furthermore, as shown in Table 3, the average gap increases gradually from 19.71% for TSP-50 to 23.41% for TSP-1000, which is consistent with the benchmark results in Tables 1 and 2. This shows that larger TSP instances remain more challenging for the proposed framework. The corresponding median values exhibit the same increasing trend, yet remain consistently lower than the respective means across all benchmark sizes. This indicates a positive skew in the distribution of the optimality gaps, suggesting that only a relatively small subset of benchmark instances produces noticeably larger gaps, whereas the majority exhibit much smaller ones. Consequently, these more challenging high-gap instances increase the arithmetic mean while having a smaller influence on the median.

In addition, the variability of the optimality gap decreases considerably as the instance size increases. The standard deviation drops from 7.24% for TSP-50 to 1.98% for TSP-1000, while the gap range narrows from 6.22% and 39.75% to 18.84% and 29.96%. These relatively large standard deviations for TSP-50 and TSP-100, as shown in Table 3, indicate that different random benchmark instances can produce noticeably different optimality gaps. Consequently, the averages reported in Tables 1 and 2, which are computed from only 30 instances, are more sensitive to the specific selected subset, explaining the larger differences from the corresponding 100-instance averages in Table 3. The difference between the mean and the median also becomes progressively smaller as the instance size increases, indicating that the influence of the high-gap instances diminishes for larger benchmark instances. A similar trend is observed for the IQR and the width of the 95% CI, which decrease consistently as the instance size increases. This decreasing trend indicates that the estimate of the expected optimality gap becomes increasingly precise as the influence of benchmark randomness decreases.

The above-mentioned trends can be partially explained by the effect of randomness in the benchmark generation process. As discussed in Section 3.1, all benchmark instances are created by randomly sampling node locations within a unit square. For smaller TSP instances, the placement of only a few nodes can substantially alter the geometry of the problem, producing benchmark instances with widely varying levels of difficulty. Consequently, the performance of the proposed framework becomes more sensitive to the specific random realization of the instance, resulting in larger variations in the observed performance gap between TabPFB-v2 and the baseline. In contrast, as the number of nodes increases, the randomly generated instances become more representative of the underlying spatial distribution. Therefore, individual random node placements contribute proportionally less to the overall instance structure, causing the influence of randomness to diminish. Overall, although larger TSP instances are inherently more difficult, they also

exhibit more homogeneous characteristics, leading to more consistent performance across independently generated benchmark instances.

4.3.2. Reproducibility Across Random Seeds

To isolate the effect of stochasticity during adaptation, the complete adaptation process was repeated using different random seeds while maintaining the same adaptation sample, benchmark datasets, number of close neighbors, and other hyperparameters. The reported minimum and maximum correspond to the smallest and largest average optimality gaps, respectively, obtained over 30 independent adaptation runs, where each average is computed from the same set of 30 benchmark instances.

Table 4 summarizes the average gap between TabPFN-v2 and the Concorde baseline obtained from 30 independent adaptation runs using different random seeds while maintaining the same adaptation sample, benchmark datasets, and hyperparameters. Consequently, the variation observed in Table 4 reflects only the stochasticity introduced during the adaptation process, rather than differences in the benchmark instances themselves.

Table 4. Summary of the optimality gap across independent adaptation runs using different random seeds.

Instance Size	Minimum Average (%)	Maximum Average (%)
TSP-50	13.27	19.67
TSP-100	15.63	19.36
TSP-500	19.66	20.18
TSP-1000	22.76	22.97

Overall, the proposed framework exhibits only limited sensitivity to the random seed used during adaptation. The average optimality gap ranges from 13.27% to 19.67% for TSP-50, 15.63% to 19.36% for TSP-100, 19.66% to 20.18% for TSP-500, and 22.76% to 22.97% for TSP-1000. Although some variation is observed for the smaller benchmark sizes, the differences become progressively smaller as the instance size increases. In particular, the range of the average gap decreases from 6.40% to only 0.21% for TSP-50 and TSP-1000, respectively, indicating that the average performance becomes increasingly insensitive to stochasticity during adaptation. This trend is consistent with the statistical analysis presented in Table 3. As already discussed in Section 4.3.1, larger TSP instances exhibit lower instance-to-instance variability because the influence of random node placement is reduced as the number of nodes increases. Consequently, averaging the performance over the same set of 30 benchmark instances produces nearly identical results across independent adaptation runs, even when different random seeds are used. In contrast, the larger variability observed for TSP-50 and TSP-100 suggests that small differences introduced during adaptation are more likely to affect the average performance when benchmark instances themselves exhibit greater variation in difficulty.

4.4. Analysis and Discussion

The results presented in Tables 1 and 2 show that the proposed method can be successfully applied across different instance sizes while maintaining competitive solution quality. This conclusion not only implies the scalability and generalization capabilities of the algorithms but also enables rapid deployment even in resource-constrained scenarios. Moreover, to the best of our knowledge, our application is the first of its kind in harnessing the powerful and advantageous features of TabPFN-v2 as an example of foundation models to solve CRO problems in general and TSP in particular.

Throughout the experiments, despite not being originally designed for sequential, combinatorial, or graph-based inputs, TabPFN-v2, when combined with the proposed node-based approach, achieves comparable or superior performance to SOTA ML-based solvers across all four instance sizes (TSP-50, TSP-100, TSP-500, and TSP-1000). This leads to four observations as follows:

- Firstly, as shown in Tables 1–4, the performance of the proposed method is achieved without extensive training, only relying on a single sample across all different instance sizes. Possible underlying reasons can include the capability of TabPFN-v2 to guide the algorithm away from the seemingly best immediate step by leveraging the strengths of in-context learning if that greedy move leads to a suboptimal solution.
- Secondly, the performance analyzed particularly in Section 4.2 highlights adaptation and training efficiency as one of the core advantages of the proposed method, especially in applications where training data are scarce, expensive, or time-consuming to acquire.
- Thirdly, the proposed method, being adapted in minutes, can be very useful when it comes to resource-scarce applications. In contrast, most traditional ML-based models often require substantial resources, including tens of thousands to millions of training samples and long training sessions ranging from several hours to multiple days.
- Fourthly, the proposed method can be adapted to new problems faster and utilize domain-specific data more efficiently while maintaining comparable performance.

In terms of methodology, another significant contribution derived from our proposed approach is the use of a node-based encoding method, as opposed to existing whole-instance encoding methods. In essence, this encoding method can leverage the above-mentioned in-context learning capacity of TabPFN-v2 to improve the generalization ability of the model, especially when scaling from small to large TSP samples. Using the Concorde exact solver [15] as the baseline, the results confirm that the proposed method can maintain relatively strong performance for varied instance sizes despite using a single adapted model. This is evidenced by smaller gaps of less than 20% for TSP-500 and slightly over 20% for TSP-1000 versus the baseline (Section 4.2). In contrast, whole-instance encoding models such as those proposed by [18] often experience performance degradation when the same whole-instance embedding architecture is applied to different instance sizes. This limitation is evident in our experiments on TSP-500 and TSP-1000, where solvers such as Eformer [18] exhibit performance gaps of approximately 30% (Section 4.2). This underscores the flexibility and scalability of the proposed node-based method, making it more suitable for real-world applications where instance sizes may vary widely.

Furthermore, our proposed method proves to be effective across both small-sized and large-sized TSP instances. Even without 2-opt post-processing, the adapted model can be equally good or even outperform a number of recent ML-based methods. For example, applying the above-mentioned baseline, TabPFN-v2 demonstrates a performance gap on TSP-500 that is almost 10% lower than the results reported by the authors of Eformer [18]. When 2-opt is applied, the overall solution quality can be improved considerably, and the performance gap between TabPFN-v2 and other ML-based models becomes negligible, illustrated by a gap difference of less than 1% between TabPFN-v2 and Fast-T2T.

Beyond the benchmark comparisons, the statistical analyses presented in Section 4.3 showed that as the number of nodes increases, the performance of the proposed method becomes increasingly consistent across independently generated benchmark instances. In addition, as the instance size increases, the randomness associated with the benchmark generation has smaller relative influence on the observed performance, leading to a more consistent performance on larger TSP instances.

Despite its significant advantages, particularly in terms of training time and data efficiency, the proposed method may not outperform some existing SOTA methods, which benefit from substantially larger training budgets, such as Fast-T2T [4].

5. Conclusions

This paper presents a near-training-free method to handle combinatorial route optimization (CRO) problems, with the Travelling Salesman Problem (TSP) as a representative case, leveraging advantageous capabilities of TabPFN-v2, a newly designed foundation model. To the best of the authors' knowledge, this can be the first application of its kind in using TabPFN-v2 to deal with CRO problems in general and TSP in particular.

The core of the proposed method includes two features. First, a novel node-based encoding method is used to avoid expressiveness issues and the lack of local spatial structure as in whole-instance encoding methods. Second, to enhance scalability, this method is applied in combination with the node-predicting adaptation strategy to predict the location of the next node in constructing the entire TSP route. The comprehensive experiment and analysis results acquired by the study demonstrate that the node-based approach, in combination with TabPFN-v2, can be effectively applied to routing-related tasks, achieving performance competitive with, or even better than, existing state-of-the-art (SOTA) Machine Learning (ML)-based solvers.

Three key strengths should be highlighted in the proposed approach in this study. First, unlike most learning-based methods, this approach requires minimal training, being able to adapt to TSP within minutes using only a single sample and to complete the inference process during evaluation in seconds, leading to rapid adaptation and enhanced data efficiency. Second, the node-based representation allows the model to achieve better generalization across varying instance sizes and to reduce the performance degradation commonly seen in whole-instance encoding methods. Third, the proposed node-predicting adaptation strategy enables TabPFN-v2 to efficiently generate TSP tours via a two-stage procedure consisting of parallel inference, which may incur significant GPU computational cost, and subsequent sequential, step-wise decoding.

Given these attributes, the proposed method has the potential to eliminate the need for extensive retraining or iterative optimization, significantly reduce computational and data requirements while maintaining competitive solution quality, mitigate performance degradation as the instance size increases, and enable rapid inference suitable for real-time applications. Furthermore, when combined with simple 2-opt refinement, the proposed method achieves performance comparable to that of other models. These findings highlight a promising direction for extending the application of powerful pre-trained foundation models such as TabPFN-v2 to a broader range of structured and combinatorial optimization problems, particularly in scenarios where training resources are limited or rapid deployment is required.

Future research could investigate alternative methods for capturing the state of visited and unvisited nodes, structural representations of partial tours or previously visited paths, and statistical descriptors of node location and spatial density distributions directly into foundation models to enhance performance. Another direction is to explore whether larger foundation models, especially those with strong reasoning capabilities [54], can improve performance without sacrificing their near-training-free and data-efficient nature or can expand to other route optimization problems, such as the vehicle routing problem (VRP) and its variants. In addition, the authors believe a comprehensive runtime analysis of the inference, decoding, and post-processing stages under standardized hardware conditions would be beneficial for practical deployment.

Author Contributions: Conceptualization, N.G.H.V., Y.T., R.L., Y.Y., H.M., K.W. and G.G.W.; methodology, N.G.H.V.; software, N.G.H.V.; validation, Y.T. and Y.Y.; formal analysis, Y.T.; investigation, N.G.H.V.; writing—original draft preparation, N.G.H.V.; writing—review and editing, N.G.H.V., Y.Y. and G.G.W.; supervision, H.M. and G.G.W.; project administration, G.G.W.; funding acquisition, G.G.W. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by MITACS grant number IT40086. The APC was funded by MDPI.

Informed Consent Statement: Not applicable.

Data Availability Statement: The testing dataset for this study is available at https://github.com/chaitjo/learning-tsp (accessed on 28 July 2026) and https://github.com/SaneLYX/TSP_Att-GCRN-MCTS (accessed on 28 July 2026).

Acknowledgments: The research grant from MITACS (project number: IT40086) is gratefully acknowledged. The waiver of APC from MDPI is also appreciated.

Conflicts of Interest: The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

1. Cappart, Q.; Chételat, D.; Khalil, E.B.; Lodi, A.; Morris, C.; Veličković, P. Combinatorial Optimization and Reasoning with Graph Neural Networks. *J. Mach. Learn. Res.* **2023**, *24*, 5922–5982.
2. Qiu, R.; Sun, Z.; Yang, Y. DIMES: A Differentiable Meta Solver for Combinatorial Optimization Problems. In *Proceedings of the Advances in Neural Information Processing Systems*; Koyejo, S., Mohamed, S., Agarwal, A., Belgrave, D., Cho, K., Oh, A., Eds.; Curran Associates, Inc.: Red Hook, NY, USA, 2022; Volume 35, pp. 25531–25546.
3. Sun, Z.; Yang, Y. DIFUSCO: Graph-Based Diffusion Solvers for Combinatorial Optimization. In *Proceedings of the Advances in Neural Information Processing Systems*; Oh, A., Naumann, T., Globerson, A., Saenko, K., Hardt, M., Levine, S., Eds.; Curran Associates, Inc.: Red Hook, NY, USA, 2023; Volume 36, pp. 3706–3731.
4. Li, Y.; Guo, J.; Wang, R.; Zha, H.; Yan, J. Fast T2T: Optimization Consistency Speeds Up Diffusion-Based Training-to-Testing Solving for Combinatorial Optimization. In *Proceedings of the Advances in Neural Information Processing Systems*; Globerson, A., Mackey, L., Belgrave, D., Fan, A., Paquet, U., Tomczak, J., Zhang, C., Eds.; Curran Associates, Inc.: Red Hook, NY, USA, 2024; Volume 37, pp. 30179–30206.
5. Alanzi, E.; Menai, M.E.B. Solving the Traveling Salesman Problem with Machine Learning: A Review of Recent Advances and Challenges. *Artif. Intell. Rev.* **2025**, *58*, 267. [CrossRef]
6. Saller, S.; Koehler, J.; Karrenbauer, A. A Survey on Approximability of Traveling Salesman Problems Using the TSP-T3CO Definition Scheme. *Ann. Oper. Res.* **2025**, *351*, 2129–2190. [CrossRef]
7. Jaillet, P.; Lu, X. Online Traveling Salesman Problems with Service Flexibility. *Networks* **2011**, *58*, 137–146. [CrossRef]
8. Vesselinova, N.; Steinert, R.; Perez-Ramirez, D.F.; Boman, M. Learning Combinatorial Optimization on Graphs: A Survey with Applications to Networking. *IEEE Access* **2020**, *8*, 120388–120416. [CrossRef]
9. Johnson, O.; Liu, J. A Traveling Salesman Approach for Predicting Protein Functions. *Source Code Biol. Med.* **2006**, *1*, 3. [CrossRef] [PubMed]
10. Crişan, G.C.; Pintea, C.M.; Pop, P.C.; Matei, O. Economical Connections between Several European Countries Based on TSP Data. *Log. J. IGPL* **2020**, *28*, 33–44. [CrossRef]
11. Singh, S.; Singh, A.; Kapil, S.; Das, M. Utilization of a TSP Solver for Generating Non-Retractable, Direction Favouring Toolpath for Additive Manufacturing. *Addit. Manuf.* **2022**, *59*, 103126. [CrossRef]
12. Held, M.; Karp, R.M. A Dynamic Programming Approach to Sequencing Problems. *J. Soc. Ind. Appl. Math.* **1962**, *10*, 196–210. [CrossRef]
13. Yang, H.; Zhao, M.; Yuan, L.; Yu, Y.; Li, Z.; Gu, M. Memory-Efficient Transformer-Based Network Model for Traveling Salesman Problem. *Neural Netw.* **2023**, *161*, 589–597. [CrossRef] [PubMed]
14. Balas, E.; Toth, P. Branch and Bound Methods for the Traveling Salesman Problem. *Math. Comput. Sci.* **1983**. [CrossRef]
15. Applegate, D.; Bixby, R.; Chvátal, V.; Cook, W. Concorde TSP Solver 2003. Available online: https://www.math.uwaterloo.ca/tsp/concorde/downloads/downloads.htm (accessed on 28 July 2026).
16. Kool, W.; van Hoof, H.; Welling, M. Attention, Learn to Solve Routing Problems! In Proceedings of the International Conference on Learning Representations (ICLR), New Orleans, LA, USA, 6–9 May 2019.

17. Kwon, Y.-D.; Choo, J.; Kim, B.; Yoon, I.; Gwon, Y.; Min, S. POMO: Policy Optimization with Multiple Optima for Reinforcement Learning. In *Proceedings of the Advances in Neural Information Processing Systems*; Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M.F., Lin, H., Eds.; Curran Associates, Inc.: Red Hook, NY, USA, 2020; Volume 33, pp. 21188–21198.
18. Meng, D.; Cao, Z.; Wu, Y.; Hou, Y.; Ge, H.; Zhang, Q. EFormer: An Effective Edge-Based Transformer for Vehicle Routing Problems. In *Proceedings of the Thirty-Fourth International Joint Conference on Artificial Intelligence*; International Joint Conferences on Artificial Intelligence Organization: Montreal, Canada, 2025; pp. 8582–8590.
19. Zhao, C.S.; Wong, L.P. A Transformer-Based Structure-Aware Model for Tackling the Traveling Salesman Problem. *PLoS ONE* **2025**, *20*, e0319711. [CrossRef] [PubMed]
20. Lu, Y.; Hao, J.K.; Wu, Q. Solving the Clustered Traveling Salesman Problem via Traveling Salesman Problem Methods. *PeerJ Comput. Sci.* **2022**, *8*, e972. [CrossRef] [PubMed]
21. Rego, C.; Gamboa, D.; Glover, F.; Osterman, C. Traveling Salesman Problem Heuristics: Leading Methods, Implementations and Latest Advances. *Eur. J. Oper. Res.* **2011**, *211*, 427–441. [CrossRef]
22. Helsgaun, K. *An Extension of the Lin-Kernighan-Helsgaun TSP Solver for Constrained Traveling Salesman and Vehicle Routing Problems*; Technical Report; Roskilde Universitet: Roskilde, Denmark, 2017.
23. Skinderowicz, R. Improving Ant Colony Optimization Efficiency for Solving Large TSP Instances. *Appl. Soft Comput.* **2022**, *120*, 108653. [CrossRef]
24. Joshi, C.K.; Cappart, Q.; Rousseau, L.M.; Laurent, T. Learning TSP Requires Rethinking Generalization. In *Proceedings of the Leibniz International Proceedings in Informatics, LIPIcs*; Michel, L.D., Ed.; Schloss Dagstuhl—Leibniz-Zentrum fur Informatik GmbH, Dagstuhl Publishing: Wadern, Germany, 2021; Volume 210, pp. 33:1–33:21.
25. Bommasani, R.; Hudson, D.A.; Adeli, E.; Altman, R.; Arora, S.; von Arx, S.; Bernstein, M.S.; Bohg, J.; Bosselut, A.; Brunskill, E.; et al. On the Opportunities and Risks of Foundation Models. *arXiv* **2022**, arXiv:2108.07258.
26. Touvron, H.; Lavril, T.; Izacard, G.; Martinet, X.; Lachaux, M.-A.; Lacroix, T.; Rozière, B.; Goyal, N.; Hambro, E.; Azhar, F.; et al. LLaMA: Open and Efficient Foundation Language Models. *arXiv* **2023**, arXiv:2302.13971.
27. Das, A.; Kong, W.; Sen, R.; Zhou, Y. A Decoder-Only Foundation Model for Time-Series Forecasting. In *Proceedings of the 41st International Conference on Machine Learning*; JMLR.org: Cambridge, MA, USA, 2024.
28. Brown, T.; Mann, B.; Ryder, N.; Subbiah, M.; Kaplan, J.D.; Dhariwal, P.; Neelakantan, A.; Shyam, P.; Sastry, G.; Askell, A.; et al. Language Models Are Few-Shot Learners. In *Proceedings of the Advances in Neural Information Processing Systems*; Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M.F., Lin, H., Eds.; Curran Associates, Inc.: Red Hook, NY, USA, 2020; Volume 33, pp. 1877–1901.
29. Schneider, J.; Meske, C.; Kuss, P. Foundation Models. *Bus. Inf. Syst. Eng.* **2024**, *66*, 221–231. [CrossRef]
30. Zhou, C.; Li, Q.; Li, C.; Yu, J.; Liu, Y.; Wang, G.; Zhang, K.; Ji, C.; Yan, Q.; He, L.; et al. A Comprehensive Survey on Pretrained Foundation Models: A History from BERT to ChatGPT. *Int. J. Mach. Learn. Cybern.* **2024**, *16*, 9851–9915. [CrossRef]
31. Radford, A.; Narasimhan, K.; Salimans, T.; Sutskever, I. Improving Language Understanding by Generative Pre-Training. 2018. Available online: https://cdn.openai.com/research-covers/language-unsupervised/language_understanding_paper.pdf (accessed on 28 July 2026).
32. Goswami, M.; Szafer, K.; Choudhry, A.; Cai, Y.; Li, S.; Dubrawski, A. MOMENT: A Family of Open Time-Series Foundation Models. In *Proceedings of the 41st International Conference on Machine Learning*; Salakhutdinov, R., Kolter, Z., Heller, K., Weller, A., Oliver, N., Scarlett, J., Berkenkamp, F., Eds.; JMLR.org: Cambridge, MA, USA, 2024.
33. Hollmann, N.; Müller, S.; Purucker, L.; Krishnakumar, A.; Körfer, M.; Hoo, S.B.; Schirrmeister, R.T.; Hutter, F. Accurate Predictions on Small Data with a Tabular Foundation Model. *Nature* **2025**, *637*, 319–326. [CrossRef] [PubMed]
34. Vu, N.G.H.; Wang, K.; Wang, G.G. Effective Prompting with ChatGPT for Problem Formulation in Engineering Optimization. *Eng. Optim.* **2025**, *57*, 3677–3694. [CrossRef]
35. Mazurowski, M.A.; Dong, H.; Gu, H.; Yang, J.; Konz, N.; Zhang, Y. Segment Anything Model for Medical Image Analysis: An Experimental Study. *Med. Image Anal.* **2023**, *89*, 102918. [CrossRef] [PubMed]
36. Neumann, A.T.; Yin, Y.; Sowe, S.; Decker, S.; Jarke, M. An LLM-Driven Chatbot in Higher Education for Databases and Information Systems. *IEEE Trans. Educ.* **2025**, *68*, 103–116. [CrossRef]
37. Hollmann, N.; Müller, S.; Eggensperger, K.; Hutter, F. TabPFN: A Transformer That Solves Small Tabular Classification Problems in a Second. In Proceedings of the International Conference on Learning Representations (ICLR), Kigali, Rwanda, 1–5 May 2023.
38. Hoo, S.B.; Müller, S.; Salinas, D.; Hutter, F. From Tables to Time: How TabPFN-v2 Outperforms Specialized Time Series Forecasting Models. *arXiv* **2025**, arXiv:2501.02945v3.
39. Saito, T.; Otake, Y.; Wu, S. Tabular Foundation Model for GEOAI Benchmark Problems BM/AirportSoilProperties/2/2025. *arXiv* **2025**, arXiv:2509.03191.
40. Luo, J.; Yuan, Y.; Xu, S. TIME: TabPFN-Integrated Multimodal Engine for Robust Tabular-Image Learning. *arXiv* **2025**, arXiv:2506.00813.
41. Larrañaga, P.; Kuijpers, C.M.H.; Murga, R.H.; Inza, I.; Dizdarevic, S. Genetic Algorithms for the Travelling Salesman Problem: A Review of Representations and Operators. *Artif. Intell. Rev.* **1999**, *13*, 129–170. [CrossRef]

42. Laporte, G. The Traveling Salesman Problem: An Overview of Exact and Approximate Algorithms. *Eur. J. Oper. Res.* **1992**, *59*, 231–247. [CrossRef]
43. Carpaneto, G.; Fischetti, M.; Toth, P. New Lower Bounds for the Symmetric Travelling Salesman Problem. *Math. Program.* **1989**, *45*, 233–254. [CrossRef]
44. Frieze, A.; Karp, R.M.; Reed, B. When Is the Assignment Bound Tight for the Asymmetric Traveling-Salesman Problem? *SIAM J. Comput.* **1995**, *24*, 484–493. [CrossRef]
45. Bektas, T. The Multiple Traveling Salesman Problem: An Overview of Formulations and Solution Procedures. *Omega* **2006**, *34*, 209–219. [CrossRef]
46. Müller, S.; Hollmann, N.; Arango, S.P.; Grabocka, J.; Hutter, F. Transformers Can Do Bayesian Inference. In Proceedings of the International Conference on Learning Representations (ICLR), Online, 25–29 April 2022.
47. Ruiz-Villafranca, S.; Roldán-Gómez, J.; Gómez, J.M.C.; Carrillo-Mondéjar, J.; Martinez, J.L. A TabPFN-Based Intrusion Detection System for the Industrial Internet of Things. *J. Supercomput.* **2024**, *80*, 20080–20117. [CrossRef]
48. Alves, F.; Pacheco, F.; Rocha, A.M.A.C.; Pereira, A.I.; Leitão, P. Solving a Logistics System for Vehicle Routing Problem Using an Open-Source Tool. In *Computational Science and Its Applications--ICCSA 2021*; Gervasi, O., Murgante, B., Misra, S., Garau, C., Blečić, I., Taniar, D., Apduhan, B.O., Rocha, A.M.A.C., Tarantino, E., Torre, C.M., Eds.; Springer International Publishing: Cham, Switzerland, 2021; Volume 12953, pp. 397–412.
49. Google Introduction to Optimization. Available online: https://developers.google.com/optimization/introduction (accessed on 28 July 2026).
50. Fu, Z.-H.; Qiu, K.-B.; Zha, H. Generalize a Small Pre-Trained Model to Arbitrarily Large TSP Instances. *Proc. AAAI Conf. Artif. Intell.* **2021**, *35*, 7474–7482. [CrossRef]
51. Li, Y.; Guo, J.; Wang, R.; Yan, J. T2T: From Distribution Learning in Training to Gradient Search in Testing for Combinatorial Optimization. In *Proceedings of the Advances in Neural Information Processing Systems*; Oh, A., Naumann, T., Globerson, A., Saenko, K., Hardt, M., Levine, S., Eds.; Curran Associates, Inc.: Red Hook, NY, USA, 2023; Volume 36, pp. 50020–50040.
52. Chang, H.J.; Lee, M.C. Applying Computer Simulation to Analyze the Normal Approximation of Binomial Distribution. *J. Comput.* **2017**, *28*, 116–131. [CrossRef]
53. Zhou, C.; Lin, X.; Wang, Z.; Zhang, Q. Learning to Reduce Search Space for Generalizable Neural Routing Solver. *arXiv* **2025**, arXiv:2503.03137.
54. Sun, J.; Zheng, C.; Xie, E.; Liu, Z.; Chu, R.; Qiu, J.; Xu, J.; Ding, M.; Li, H.; Geng, M.; et al. A Survey of Reasoning with Foundation Models: Concepts, Methodologies, and Outlook. *ACM Comput. Surv.* **2025**, *57*, 1–43. [CrossRef]