

RESEARCH ARTICLE OPEN ACCESS

A Supervised Learning Framework for Accelerating Solvers and Metaheuristics in Routing Problems

Johannes Gückel¹  | Stefan Voigt^{1,2}  | Pirmin Fontaine¹ 

¹Ingolstadt School of Management & Mathematical Institute for Machine Learning and Data Science, Catholic University of Eichstätt-Ingolstadt, Ingolstadt, Germany | ²Frankfurt University of Applied Sciences, Frankfurt am Main, Germany

Correspondence: Pirmin Fontaine (pirmin.fontaine@ku.de)

Received: 3 March 2026 | **Revised:** 8 September 2026 | **Accepted:** 27 September 2026

Keywords: machine learning | metaheuristics | routing problems

ABSTRACT

Routing problems such as the traveling salesman problem (TSP) and the capacitated vehicle routing problem (CVRP) are computationally challenging due to the exponential growth of their feasible solution spaces with increasing instance size. Established optimization approaches, including exact algorithms and (meta)heuristics, often face scalability limits as instance size increases. In this study, we introduce a supervised machine learning (ML) framework that assigns a confidence score to each edge, indicating the likelihood of appearing in high-quality solutions. Using these predictions, we develop systematic strategies to reduce the search space by pruning low-relevance edges. We integrate the ML-based edge classifier into a mathematical optimization solver and a metaheuristic as a preprocessing step and evaluate its effectiveness in a comprehensive numerical study on instances with up to 1,000 nodes. On the tested instances, the proposed ML-based preprocessing removed approximately up to 90% of candidate edges and reduced solver runtimes, while preserving the baseline-optimal solution in all or nearly all cases for conservative parameter settings. For instances not solved to optimality within the time limit, average optimality gaps decrease by up to 1.7% (TSP) and over 7% (CVRP). Within a state-of-the-art metaheuristic, learned pruning decreases the time required to reach solutions of equal or better quality by up to 20–25 s under a 60-second time limit, and improves average solution quality by up to 0.75% (TSP) and 0.86% (CVRP) on large-scale instances.

1 | Introduction

Routing problems have been central to research, teaching, and practice in operations research for decades. They belong to the most extensively studied problem classes in combinatorial optimization, and substantial progress has been made on both exact mathematical formulations and advanced heuristic algorithms [1]. These methods have steadily improved in their ability to deliver high-quality solutions with reasonable computational effort. In recent years, a complementary research direction has emerged, driven by advances in machine learning (ML). Modern ML methods have been increasingly applied to

support decision-making, estimate problem parameters, or predict structural properties of transportation and routing problems [2, 3]. Although ML alone has not yet matched the performance of state-of-the-art optimization techniques on large-scale routing problems, a natural research question arises: *Can ML enhance classical optimization by providing problem-specific guidance or reduction mechanisms that improve computational efficiency without compromising solution quality?*

Routing problems are typically modeled on graphs comprising nodes and edges. Even moderately sized instances already lead to extremely large search spaces. However, solutions contain only

This is an open access article under the terms of the [Creative Commons Attribution](https://creativecommons.org/licenses/by/4.0/) License, which permits use, distribution and reproduction in any medium, provided the original work is properly cited.

© 2026 The Author(s). *Networks* published by Wiley Periodicals LLC.

a small fraction of all available edges. From a structural perspective, many edges appear implausible candidates for inclusion in high-quality solutions. Identifying and excluding such edges in advance offers significant potential to accelerate both exact and (meta)heuristic methods.

To address this challenge, we propose an ML-based preprocessing framework. The central idea is to employ supervised learning to assign a confidence score to each edge, reflecting how likely it is to appear in a high-quality solution. These confidence scores can then be used to reduce the search space for mathematical solvers or to guide the search process within heuristic algorithms. Several recent studies have demonstrated the potential of ML-driven reduction and prediction techniques in related combinatorial optimization settings (e.g., [4–6]). Building on these insights, our approach integrates predictive information directly into classical optimization routines. The contributions of this study are fourfold:

- We introduce a general supervised learning framework to extract structural information from high-quality routing solutions by assigning to each edge a confidence score that reflects the model's confidence that the edge appears in high-quality solutions.
- We analyze several state-of-the-art ML algorithms and identify those most suitable for this classification task.
- We develop two strategies for embedding the learned confidence scores into mixed-integer programming formulations via variable reduction and into metaheuristic frameworks by introducing a data-driven generalization of granular neighborhoods. For mixed-integer programming formulations, the proposed variable reduction constitutes a heuristic preprocessing strategy that reduces the search space prior to optimization.
- We demonstrate significant performance gains for both mixed-integer programming formulations and metaheuristics on the traveling salesman problem (TSP) and the capacitated vehicle routing problem (CVRP) and provide insights into the trade-offs between reduction intensity and search reliability.

The remainder of the paper is structured as follows. Section 2 introduces the formal problem definition for both the routing problem and the edge classification task. Section 3 reviews the relevant literature on routing optimization and ML applications in transportation and logistics. Section 4 describes our methodology, including feature engineering, ML models, and the integration of the confidence scores into established exact and metaheuristic solution approaches. Section 5 presents numerical results and evaluates both prediction performance and the effectiveness of incorporating the ML-based preprocessing step. Section 6 concludes the paper and outlines future research.

2 | Problem Description

This section introduces the general structure of routing problems (Section 2.1) and then discusses the edge classification task that

forms the basis for the proposed enhancements of solvers and metaheuristics (Section 2.2).

2.1 | Routing Problems

Routing problems are commonly modeled on a graph $\mathcal{G} = (\mathcal{N}, \mathcal{E})$, where $\mathcal{N}$ denotes the set of nodes and $\mathcal{E}$ denotes the set of edges. Each edge $(i, j) \in \mathcal{E}$ connects two nodes $i, j \in \mathcal{N}$ and is associated with a cost c_{ij} , which may represent distance, travel time, or any other relevant measure. The objective of a routing problem is generally to determine one or more routes that visit all required nodes at minimal total cost, where the cost of a route is given by the sum of the costs c_{ij} of the traversed edges. Using this notation, such routing problems can be formulated as mixed-integer linear programs. The full mathematical formulations of the models, which are also used in the numerical study in Section 5.4, are provided in A.

The literature on routing problems is extensive and includes many variants; see, for example, the surveys by Vidal et al. [7] and Braekers et al. [8]. Classical extensions include the vehicle routing problem with time windows [9], multi-depot variants [10], stochastic routing problems [11], two-echelon routing [12], and pickup-and-delivery problems [13], among many others. In this work, we focus on two fundamental problems that form the basis for many of these extensions: the TSP and the CVRP.

The TSP seeks the shortest Hamiltonian cycle on $\mathcal{G}$, that is, a route that visits each node exactly once and returns to the depot. The TSP is a prototypical $\mathcal{NP}$ -hard combinatorial optimization problem. The CVRP generalizes the TSP to the case of a vehicle fleet. Each customer node $i \in \mathcal{N} \setminus \{0\}$ has a demand d_i , and each vehicle has a maximum capacity Q . The task is to construct a set of routes starting and ending at the depot 0, such that every customer node is visited exactly once, the total demand on each route does not exceed Q , and the travel cost is minimized. Formally, each route $\mathcal{R}$ must satisfy

$$\sum_{i \in \mathcal{R}} d_i \leq Q,$$

and the objective is to minimize the sum of the route costs across the fleet.

2.2 | Classifying Edges in Routing Problems

In both the TSP and the CVRP, a solution can be represented by a set of traversed edges. Mathematical formulations typically employ binary decision variables of the form

$$x_{ij} = \begin{cases} 1, & \text{if edge } (i, j) \in \mathcal{E} \text{ is used in the solution,} \\ 0, & \text{otherwise,} \end{cases}$$

for all $(i, j) \in \mathcal{E}$. In the TSP, exactly $|\mathcal{N}|$ edges are selected to form a single Hamiltonian cycle. Assuming a complete graph, the number of potential edges in a TSP instance with $|\mathcal{N}|$ nodes is

$$|\mathcal{E}| = \frac{|\mathcal{N}|(|\mathcal{N}| - 1)}{2}.$$

For example, an instance with $|\mathcal{N}| = 50$ contains 1225 undirected edges, but only 50 edges (approximately 4%) are selected in a solution.

The situation is similar in the CVRP. If k routes are used in the optimal solution, then the number of edges selected equals $|\mathcal{N}| - 1 + k$, since each route forms a cycle that includes a return to the depot. Even for large CVRP instances, the number of edges in the solution remains small relative to the size of the complete graph.

These observations highlight that only a small subset of the available edges is relevant for the optimal solution. For some edges, it is intuitively clear that they are unlikely to be included; for others, the decision is less obvious. As the number of edges directly influences the computational effort of exact solvers and the neighborhood size of heuristic methods, reducing the candidate edge set offers a possibility to improve computational performance.

3 | Literature Review

This section reviews prior work that relates to the use of learned information in routing problems. We first summarize studies that have incorporated ML into transportation and routing contexts (Section 3.1), followed by research on ML-based reduction techniques in combinatorial optimization (Section 3.2), and research gaps (Section 3.3).

3.1 | Machine Learning in Transport and Logistics

Existing research integrating ML into transportation and logistics can broadly be grouped into three streams: predictive and approximation tasks, the embedding of ML into metaheuristic solution methods, and the incorporation into exact optimization procedures.

A first line of work applies ML to predictive and approximation tasks that support planning and decision-making in transportation and logistics. Examples include travel time prediction [14], energy consumption forecasts for electric vehicle routing [15], marginal routing and fulfillment cost estimation in delivery operations [16, 17], and second-stage outcome prediction in stochastic rail logistics models [18]. Additional applications involve distance approximation in routing problems [19] and the identification of hidden structural constraints [20]. ML has also been used to approximate computationally intractable measures. For example, Gückel and Fontaine [21] propose a learning-based Shapley value approximator for routing problems, enabling fast and accurate cost allocation in TSP and CVRP settings. A broader overview of ML in routing problems is provided by Bai et al. [2].

A second stream integrates ML directly within metaheuristic solution procedures. In this context, ML has been used for guiding search operators [22], deriving first-stage decisions [23], and learning constructive or improvement components [24, 25]. Arnold and Sörensen [26] investigate structural characteristics of high-quality VRP solutions using data-mining techniques and

show how such knowledge can be exploited within heuristic search procedures. Building on this idea, Arnold and Sörensen [27] develop a knowledge-guided local search framework for the CVRP that uses learned structural information to guide the search process more effectively. A related hybrid approach is proposed by Maia et al. [28], who integrate data mining into a genetic search framework for the CVRP to bias search toward promising solution structures. Similarly, Herdianto et al. [29] propose a feature-guided metaheuristic for the CVRP in which explainable ML models identify solution features correlated with high quality, and these insights are used to guide a neighborhood search. Applications to algorithm configuration and selection for routing problems are reported in Karimi-Mamaghan et al. [30] and Asín Achá et al. [31]. ML-enhanced large neighborhood search methods have also been developed for routing problems under uncertainty [32], and ML-based reoptimization strategies have been proposed to handle instance updates efficiently [33].

A third stream focuses on incorporating learned information into exact optimization procedures. ML has supported column generation by predicting promising columns or reducing the number of pricing iterations [34, 35]. Further enhancements include approximating Benders cuts [36] and learning improved branching strategies in mixed-integer programming [37, 38].

Overall, these studies indicate that ML-derived information can improve optimization processes when embedded in a structured manner.

3.2 | Machine Learning for Problem Reduction

A growing line of research investigates using ML to reduce the search space of combinatorial optimization problems. These approaches aim to prune decision variables or restrict search regions in advance in order to decrease computational effort.

Within routing-related contexts, Sun et al. [6] applied a support vector machine to identify nonpromising edges in the TSP, enabling variable reduction before optimization. Rashedi et al. [39] introduced an offline/online learning framework for aircraft recovery, using ML-based pruning to accelerate solution times while maintaining near-optimal solutions. Data-driven metaheuristic frameworks have also been developed to restrict search neighborhoods or generate high-quality initial solutions [40].

Related ideas have also been explored in other combinatorial optimization settings. Sun et al. [41] introduced an ML-driven problem reduction approach that leverages statistical features from sampled solutions to prune decision variables in graph-based optimization, achieving significant performance gains on the maximum weight clique problem. Graph-based learning has been used to accelerate exact optimization. Li et al. [42] employed a graph convolutional network (GCN) to guide tree search for $\mathcal{NP}$ -hard problems, and Ding et al. [43] used a GCN to predict binary decisions for local branching in mixed-integer programs. Huang et al. [44] use ML within column-and-row generation to eliminate poor delay decisions, thereby reducing model growth in an integrated air cargo recovery problem. A closely related idea is explored by Kouteck et al. [45], who use ML to rank pricing problems in branch-and-price,

thereby focusing computational effort on the most promising subproblems. Beyond learning-based approaches, recent work has proposed exact edge-elimination techniques for the TSP. For example, Cook et al. [46] develop a combinatorial elimination procedure that provably removes edges that cannot belong to any optimal tour while preserving optimality. In contrast, our work investigates a learning-based preprocessing strategy that does not provide such guarantees but is applicable in a broader data-driven optimization framework.

3.3 | Research Gaps

Although existing work demonstrates the potential of ML-derived information to support optimization, several gaps remain relevant to routing problems:

- It is not yet well understood how accurately edge appearance in high-quality routing solutions can be predicted using supervised ML, nor which model classes perform best in this context.
- The integration of ML-based reduction mechanisms into mathematical solvers and state-of-the-art metaheuristics remains limited. Systematic studies quantifying the impact of ML-based preprocessing on runtime and solution quality are scarce.
- Existing studies often focus on predicting specific structural components within narrowly defined problem formulations, rather than learning decision-variable relevance in fundamental routing problems and systematically integrating such predictions into solution methods.

This study addresses these gaps by investigating the predictability of edge appearance in high-quality routing solutions and by evaluating how ML-based preprocessing can enhance both exact optimization methods and heuristic frameworks.

4 | Methodology

We begin by outlining the overall structure of our approach (Section 4.1). We then describe the feature set constructed for each edge (Section 4.2) and the ML models used for edge classification (Section 4.3). Finally, we show how the resulting confidence scores can be incorporated into mathematical optimization solvers (Section 4.4) and a state-of-the-art metaheuristic (Section 4.5).

4.1 | Overview

Our methodology consists of three stages:

- generating and solving routing problem instances,
- training predictive models for edge classification, and
- integrating the resulting scores into optimization and metaheuristic solution methods.

Algorithm 1 summarizes the overall ML-based preprocessing framework.

ALGORITHM 1 | ML-based preprocessing framework.

Input: Set of instances $\mathcal{T}^{all}$
// Offline training phase (executed once)
Partition $\mathcal{T}^{all}$ into $\mathcal{T}^{train}$ and $\mathcal{T}^{test}$
for $t \in \mathcal{T}^{all}$ **do**
 $s(t) \leftarrow \text{solveRoutingProblem}(t)$
 for $(i, j) \in \mathcal{E}(t)$ **do**
 $f(i, j) \leftarrow \text{computeFeatures}(t, (i, j))$ // Section 4.2
 $l(i, j) \leftarrow \text{labelEdge}(s(t), (i, j))$
trainData $\leftarrow \text{assembleDataset}(\{f(i, j), l(i, j) \mid (i, j) \in \mathcal{E}(t), t \in \mathcal{T}^{train}\})$
testData $\leftarrow \text{assembleDataset}(\{f(i, j), l(i, j) \mid (i, j) \in \mathcal{E}(t), t \in \mathcal{T}^{test}\})$
for $m \in \mathcal{M}$ **do**
 train(m , trainData)
 evaluate(m , testData)
 $m^* \leftarrow \text{selectBestModel}(\mathcal{M})$
// Online prediction phase (for each new instance)
 $t' \leftarrow \text{newRoutingInstance}$
for $(i, j) \in \mathcal{E}(t')$ **do**
 $f(i, j) \leftarrow \text{computeFeatures}(t', (i, j))$
 $\hat{x}_{ij} \leftarrow \text{predict}(m^*, f(i, j))$
solveWithPredictions(t' , $\{\hat{x}_{ij}\}$) // Sections 4.4–4.5

We begin with a set of routing instances $\mathcal{T}^{all}$, which is partitioned into a training set $\mathcal{T}^{train}$ and a test set $\mathcal{T}^{test}$. Each instance $t \in \mathcal{T}^{all}$ is solved, yielding a reference solution $s(t)$. Based on this solution, we construct an edge-level dataset: for every edge $(i, j) \in \mathcal{E}(t)$, we compute a feature vector $f(i, j)$ that captures relevant structural characteristics of the instance (see Section 4.2), and we assign a label $l(i, j)$ indicating whether the edge is part of a high-quality solution.

The resulting feature-label pairs are assembled into training and test datasets. A collection of ML models $\mathcal{M}$ is then trained on the training data to predict whether an edge appears in the solution. Each model is evaluated on the test set, and the best-performing model, denoted by m^* , is selected according to the chosen performance metric.

For a new and previously unseen routing instance t' , we compute the feature vector $f(i, j)$ for every edge $(i, j) \in \mathcal{E}(t')$ and apply the selected model m^* to obtain a confidence score $\hat{x}_{ij} \in [0, 1]$. These scores represent the model's confidence that an edge belongs to a high-quality solution.

Finally, the confidence scores are embedded into the solution process. Specifically, they are used as a preprocessing mechanism to reduce the search space within mathematical optimization solvers and metaheuristic algorithms, as detailed in Sections 4.4 and 4.5.

4.2 | Features

All features are defined with respect to an edge $(i, j) \in \mathcal{E}$ connecting two nodes i and j in the routing instance. We group the features into four categories: *Instance-based features*, *geometric features*, *local neighborhood features*, and *insertion-cost features*. For the CVRP, we additionally include *CVRP-specific features* based on node demand and vehicle capacity. Many

TABLE 1 | Feature description for predicting edge membership in high-quality TSP/CVRP solutions.

Feature name	Description	Expression (for node i)
<i>Instance-based features</i>		
No. of nodes	Total number of nodes in the instance.	$ \mathcal{N} $
x/y coordinate of i	X- and Y-coordinate of node i .	(x_i, y_i)
x/y coordinate percentile of i	Percentile rank of x_i and y_i among all node coordinates.	$\text{perc}(x_i; \{x_n : n \in \mathcal{N}\})$
Angle to depot of i	Polar angle of node i relative to the depot.	$\angle(i)$
<i>Edge geometry features</i>		
Edge cost	Travel cost associated with edge (i, j) .	c_{ij}
Edge cost percentile	Percentile of c_{ij} among all edge costs.	$\text{perc}(c_{ij}; \{c_{ab} : (a, b) \in \mathcal{E}\})$
Edge cost percentile (node i)	Percentile of c_{ij} among all outgoing costs from node i .	$\text{perc}(c_{ij}; \{c_{in} : n \in \mathcal{N} \setminus \{i\}\})$
Cost statistics of i	Mean, min, max, and std of costs from i to all other nodes.	$\{\text{mean}, \min, \max, \text{std}\} \{c_{in} : n \in \mathcal{N} \setminus \{i\}\}$
<i>Local neighborhood features</i>		
Closer-node count	Number of nodes with lower cost from i than the cost of (i, j) .	$ \{n \in \mathcal{N} \setminus \{i, j\} : c_{in} < c_{ij}\} $
m -neighbor count	Number of nodes for which i is the m -th closest node ($r_i(n)$: rank of i by cost from n), $m = 1, \dots, 5$.	$ \{n \in \mathcal{N} \setminus \{i\} : r_i(n) = m\} $
<i>Insertion-cost features</i>		
Mean insertion cost	Mean cost of inserting a node n between i and j , $\Delta_{ij}(n)$.	$\frac{1}{ \mathcal{N} -2} \sum_{k \in \mathcal{N} \setminus \{i, j\}} \Delta_{ij}(k)$
m -th min insertion cost	m -th smallest value of $\Delta_{ij}(n)$ over $n \in \mathcal{N} \setminus \{i, j\}$, $m = 1, \dots, 5$.	$\Delta_{ij}^{(m)}$
<i>CVRP-specific features</i>		
Depot-edge indicator	Whether edge (i, j) involves the depot.	$\mathbb{1}_{\{i=0 \vee j=0\}}$
Vehicle capacity	Vehicle capacity of the instance.	Q
Demand of i	Demand volume of node i .	d_i
m -neighbor demand of i	Sum of demands of the m nearest neighbors of i ($N_i(m)$: set of m nearest nodes to i), $m = 1, \dots, 5$.	$\sum_{k \in N_i(m)} d_k$

features depend on properties of the individual nodes and therefore produce two values: one for node i and one for node j . In Table 1, we show the mathematical expression only for node i ; the corresponding feature for node j is defined analogously.

Please note that the features are tailored to the TSP and CVRP considered in this study. For other routing variants, additional problem-specific features may be required in order to adequately exploit the underlying problem structure.

4.3 | Machine Learning Models

We employ established supervised ML models to predict whether an edge (i, j) is included in a high-quality solution. For each edge, a feature vector $f(i, j) \in \mathbb{R}^d$ is constructed as described in Section 4.2, and a binary label indicates whether the edge is part of a high-quality solution. Each model produces a real-valued output

$$\hat{x}_{ij} \in [0, 1], \quad (1)$$

which we interpret as a confidence score reflecting the model's confidence that edge (i, j) belongs to a high-quality solution. Because the majority class is undersampled during training (see Section 5.2), $\hat{x}_{ij}$ does not constitute a calibrated estimate of $\mathbb{P}(x_{ij} = 1 | f(i, j))$ under the original edge distribution; we therefore treat it throughout as a confidence/ranking score rather than a probability.

We deliberately select four model classes that represent complementary ML paradigms and levels of complexity. The k -Nearest Neighbors classifier serves as a simple, nonparametric baseline based on local similarity in feature space. Neural networks represent a highly expressive model class capable of capturing complex nonlinear interactions among edge features. The two tree-based ensemble methods reflect different ensemble principles: Random Forest relies on bagging and variance reduction, whereas XGBoost employs boosting and sequential error correction. The individual model classes are described below.

4.3.1 | k-Nearest Neighbors

As a simple nonparametric baseline, we employ the k -Nearest Neighbors (KNN) classifier. The predicted output is computed as

$$\hat{x}_{ij} = \frac{1}{k} \sum_{(a,b) \in \mathcal{N}_k(i,j)} x_{ab}, \quad (2)$$

where $\mathcal{N}_k(i,j)$ denotes the set of k nearest training edges in feature space according to a chosen distance metric. The k -NN classifier makes no parametric assumptions and bases its prediction solely on local similarity [47].

4.3.2 | Neural Networks

Neural networks (NN) approximate complex nonlinear mappings by composing affine transformations with nonlinear activation functions [48]. In this study, we employ feedforward neural networks with fully connected layers. Given an edge feature vector $f(i,j)$, the network computes

$$\hat{x}_{ij} = \sigma(W_L \phi_{L-1}(\dots \phi_1(W_1 f(i,j) + b_1)) + b_L), \quad (3)$$

where W_ℓ and b_ℓ denote the weights and biases of layer ℓ , $\phi_\ell(\cdot)$ are nonlinear activation functions, and $\sigma(\cdot)$ is the logistic sigmoid.

4.3.3 | Random Forest

Random Forest (RF) is an ensemble learning method that aggregates a collection of decorrelated decision trees [49]. Each tree is trained on a bootstrap sample of the training data, with additional randomness introduced through feature subsampling at each split. The predicted output is obtained by averaging the individual tree predictions,

$$\hat{x}_{ij} = \frac{1}{M} \sum_{m=1}^M T_m(f(i,j)), \quad (4)$$

where $T_m(\cdot)$ denotes the prediction of tree m .

4.3.4 | XGBoost

XGBoost (XGB) is a gradient-boosted decision tree method that constructs an additive model in a stage-wise manner [50]. The model represents the prediction for an edge (i,j) as a sum of tree outputs,

$$\hat{x}_{ij} = \sigma\left(\sum_{t=1}^T f_t(f(i,j))\right), \quad (5)$$

where $f_t(\cdot)$ denotes the prediction of tree t , T is the total number of trees, and $\sigma(\cdot)$ is the logistic sigmoid.

All models therefore output a real-valued confidence score in $[0, 1]$, rather than only a hard class label. In the remainder of this study, we exclusively use these confidence scores, as they provide more informative guidance for preprocessing and search-space reduction than binary classifications.

ALGORITHM 2 | Search space reduction using predicted scores.

Input: Instance t with nodes $\mathcal{N}$ and edges $\mathcal{E}$, confidence scores $\hat{x}_{ij}$ for all $(i,j) \in \mathcal{E}$, threshold θ
Output: Solution of instance t

$\mathcal{E}' \leftarrow \emptyset$
for $(i,j) \in \mathcal{E}$ **do**
 if $\hat{x}_{ij} > \theta$ **then**
 $\mathcal{E}' \leftarrow \mathcal{E}' \cup \{(i,j)\}$
 $S(t) \leftarrow \text{solve}(\mathcal{N}, \mathcal{E}')$
Return $S(t)$

4.4 | Embedding in Mathematical Solvers

The confidence scores produced by the predictive model can be integrated into mathematical optimization solvers in different ways. In this study, we consider two mechanisms. The first approach reduces the search space by pruning edges with a low predicted confidence score based on a predefined confidence threshold (Section 4.4.1).

The second approach also aims at reducing the search space but follows a nearest-neighbor principle. Specifically, only edges connecting a limited set of promising neighboring nodes are retained, where neighborhood relationships are not defined by geometric distance but by the predicted confidence scores assigned to the edges (Section 4.4.2).

It should be noted that the proposed edge-pruning strategy modifies the feasible region of the original optimization problem. Consequently, while it may substantially reduce computational effort, optimality of the resulting solution can no longer be guaranteed.

4.4.1 | Problem Reduction Based on Thresholds

The first mechanism exploits the predicted confidence scores to reduce the number of decision variables. For a given instance t , we generate the corresponding graph $\mathcal{G} = (\mathcal{N}, \mathcal{E})$ and compute feature representations for all edges. The best-performing predictive model m^* assigns to each edge $(i,j) \in \mathcal{E}$ a score

$$0 \leq \hat{x}_{ij} \leq 1,$$

which is interpreted as the ML model's confidence that the edge is included in a high-quality solution.

Edges with predicted scores below a predefined threshold θ are removed from the graph. The resulting reduced graph $(\mathcal{N}, \mathcal{E}')$ is then passed to the solver. The choice of θ is critical: small values lead to limited reduction and thus minor speed-ups, whereas large values increase the risk of excluding edges required for feasibility or optimality. The detailed procedure is illustrated in Algorithm 2.

4.4.2 | Problem Reduction Based on Nearest Neighbors

The second mechanism differs from the threshold-based pruning described in Section 4.4.1. Rather than pruning edges through

ALGORITHM 3 | Candidate-graph construction using ML-based nearest neighbors.

Input: Instance t with nodes $\mathcal{N}$ and edges $\mathcal{E}$, confidence scores $\hat{x}_{ij}$ for all $(i, j) \in \mathcal{E}$, neighborhood size n

Output: Solution of instance t

$\mathcal{E}' \leftarrow \{(i, j) \in \mathcal{E} \mid i = 0 \text{ or } j = 0\}$ if CVRP else $\mathcal{E}' \leftarrow \emptyset$ // Keep all depot edges for CVRP

for $i \in \mathcal{N}$ **do**

$\mathcal{N}_i \leftarrow \text{topN}(\{\hat{x}_{ij} \mid j \in \mathcal{N} \setminus \{i\}\}, n)$ // Select n
highest-scoring neighbors of i

$\mathcal{E}' \leftarrow \mathcal{E}' \cup \{(i, j) \mid j \in \mathcal{N}_i\}$

$S(t) \leftarrow \text{solve}(\mathcal{N}, \mathcal{E}')$

Return $S(t)$

a fixed threshold, it constructs a reduced graph by selectively inserting edges that connect each node to a limited set of neighboring nodes. In contrast to classical nearest-neighbor approaches, neighborhood relationships are not defined by geometric distance. Instead, neighbors are determined by the predicted confidence scores produced by the ML model. Specifically, for each node, only edges connecting it to its n most likely neighbors are included, where this likelihood is measured by the predicted confidence score associated with the corresponding edge. This approach therefore avoids the use of a global confidence threshold. Instead, it ensures that each node is connected to a fixed number of the most promising neighboring nodes according to the learned model, resulting in a reduced graph that preserves locally relevant edges. Analogous to the threshold parameter θ , the neighborhood size n controls the balance between search-space reduction and the risk of excluding edges necessary for feasibility or optimality: larger values of n yield a less aggressive reduction, whereas smaller values increase the risk of removing edges required for feasibility or optimality. The detailed procedure is illustrated in Algorithm 3. Please note that, for the CVRP, edges incident to the depot node 0 are always included in the reduced graph.

Unlike threshold-based pruning, this mechanism limits the number of retained edges per node to a fixed value. Consequently, it provides a stronger control over edge growth for large-scale instances and can be interpreted as a learned candidate-list approach. While threshold-based pruning reduces the edge set by removing low-confidence edges, the nearest-neighbor strategy explicitly bounds the number of candidate edges associated with each node.

4.5 | Embedding in Metaheuristics

The idea of restricting the search space within metaheuristics is closely related to the concept of granular neighborhoods introduced in granular tabu search [51]. In that framework, the neighborhood of local search operators is restricted based on geographic proximity, that is, only moves involving edges between nearby nodes are considered. This substantially reduces the number of candidate moves and focuses the search on structurally promising regions of the solution space. In our approach, we adopt a similar principle but replace geometric proximity by predicted scores. Specifically, we use the predicted confidence

scores $\hat{x}_{ij}$ to construct tailored candidate lists for local search operators. The candidate sets are constructed analogously to the two previously introduced concepts. Specifically, either (i) only edges with scores exceeding a threshold θ are retained (threshold-based pruning, Section 4.4.1), or (ii) only the top n highest-scoring edges are selected (neighbor-based pruning, Section 4.4.2). Under the neighbor-based reduction, for each node i , at least one edge (i, j) attaining the maximum score is guaranteed to be included. Consequently, move evaluations are restricted to edges that the ML model identifies as likely to belong to high-quality or optimal solutions. Both mechanisms can be interpreted as a data-driven generalization of granular neighborhoods. Whereas classical granular search relies on hand-crafted proximity rules typically based on distance metrics, our approach learns structural patterns directly from data.

As the underlying metaheuristic framework, we employ the hybrid genetic search (HGS, [52]) in its open-source implementation [53], a state-of-the-art algorithm for routing problems that already includes a granular neighborhood, restricting the search to distance-based nearest neighbors. The ML-based candidate restriction is incorporated without modifying the core components of HGS, such as population management, crossover, and local search strategies. The ML-based pruning is integrated into the move-generation phase of the metaheuristic. All neighborhood operators (e.g., edge exchanges or relocations) are restricted to consider only edges contained in the ML-derived candidate lists. Importantly, this does not alter the structure of the underlying operators. Instead, it reduces the set of admissible moves, thereby accelerating the search while maintaining its original logic. This integration ensures that the performance impact observed in the numerical study can be attributed directly to the learned edge pruning mechanism rather than to structural changes in the metaheuristic itself.

5 | Numerical Study

This section presents a comprehensive evaluation of the proposed methodology. We begin by describing the experimental setup and the benchmark instances used (Section 5.1). We then report the predictive performance of the ML models (Section 5.2) and provide insights into the most influential features for predicting the appearance of edges in high-quality routing solutions (Section 5.3). Next, we analyze the impact of the learned information when embedded into an exact solver (Section 5.4) and within a state-of-the-art metaheuristic (Section 5.5).

5.1 | Numerical Setting and Instance Description

The computational experiments are based on three sets of benchmark instances generated using the instance generator of Queiroga et al. [54]. The first set comprises instances with 20 to 100 nodes. For each instance size, 100 instances are generated, of which 90 are used to train and evaluate the ML models (Section 5.2) and 10 are reserved for solver experiments (Section 5.4). The second set consists of instances with 200 to 500 nodes. For each instance size, 20 instances are generated, with 17 used for training and validation and 3 reserved for metaheuristic evaluation (Section 5.5). The third set contains instances with

900 to 1000 nodes. For each instance size, 13 instances are generated, of which 10 are used for training and validation, and 3 are used for metaheuristic evaluation (Section 5.5). In the following, we refer to these three instance sets as *small*, *medium*, and *large*, respectively.

The instances have the following structural configuration: the depot is centrally located, nodes are uniformly distributed, and demands follow a low-variability distribution. The route-length parameter is set to *long* for the instances used with exact solvers (small instances) and to *very long* for those used with heuristics (medium and large instances). Further details on the generator and instance characteristics can be found in Queiroga et al. [54]. The same coordinates are used for the TSP and CVRP; for the TSP, demand information is simply ignored. All experiments are run on a workstation equipped with an AMD Ryzen 9 5950X processor (16 cores, 3.40 GHz) and 128 GB RAM. Python 3.10.10 is used for implementation, and Gurobi 12 is employed for solving the mathematical optimization models.

To generate the labels used for training the ML models, we solve the training instances with PyVRP [55], a state-of-the-art hybrid genetic search heuristic for vehicle routing problems. PyVRP is based on the HGS proposed by Vidal et al. [52] and has been shown to achieve state-of-the-art solution quality on standard benchmark instances while maintaining competitive computation times [53, 55]. We selected PyVRP as a practical compromise between label quality and computational effort, as generating optimal labels for the large number of training instances considered in this study would be computationally infeasible. A time limit of 60 s is imposed for *small* instances, 120 s for *medium* instances, and 300 s for *large* instances. Although heuristic-based labeling may introduce a small bias, prior research suggests that minor systematic biases in training data generally have only a marginal impact on predictive performance in routing problems [21]. To further assess the quality of the generated labels, we compare the PyVRP-generated labels with optimal labels on a subset of small TSP training instances containing 20–49 nodes that were solved to proven optimality by the exact solver. Among the 1150 instances for which an optimal solution was obtained, PyVRP identified the exact optimal tour in 1098 cases (95.48%). Across all optimally solved instances, the average recall of optimal edges was 99.66%, indicating that only a very small fraction of edges belonging to the optimal solution were missed by the heuristic. These results demonstrate that the discrepancy between heuristic and optimal labels is very limited for the considered instances and suggest that the generated training labels provide a highly accurate approximation of the true optimal edge structure.

Separate predictive models are estimated for the TSP and the CVRP, reflecting the structural differences between the two problem classes. For each problem type, hyperparameters are tuned independently using established tuning procedures for the respective model class, with the objective of identifying the best-performing configuration of each model. The resulting configurations are reported in B. The neural network model is implemented in TensorFlow Keras and tuned via Bayesian optimization using KerasTuner [56]. Random Forest and k-NN models are implemented using Scikit-learn [57], and XGBoost models use the XGBoost library [50], with hyperparameters selected through Bayesian search.

5.2 | Machine Learning Classification Performance

In this section, we evaluate the classification performance of the ML models introduced in Section 4.3. Besides overall predictive accuracy, we specifically analyze the distribution of confidence scores for true-negative and false-negative edges. This aspect is essential, as the effectiveness of pruning relies on removing a large share of true negatives while avoiding the elimination of edges that contribute to a high-quality solution. Since edges labeled as 0 significantly outnumber edges labeled as 1, using the original dataset would result in a strong class imbalance and could lead to overly optimistic accuracy values. To mitigate this issue, we randomly undersample the majority class before model training, resulting in a 2:1 ratio between label 0 and label 1 observations. As a consequence, the resulting confidence scores reflect relative confidence rather than calibrated probabilities under the original class distribution. The resulting rebalanced dataset is then split into training and test sets using an 80:20 partition.

Table 2 summarizes the classification accuracy across problem types (TSP, CVRP) and instance sizes. The results show that XGB and NN consistently outperform KNN and RF in all settings. Moreover, accuracy increases with problem size for both TSP and CVRP, indicating that the learning models benefit from richer problem structures in larger instances.

Although Table 2 reports standard classification accuracy, this metric should be interpreted with caution in the present setting. Since the ultimate objective is safe edge pruning, the most relevant performance measure is the trade-off between correctly pruned true-negative edges and incorrectly pruned positive edges, which is analyzed in the remainder of this section. Our primary interest lies in the ability of the ML models to safely prune a large fraction of true-negative edges from the search space. We therefore analyze, for varying confidence thresholds θ , the proportion of edges correctly pruned as true negatives and the proportion of incorrectly pruned as false negatives. Each point on a trade-off curve corresponds to one specific threshold θ : as θ increases from 10^{-6} to 0.3, pruning becomes more aggressive and we move along the curve from left to right.

For each threshold, the x-axis reports the share of negative edges whose confidence score falls below θ (and are thus safely pruned), while the y-axis reports the share of positive edges that are incorrectly pruned under the same threshold. Lower curves therefore indicate better pruning performance, as they achieve a larger reduction of non-high-quality edges while incurring fewer

TABLE 2 | Accuracy of the ML algorithms in %.

Problem	Instance set	KNN	RF	XGB	NN
TSP	Small	95.71	97.03	97.16	97.21
	Medium	98.39	98.71	98.96	98.86
	Large	98.70	98.85	98.92	98.90
CVRP	Small	94.40	95.49	95.75	95.74
	Medium	97.27	97.94	98.03	98.10
	Large	97.99	98.37	98.46	98.49

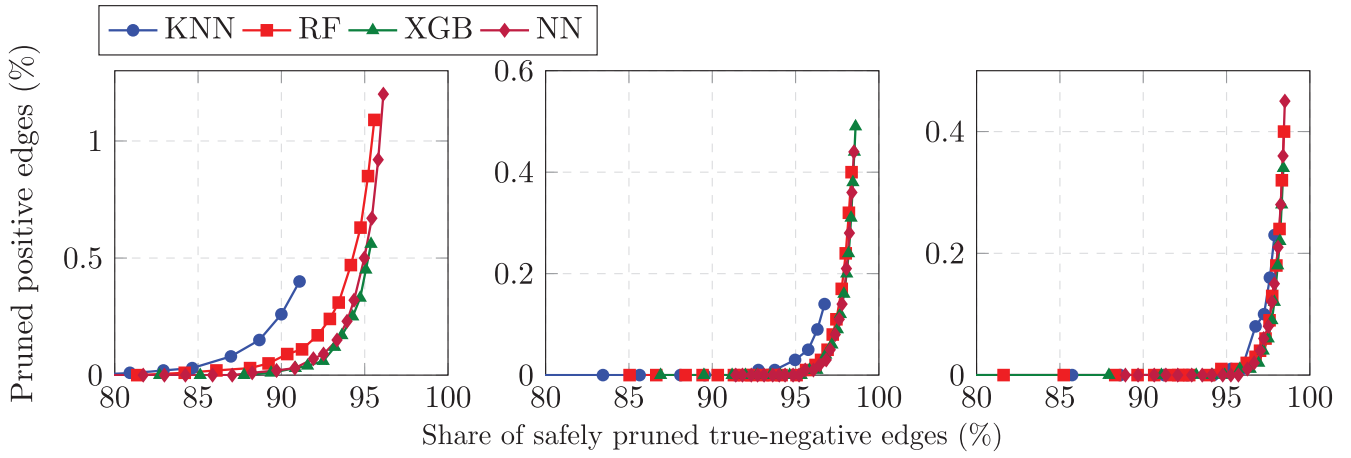


FIGURE 1 | Pruning trade-off for TSP instances across scales: *small* (left), *medium* (middle), and *large* (right).

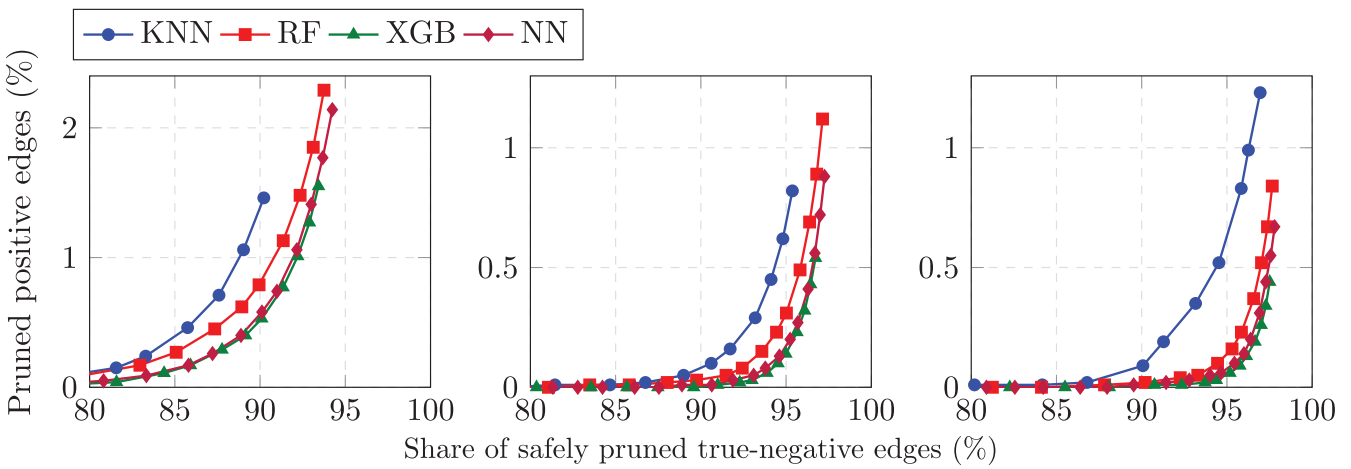


FIGURE 2 | Pruning trade-off for CVRP instances across scales: *small* (left), *medium* (middle), and *large* (right).

incorrectly pruned high-quality edges. The resulting curves in Figures 1 and 2 visualize the trade-off between search-space reduction through pruning and the risk of removing edges that belong to a high-quality solution.

Figures 1 and 2 show that XGB and NN consistently provide the strongest pruning capability, reliably identifying a large share of true-negative edges while keeping the misclassification of positive edges extremely low. In particular, for larger problem instances, both models can eliminate close to 90% of negative edges without discarding any edge that belongs to a high-quality solution (i.e., a positive edge). Moreover, the trade-off curves indicate that pruning performance improves with increasing instance size, and that results on TSP instances are generally superior to those obtained on CVRP instances.

These findings highlight that both XGB and NN are strong candidates for high-quality pruning. Since neither model is distinctly superior across all settings, we select the simpler XGB as a classifier and use its predicted confidence scores for all subsequent experiments. Moreover, the cost of generating confidence scores with XGB remains negligible even for large instances, averaging 0.48 s for 1000-node instances, 0.18 s for 500-node instances, and

0.03 s for 100-node instances. These times include the time for generating the features.

5.3 | Feature Importance

In this section, we analyze the relative importance of the features used in the predictive models and provide insights into the structural factors that influence whether an edge is included in a high-quality solution. To this end, we employ SHAP values (SHapley Additive exPlanations) [58], which interpret ML predictions by quantifying the marginal contribution of each feature based on Shapley value theory.

Figure 3 reports the SHAP values of the ten most influential features for both the TSP and the CVRP. For clarity and space considerations, feature importance is shown only for the XGBoost model and for instances with 200–500 nodes; however, additional experiments indicate that very similar importance patterns are observed across the other model classes.

Overall, the results show that features related to insertion costs and edge length consistently have the highest importance in both

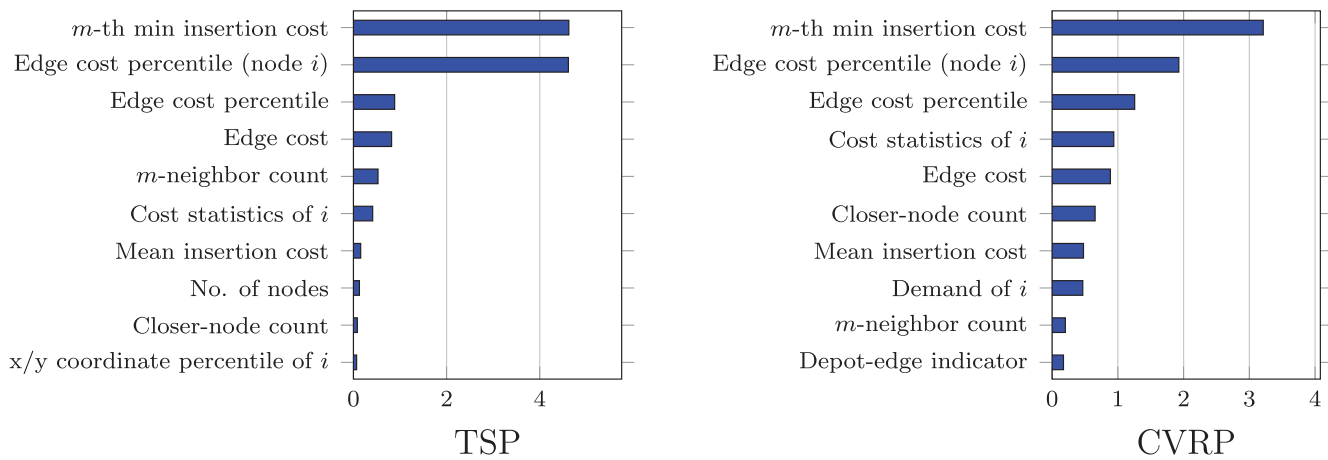


FIGURE 3 | Feature importance of the ten most important features. (a) *TSP*, (b) *CVRP*.

TABLE 3 | Number of baseline-optimal TSP instances that remain optimal under preprocessing.

Nodes	Instances	θ							n					
		0.005	0.01	0.03	0.05	0.1	0.2	0.3	13	11	9	7	5	3
20–50	296	296	295	288	280	266	229	206	295	295	293	287	235	58
51–80	209	209	209	204	198	184	148	118	209	209	209	198	142	14
81–100	53	53	53	52	50	47	38	29	53	53	53	50	37	3

problem settings. In the CVRP case, demand-related features and the depot-edge indicator also appear among the most influential predictors.

5.4 | Results Using an Exact Solver

In this section, we evaluate the performance of the solution approaches introduced in Section 4.4. The analysis is conducted on instances grouped by size, namely 20–50 nodes, 51–80 nodes, and 81–100 nodes. Note that ten instances for each number of nodes are used for testing the methodology. All instances are solved under a time limit of 180 s.

For each instance, we consider three solver configurations. First, we solve the original formulation on the full graph without any preprocessing (baseline). Second, we apply threshold-based preprocessing as described in Section 4.4.1, where edges with predicted confidence scores below a threshold θ are removed. We consider values of θ ranging from 0.005 to 0.3. Third, we apply the nearest-neighbor-based preprocessing proposed in Section 4.4.2, where only edges connecting each node to its n most promising neighbors, according to the predicted confidence scores are retained. For this approach, we vary n between 3 and 13. In addition to computational performance, the experiments explicitly investigate how often optimal solutions are lost due to preprocessing and how this behavior changes with increasing reduction intensity.

The results are presented separately for the TSP and the CVRP in Sections 5.4.1 and 5.4.2, respectively. Please note that even for a very small threshold of $\theta = 0.005$, already 90.28% of the edges for the instances with 100 nodes can be removed. In

addition, Sections 5.4.3 and 5.4.4 provide benchmark comparisons against alternative pruning strategies based on KNN confidence scores and simple distance-based candidate lists, while Section 5.4.5 examines generalization under structural distribution shifts between training and test instances.

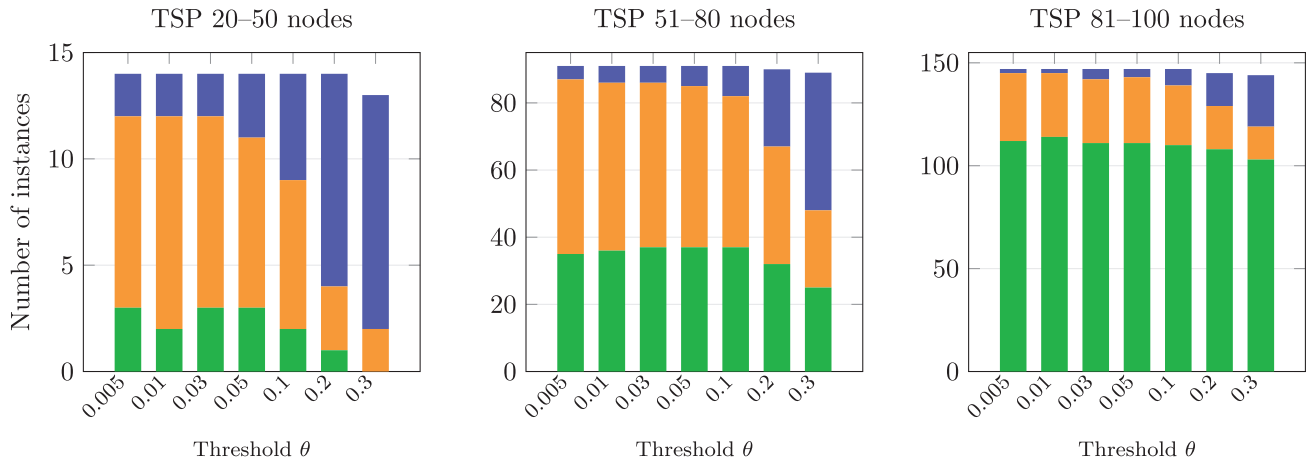
5.4.1 | TSP

We begin by examining whether the proposed preprocessing mechanisms affect the solver's ability to reach optimal solutions. Table 3 reports, for each instance size, the number of TSP instances that are solved to optimality by Gurobi when using the full graph without preprocessing. For each preprocessing configuration, the table then reports how many of these instances are still solved to optimality once preprocessing is applied. Since larger neighborhood sizes n correspond to less restrictive pruning, the n -columns are ordered from 13 to 3. This comparison allows us to directly assess whether the introduction of preprocessing leads to a loss of optimal solutions.

The results show that for small threshold values θ and for larger neighborhood sizes n , almost all instances that are solved to optimality in the baseline setting remain solvable to optimality when preprocessing is applied. For example, for $\theta \leq 0.01$, no loss of optimality is observed across all instance sizes, except for one case within the class of 20–50 nodes. For $n \geq 11$, optimality is preserved for nearly all baseline-optimal instances. As expected, the absolute number of instances solved to optimality decreases with increasing instance size. Overall, these results indicate that conservative preprocessing configurations reliably preserve optimality for the vast majority of instances. Even in cases where optimality is not preserved, the resulting deviations from the

TABLE 4 | Average solver runtimes (in seconds) for optimally solved TSP instances under different preprocessing configurations.

Nodes	Baseline	θ							n					
		0.005	0.01	0.03	0.05	0.1	0.2	0.3	13	11	9	7	5	3
20–50	10.7	5.2	4.5	2.9	2.1	1.3	0.7	0.5	8.7	8.5	8.3	6.1	4.3	0.9
51–80	48.2	18.0	15.1	10.6	8.1	5.5	3.6	3.6	30.1	28.0	24.5	18.9	13.1	4.9
81–100	86.8	22.8	18.5	14.4	11.9	8.8	5.6	4.9	35.8	35.1	29.5	24.2	20.7	14.0

**FIGURE 4** | Solution quality at the time limit under threshold-based preprocessing. Green bars indicate improved solutions, orange bars equal quality, and blue bars worse solutions.

baseline lower bound remain very small. As reported in C, the average relative gaps, computed with respect to the lower bound obtained without preprocessing, remain below approximately 1% for all θ -configurations and below 2% even for the most aggressive neighborhood settings.

We next examine whether the preprocessing mechanisms reduce solver runtimes for the subset of instances that are solved to optimality. Table 4 reports the average solver runtimes obtained for the baseline configuration without preprocessing and for the different preprocessing configurations.

The results reveal substantial reductions in solver runtimes across all preprocessing configurations. As expected, more aggressive problem reductions that correspond to larger values of θ and smaller values of n , lead to increasingly pronounced runtime improvements. For example, for instances with 81–100 nodes, the average runtime decreases from 86.8 s in the baseline setting to 4.9 s for $\theta = 0.3$, corresponding to a reduction of more than 94%.

Importantly, even for conservative preprocessing configurations, significant runtime reductions are achieved while preserving optimality in almost all cases. In particular, small values of θ and larger values of n combine substantial speed-ups with the preservation of optimality on almost all tested instances.

We next analyze instances that are not solved to optimality within the imposed time limit and evaluate whether the preprocessing configurations improve solution quality, as measured by the objective value obtained at termination. Figures 4 and 5 report,

for each preprocessing configuration, how many of the instances yield a better (green), equal (orange), or worse (blue) solution compared to the baseline without preprocessing. For very aggressive preprocessing settings ($\theta \geq 0.2$), feasibility is occasionally lost on a small number of instances; such cases are excluded from the comparison, which explains why the counts may not always sum to the total number of non-optimally solved instances.

The results indicate that for smaller instance sizes no clear effect can be observed, which is expected given the generally small optimality gaps and the limited number of non-optimally solved instances. For larger instances, however, a clear pattern emerges: the majority of instances are solved with equal or improved solution quality compared to the baseline without preprocessing. This holds across a wide range of preprocessing configurations. Notably, even for relatively aggressive configurations, improvements in solution quality are frequently observed, with the exception of the very restrictive setting $n = 3$, which consistently proves too limiting. Overall, these findings suggest that preprocessing not only accelerates the solution process but can also effectively guide the solver toward better-quality solutions on more challenging instances.

In addition to analyzing the direction of solution quality changes, we also report the average optimality gaps relative to the lower bound obtained by the baseline configuration without preprocessing. Table 5 summarizes the corresponding results for instances that are not solved to optimality within the time limit.

The results show a substantial reduction in average optimality gaps compared to the baseline, particularly for larger instances.

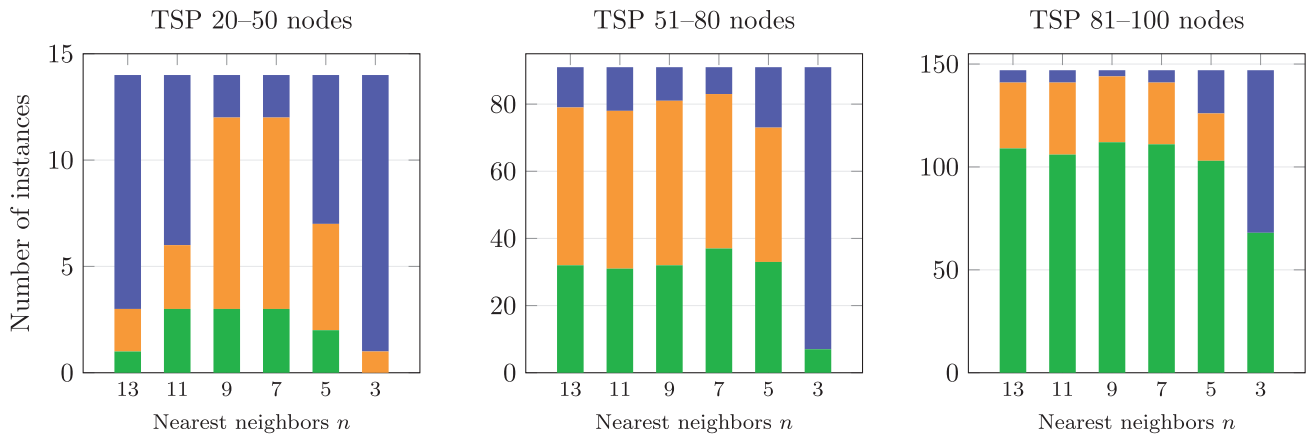


FIGURE 5 | Impact of nearest-neighbor preprocessing on solution quality at the time limit (TSP instances not solved to optimality). Green bars indicate improved solutions, orange bars equal quality, and blue bars worse solutions.

TABLE 5 | Average optimality gaps (in %) at the time limit, measured relative to the lower bound obtained without preprocessing (TSP instances not solved to optimality).

Nodes	Baseline	θ							n					
		0.005	0.01	0.03	0.05	0.1	0.2	0.3	13	11	9	7	5	3
20–50	1.98	1.94	1.94	1.93	2.00	2.14	2.58	2.82	1.94	1.95	1.94	1.99	2.21	3.10
51–80	1.86	1.62	1.60	1.58	1.59	1.63	1.71	1.90	1.70	1.73	1.71	1.62	1.66	3.09
81–100	3.78	2.14	2.09	2.09	2.06	2.07	2.09	2.21	2.20	2.24	2.18	2.17	2.17	3.47

For instances with 81–100 nodes, all preprocessing configurations reduce the gap relative to the full-graph formulation, with reductions of more than one percentage point for moderate values of θ and intermediate neighborhood sizes n . These findings confirm that preprocessing not only accelerates the solution process but also leads to significantly higher-quality solutions on challenging instances where optimality cannot be reached within the time limit.

Overall, the results in this section demonstrate that, with appropriate parameter choices, the proposed preprocessing methodology yields substantial computational benefits for the TSP. In particular, conservative configurations achieve significant runtime reductions for optimally solved instances without compromising optimality in almost all cases, while for instances that are not solved to optimality within the time limit, preprocessing consistently leads to improved solution quality. Another interesting observation is that the fraction of pruned edges increases with instance size. For example, using a threshold of $\theta = 0.1$, approximately 94.2% of all edges are removed for instances with 100 nodes, whereas only 72.1% are removed for instances with 20 nodes. This suggests that the proposed pruning strategy becomes increasingly effective at reducing the search space as the problem size grows.

5.4.2 | CVRP

We perform the same set of experiments for the CVRP as for the TSP. Due to the higher complexity of the CVRP, only a subset of instances can be solved to optimality within the time

limit; in particular, only 38 instances in the 20–50 node class are baseline-optimal, and the optimality analysis is therefore restricted to this group.

Table 6 reports, analogously to the TSP case, the number of baseline-optimal CVRP instances that remain optimal under preprocessing. The results show that for conservative preprocessing configurations, optimality is largely preserved, whereas more aggressive settings may lead to losses of optimality. Overall, the observed patterns are consistent with those obtained for the TSP. Even in cases where optimality is not preserved, the resulting deviations from the baseline remain small, as documented in C.

Table 7 reports the average solver runtimes for CVRP instances that are solved to optimality. Similar to the TSP case, substantial runtime reductions are achieved across all preprocessing configurations. In particular, threshold-based preprocessing yields pronounced speed-ups. Nearest-neighbor-based preprocessing also leads to significant runtime reductions for small values of n , while larger neighborhood sizes approach the baseline runtime.

We next consider CVRP instances that are not solved to optimality within the time limit. For some instances with 81–100 nodes, however, the solver without preprocessing failed to find any feasible solution within the time limit, and these are therefore excluded from our analysis. Figures 6 and 7 report, analogously to the TSP case, how preprocessing affects solution quality relative to the baseline. The results show a clear and consistent improvement in solution quality for the CVRP. Across all instance sizes, the majority of non-optimally solved instances

TABLE 6 | Number of baseline-optimal CVRP instances that remain optimal under preprocessing.

Nodes	Instances	θ							n					
		0.005	0.01	0.03	0.05	0.1	0.2	0.3	13	11	9	7	5	3
20–50	38	38	38	38	37	37	33	29	38	38	38	38	34	16

TABLE 7 | Average solver runtimes (in seconds) for optimally solved CVRP instances under different preprocessing configurations.

Nodes	Baseline	θ							n					
		0.005	0.01	0.03	0.05	0.1	0.2	0.3	13	11	9	7	5	3
20–50	68.0	49.5	42.2	22.8	14.7	6.2	2.8	1.1	66.5	61.2	59.7	47.7	25.2	1.6

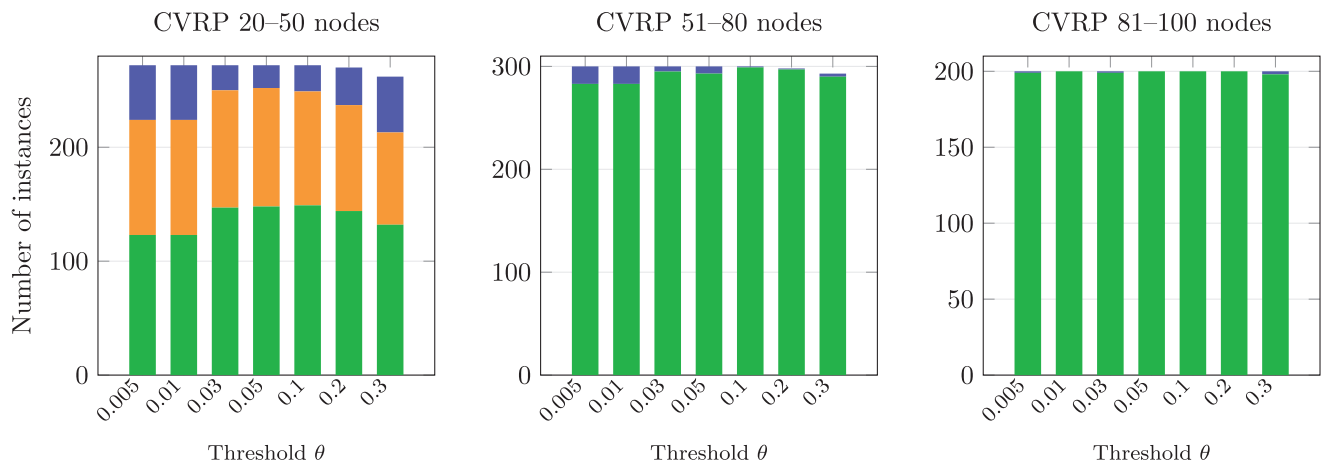


FIGURE 6 | Impact of threshold-based preprocessing on solution quality at the time limit (CVRP instances not solved to optimality). Green bars indicate better solutions, orange bars equal quality, and blue bars worse solutions.

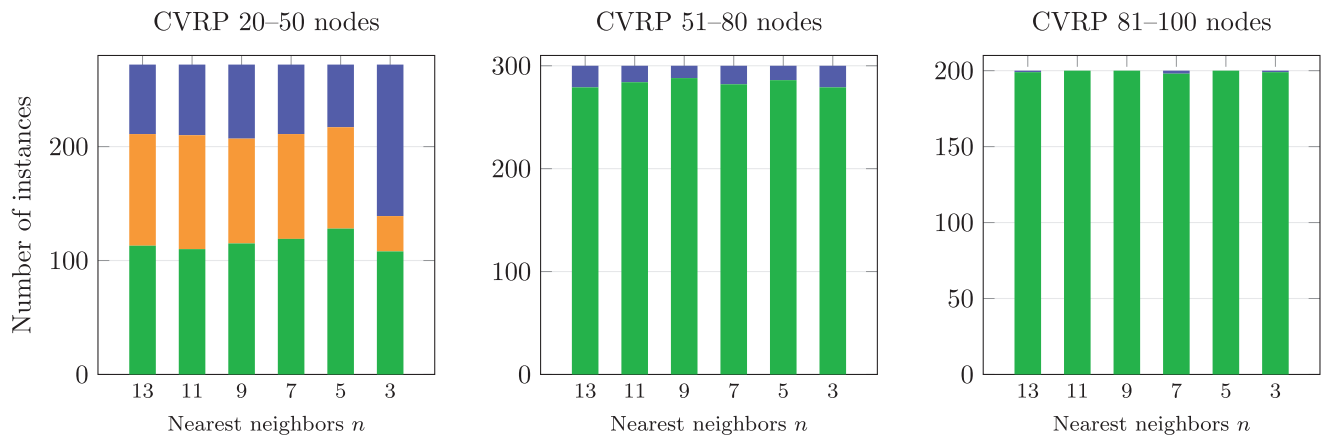


FIGURE 7 | Impact of nearest-neighbor preprocessing on solution quality at the time limit (CVRP instances not solved to optimality). Green bars indicate better solutions, orange bars equal quality, and blue bars worse solutions.

benefit from preprocessing, with a large share achieving better incumbent solutions than the baseline. This effect becomes particularly pronounced for medium and large instances, where nearly all instances exhibit improved solution quality for both threshold-based and nearest-neighbor-based preprocessing. Even relatively aggressive preprocessing configurations continue to yield improvements in most cases.

Table 8 further confirms the positive impact of preprocessing on solution quality for the CVRP. Across all instance sizes,

average optimality gaps are substantially reduced compared to the baseline formulation. The effect becomes increasingly pronounced for larger instances, where threshold-based preprocessing reduces average gaps by more than 7 percentage points for instances with 81–100 nodes. Nearest-neighbor-based preprocessing yields similarly consistent improvements, although slightly less pronounced.

In summary, the results for the CVRP confirm that the proposed preprocessing methodology yields substantial

TABLE 8 | Average optimality gaps (in %) at the time limit, measured relative to the lower bound obtained without preprocessing (CVRP instances not solved to optimality).

Nodes	Baseline	θ							n					
		0.005	0.01	0.03	0.05	0.1	0.2	0.3	13	11	9	7	5	3
20–50	13.79	13.22	13.17	12.96	12.91	12.86	12.89	13.09	13.28	13.29	13.28	13.26	13.10	13.61
51–80	29.87	25.62	25.54	25.24	25.06	24.64	24.38	24.39	25.73	25.81	25.69	25.57	25.61	25.91
81–100	38.44	31.95	31.71	31.56	31.39	30.87	30.50	30.43	32.01	31.99	32.08	31.86	31.71	32.19

computational benefits also for this more challenging problem class. With appropriate parameter choices, significant runtime reductions can be achieved for optimally solved instances while largely preserving optimality. Moreover, for instances that are not solved to optimality within the time limit, preprocessing consistently improves solution quality and significantly reduces optimality gaps. Interestingly, the improvements obtained for the CVRP are even more pronounced than for the TSP. This suggests that richer routing problems may provide additional structural information that can be exploited through problem-specific features to further improve the predictive quality of the confidence scores.

5.4.3 | Benchmark Against Confidence Scores With KNN

To better assess the contribution of the ML component, we apply the proposed framework using the confidence scores produced by the KNN classifier, which exhibited the weakest predictive performance among the evaluated ML models.

The complete benchmark results are reported in D. The results reveal the following observations:

- For threshold-based pruning, the KNN-based scores achieve a level of reliability comparable to the proposed XGB-based approach for conservative thresholds. However, a more detailed analysis reveals that substantially fewer edges are removed than with XGB confidence scores, resulting in noticeably smaller runtime reductions despite a similar preservation of optimal solutions (Tables D1 and D4).
- The superiority of XGB becomes even more pronounced for the top- n strategy. While KNN-based pruning still reduces runtimes compared to solving the original problem without preprocessing, the reductions obtained with XGB confidence scores are substantially larger and preserve a considerably larger number of baseline-optimal instances (Tables D2 and D5).
- For non-optimally solved instances, KNN-based pruning consistently improves the average optimality gaps compared to the baseline. However, the resulting gaps are generally slightly larger than those obtained with XGB confidence scores (Tables D3 and D6).

Overall, these findings indicate that the proposed framework is robust with respect to the underlying ML model. Nevertheless, the superior results obtained with XGB demonstrate that more accurate confidence scores enable more

aggressive yet reliable edge pruning, resulting in larger runtime reductions and improved solution quality.

5.4.4 | Benchmark Against Distance-Based Pruning

As a second benchmark, we considered the same top- n pruning strategy as before, but replaced the ML confidence scores with the Euclidean distances between nodes. Consequently, for each node, only the n closest neighbors were retained.

This benchmark was evaluated only for the TSP. For the CVRP, preliminary experiments showed that relying solely on distances leads to poor results, since route construction is strongly influenced by customer demands and vehicle capacities. The complete benchmark results are reported in E.

The results reveal the following observations:

- Distance-based pruning preserves considerably fewer baseline-optimal solutions than the proposed ML-based approach, particularly for stronger pruning (Table E1).
- Although distance-based pruning reduces solver runtimes compared to the baseline, the reductions are substantially smaller than those achieved with the proposed ML-based pruning. Moreover, stronger pruning often results in higher runtimes while simultaneously preserving fewer optimal solutions (Table E2).
- For non-optimally solved instances, distance-based pruning improves the average optimality gaps compared to the baseline for moderate pruning levels, that is, larger values of n . However, the resulting gaps are consistently slightly larger than those obtained with the proposed ML-based pruning (Table E3).

Furthermore, for very aggressive pruning ($n = 3$ and, to a lesser extent, $n = 5$), distance-based candidate lists frequently remove edges required to construct a Hamiltonian cycle, preventing the solver from finding a feasible solution. This behavior was not observed for the proposed ML-based pruning strategy.

5.4.5 | Generalization to Distribution-Shifted Instances

To evaluate the robustness of the proposed edge classification approach under structural distribution shifts, we repeated the threshold-based pruning experiments for the CVRP on a new

TABLE 9 | Number of baseline-optimal CVRP instances that remain optimal and average solver runtimes (in seconds) under threshold-based pruning on distribution-shifted instances.

Nodes	Metric	Baseline	0.005	0.01	0.03
20–50	Optimal solutions	31	31	31	24
	Runtime (s)	56.7	21.2	11.2	1.9

TABLE 10 | Average optimality gaps (in %) at the time limit for baseline nonoptimal CVRP instances under threshold-based pruning on distribution-shifted instances. Gaps are measured relative to the lower bound obtained without pruning.

Nodes	Baseline	0.005	0.01	0.03
20–50	20.82	19.69	19.66	19.82
51–80	39.95	35.49	35.39	35.18
81–100	47.80	42.09	41.91	41.69

set of instances generated with modified generator settings while keeping the training data unchanged. Consequently, the classifier was trained on the original instance distribution but evaluated on the modified instance distribution, allowing us to assess its generalization performance under a structural distribution shift between the training and test data. Compared with the training instances, the test instances differ in the demand distribution (small values with high instead of low coefficient of variation) and the average route size (medium instead of long). A detailed description of the instance generator and its configuration parameters is provided by Queiroga et al. [54]. For each instance size from 20 to 100 customers, 20 additional instances were generated.

Preliminary experiments showed that larger pruning thresholds frequently removed too many candidate edges on these structurally different instances, occasionally preventing the solver from finding a feasible solution. Therefore, we restrict the following analysis to the practically relevant thresholds 0.005, 0.01, and 0.03.

Tables 9 and 10 summarize the results. For baseline-optimal instances, thresholds 0.005 and 0.01 preserved all optimal solutions while reducing the average computation time from 56.7 s to 21.2 s and 11.2 s, respectively. Although a threshold of 0.03 further reduced the average runtime to 1.9 s, it preserved only 24 of the 31 baseline-optimal solutions.

For instances that were not solved to optimality within the time limit, moderate pruning again resulted in smaller average optimality gaps than the baseline across all instance sizes. For example, the average gap decreased from 39.95% to 35.39% for instances with 51–80 customers and from 47.80% to 41.91% for instances with 81–100 customers when using a threshold of 0.01. Overall, these results indicate that moderate ML-based pruning remains effective even under structural differences between the training and test distributions, whereas more aggressive pruning becomes less robust. However, in real-world applications, classifiers would typically be retrained if substantial structural changes occur.

5.5 | Results Using a Metaheuristic

We next assess the impact of the ML-based pruning when embedded into a state-of-the-art metaheuristic on the *medium* ($|\mathcal{N}| = 200\text{--}500$) and *large* ($|\mathcal{N}| = 900\text{--}1000$) instance set (see Section 5.1). To this end, we restrict the neighborhood explored by the heuristic using the predicted confidence scores, either by including only candidates with scores above a threshold θ (Section 4.4.1) or by retaining only the n most promising neighbors per node (Section 4.4.2).

Performance is measured relative to the baseline heuristic without learned pruning, in terms of (i) the average objective gap (negative values indicate improvements) and (ii) the average runtime difference (negative values indicate speed-ups). The average runtime difference is measured as the time needed to achieve the same (or a better) solution as the baseline approach (or vice versa if the baseline approach leads to a better solution). For example, suppose the baseline approach attains its best objective value of 1000 after 50 s. We then determine the earliest time at which the pruning-based approach achieves a solution with objective value 1000 or better. If this occurs after 40 s, the resulting runtime difference is -10 s.

All settings, problem classes, and instance sizes were solved under a runtime limit of 60 s per instance. Consequently, the reported runtime savings must be interpreted relative to this limit. In many cases, particularly for simpler instances, the best solution was found substantially earlier than the time limit. This implies that the relative runtime savings are effectively even larger than what the absolute differences alone suggest.

5.5.1 | TSP

Figures 8 and 9 summarize the results for the TSP. For medium-sized instances ($|\mathcal{N}| = 200\text{--}500$), both pruning mechanisms yield consistent runtime reductions while maintaining solution quality close to the baseline. Neighbor-based pruning with moderate values of n (e.g., $n \geq 5$) typically preserves the objective value while achieving speed-ups of approximately 3.8 s. Very small neighborhoods ($n = 3$) can lead to a slight deterioration on smaller instances, but remain competitive overall.

For large instances ($|\mathcal{N}| = 900\text{--}1000$), the effect becomes stronger. Here, pruning not only reduces runtime but frequently improves solution quality. With neighbor-based reduction, average gaps improve by up to roughly -0.75% , accompanied by runtime reductions of around 20 s for moderate values of n . Threshold-based pruning exhibits an even clearer and more systematic trend. Increasing θ leads to progressively stronger improvements in both runtime and solution quality. For large instances, aggressive thresholding (e.g., $\theta \geq 0.5$) achieves average improvements exceeding -0.6% while reducing runtime by up to approximately 21 s. Across most instance sizes, threshold-based pruning appears to perform more robustly and consistently than fixed-size neighborhood pruning. Overall, the TSP results indicate that moderate-to-aggressive pruning is highly effective within the metaheuristic. Given the 60-second runtime limit, savings of up to 20 s correspond to substantial relative reductions in computational effort.

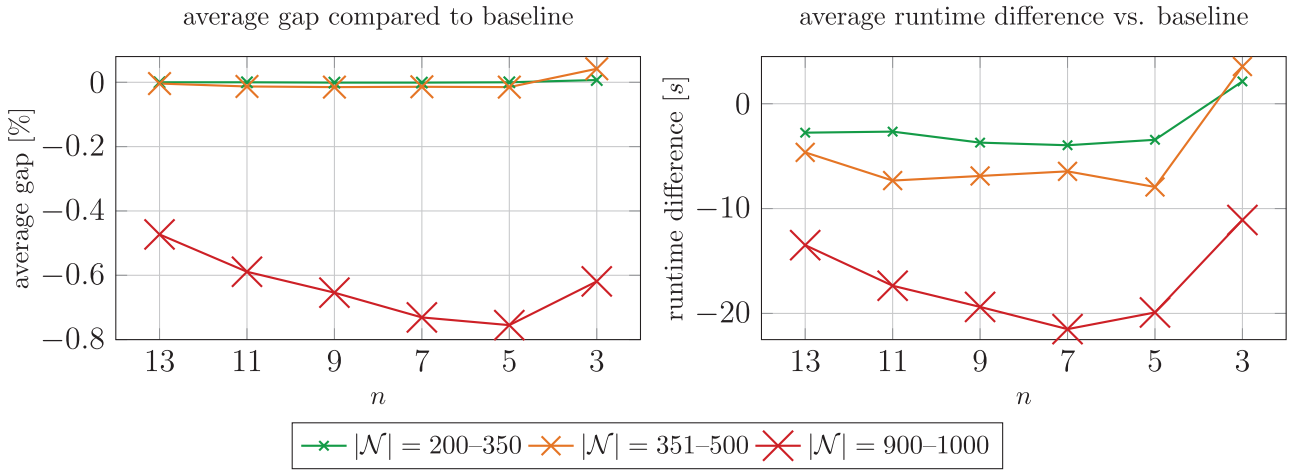


FIGURE 8 | Comparison to the baseline across different instance sizes for TSP and varying neighborhood size n : (left) average gap in percent, (right) average runtime difference in seconds.

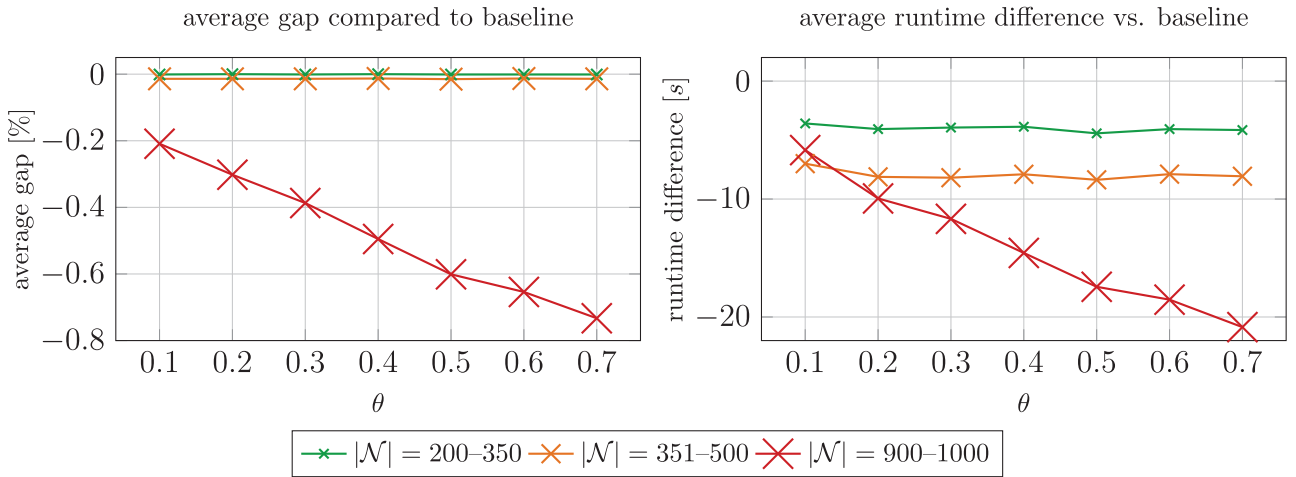


FIGURE 9 | Comparison to the baseline across different instance sizes for TSP and varying thresholds θ : (left) average gap in percent, (right) average runtime difference in seconds.

Note that the baseline approach already restricts the neighborhood based solely on distance. Without any neighborhood restriction, the algorithm's performance would deteriorate substantially. Compared to the baseline, the average runtime difference would increase by 26 s ($|\mathcal{N}| = 200-350$), 45 s ($|\mathcal{N}| = 351-500$), and 34 s ($|\mathcal{N}| = 900-1000$), while yielding worse average gaps of 0.042%, 0.599%, and 2.55%, respectively.

5.5.2 | CVRP

The results for the CVRP (Figures 10 and 11) are similar to those of the TSP. For instances with $|\mathcal{N}| = 200-350$, both pruning strategies yield stable runtime reductions of roughly 3 s with negligible impact on solution quality. As in the TSP, overly restrictive neighborhoods ($n = 3$) can be detrimental, but moderate settings remain robust. For instances with $|\mathcal{N}| = 351-500$, pruning becomes more beneficial. Threshold-based pruning produces small but consistent quality improvements (down to approximately -0.03%) together with runtime savings of up to about 14 s. Neighbor-based pruning achieves comparable speed-ups.

The strongest effects are observed for large instances ($|\mathcal{N}| = 900-1000$). Here, both mechanisms substantially reduce runtime and often improve solution quality. Neighbor-based pruning improves the average gap by up to about -0.50% while reducing runtime by roughly 10–13 s. Threshold-based pruning again shows a clear pattern: while very small thresholds may be ineffective or even slightly detrimental, more aggressive settings ($\theta \geq 0.4$) lead to improvements, reaching average gains of about -0.86% together with runtime reductions exceeding 20 s. Compared to the TSP, the qualitative behavior is similar but slightly less uniform. The CVRP is structurally more complex due to capacity constraints and route interactions, which likely explains the higher sensitivity to very aggressive pruning on smaller instances. Nevertheless, for large-scale instances, the overall picture mirrors the TSP results: threshold-based pruning in particular performs very well and benefits from more aggressive filtering.

In summary, within a metaheuristic framework, learned pruning yields substantial runtime reductions of up to 25 s under a 60 s limit while maintaining or even improving solution quality.

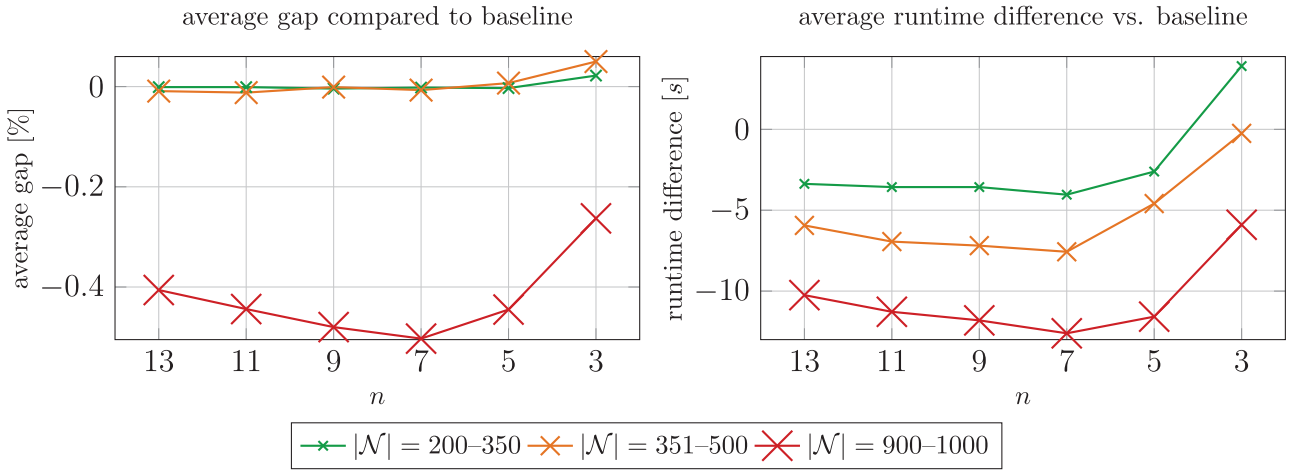


FIGURE 10 | Comparison to the baseline across different instance sizes for CVRP and varying neighborhood size n : (left) average gap in percent, (right) average runtime difference in seconds.

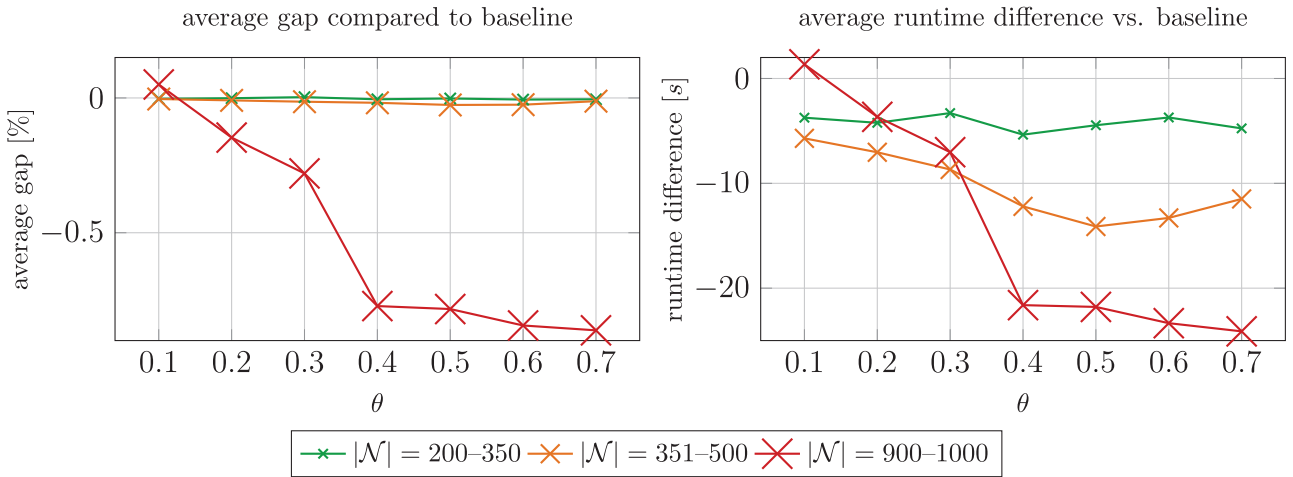


FIGURE 11 | Comparison to the baseline across different instance sizes for CVRP and varying thresholds θ : (left) average gap in percent, (right) average runtime difference in seconds.

In contrast to the setting with exact solvers (Section 5.4), more aggressive pruning is feasible in the metaheuristic context. While removing a critical edge can eliminate the true optimum when an exact solver is applied to the pruned graph, the metaheuristic does not strictly exclude edges from the global search space. Even if certain high-quality candidates are not included in the restricted neighborhood, alternative moves may still recover competitive or superior solutions. Hence, the risk associated with aggressive pruning is lower in this setting.

6 | Conclusion

In this study, we developed a machine-learning-based framework to generate confidence scores that predict whether an edge is likely to be part of a high-quality routing solution. These confidence scores are used within an ML-based preprocessing step to prune edges that are unlikely to belong to high-quality solutions before the optimization process begins. The resulting reduced problem instances were then solved using a mathematical optimization solver and a metaheuristic, leading to significant

reductions in computational time and improvements in solution quality, especially for larger problem instances. This work demonstrates that problem-specific structural information, when encoded through suitable features, enables ML models to accurately discriminate between edges that are and are not part of a high-quality solution. Our numerical study shows that the resulting confidence scores can be reliably integrated into both exact and state-of-the-art metaheuristic approaches. This integration yields substantial performance improvements by reducing the search space and guiding the solution process.

The implications of these results go beyond routing problems. First, ML-based preprocessing can meaningfully shrink the search space of large combinatorial optimization problems, thereby accelerating the solution process of both solvers and heuristic methods. On the tested instances, this reduction empirically preserved optimality or near-optimality in most cases, although no formal guarantee is provided. Second, the resulting confidence scores provide valuable guidance for steering heuristics, improving their decisions during the search. Third, our feature analysis offers insights into the structural properties that

determine whether an edge is likely to belong to a high-quality solution.

These findings open several promising avenues for future work. Additional learning opportunities arise within routing problems, such as developing problem-specific operators for metaheuristics or integrating ML-based edge pruning into more advanced exact algorithms. Beyond routing, many large-scale optimization problems exhibit similarly vast search spaces in which substantial portions can potentially be eliminated through learning-based approaches.

An important limitation of the current study is that the experimental analysis is restricted to the TSP and CVRP. While the proposed framework is flexible and can incorporate additional problem-specific features for richer routing settings, its applicability to variants such as VRPTW, pickup-and-delivery, or stochastic routing problems remains an interesting direction for future research. Furthermore, evaluating the robustness of the proposed framework across completely different instance distributions, such as clustered customer locations, represents another promising direction for future research.

We hope that our results encourage further research at the intersection of ML and combinatorial optimization, both in vehicle routing and in other challenging $\mathcal{NP}$ -hard problem settings.

Author Contributions

Johannes Gückel: conceptualization; methodology; software; validation; visualization; writing – original draft; writing – review and editing; data curation. **Stefan Voigt:** conceptualization; methodology; software; validation; visualization; writing – original draft; writing – review and editing. **Pirmin Fontaine:** conceptualization; methodology; validation; supervision; funding acquisition; project administration; writing – review and editing; resources.

Acknowledgments

We thank the anonymous reviewers for their valuable recommendations, which have significantly improved our paper. Open Access funding enabled and organized by Projekt DEAL.

Funding

The authors have nothing to report.

Conflicts of Interest

The authors declare no conflicts of interest.

Data Availability Statement

The data that support the findings of this study are available from the corresponding author upon reasonable request.

References

1. P. Toth and D. Vigo, *Vehicle Routing: Problems, Methods, and Applications*, 2nd ed. (SIAM, 2014).
2. R. Bai, X. Chen, Z. L. Chen, et al., “Analytics and Machine Learning in Vehicle Routing Research,” *International Journal of Production Research* 61 (2021): 4–30, <https://doi.org/10.1080/00207543.2021.2013566>.

3. R. Shahbazian, L. D. P. Pugliese, F. Guerriero, and G. Macrina, “Integrating Machine Learning Into Vehicle Routing Problem: Methods and Applications,” *IEEE Access* 12 (2024): 93087–93115, <https://doi.org/10.1109/access.2024.3422479>.
4. C. Spieckermann, S. Minner, and M. Schiffer, “Reduce-Then-Optimize for the Fixed-Charge Transportation Problem,” *Transportation Science* 59 (2025): 540–564, <https://doi.org/10.1287/trsc.2023.0407>.
5. J. Lauri, S. Dutta, M. Grassia, and D. Ajwani, “Learning Fine-Grained Search Space Pruning and Heuristics for Combinatorial Optimization,” *Journal of Heuristics* 29 (2023): 313–347, <https://doi.org/10.1007/s10732-023-09512-z>.
6. Y. Sun, A. Ernst, X. Li, and J. Weiner, “Generalization of Machine Learning for Problem Reduction: A Case Study on Travelling Salesman Problems,” *OR Spectrum* 43 (2020): 607–633, <https://doi.org/10.1007/s00291-020-00604-x>.
7. T. Vidal, G. Laporte, and P. Matl, “A Concise Guide to Existing and Emerging Vehicle Routing Problem Variants,” *European Journal of Operational Research* 286 (2020): 401–416, <https://doi.org/10.1016/j.ejor.2019.10.010>.
8. K. Braekers, K. Ramaekers, and I. Van Nieuwenhuysse, “The Vehicle Routing Problem: State of the Art Classification and Review,” *Computers & Industrial Engineering* 99 (2016): 300–313, <https://doi.org/10.1016/j.cie.2015.12.007>.
9. A. W. J. Kolen, A. H. G. Rinnooy Kan, and H. W. J. M. Trienekens, “Vehicle Routing With Time Windows,” *Operations Research* 35 (1987): 266–273, <https://doi.org/10.1287/opre.35.2.266>.
10. J. Renaud, G. Laporte, and F. F. Boctor, “A Tabu Search Heuristic for the Multi-Depot Vehicle Routing Problem,” *Computers & Operations Research* 23 (1996): 229–235, [https://doi.org/10.1016/0305-0548\(95\)00026-p](https://doi.org/10.1016/0305-0548(95)00026-p).
11. M. Gendreau, G. Laporte, and R. S. Guin, “Stochastic Vehicle Routing,” *European Journal of Operational Research* 88 (1996): 3–12, [https://doi.org/10.1016/0377-2217\(95\)00050-x](https://doi.org/10.1016/0377-2217(95)00050-x).
12. G. Perboli, R. Tadei, and D. Vigo, “The Two-Echelon Capacitated Vehicle Routing Problem: Models and Math-Based Heuristics,” *Transportation Science* 45 (2011): 364–380, <https://doi.org/10.1287/trsc.1110.0368>.
13. H. Min, “The Multiple Vehicle Routing Problem With Simultaneous Delivery and Pick-Up Points,” *Transportation Research Part A, General* 23 (1989): 377–386, [https://doi.org/10.1016/0191-2607\(89\)90085-x](https://doi.org/10.1016/0191-2607(89)90085-x).
14. X. Li, R. Bai, P. O. Siebers, and C. Wagner, “Travel Time Prediction in Transport and Logistics: Towards More Efficient Vehicle Gps Data Management Using Tree Ensemble Methods,” *VINE Journal of Information and Knowledge Management Systems* 49 (2019): 277–306, <https://doi.org/10.1108/vjiks-11-2018-0102>.
15. R. Basso, B. Kulcs r, and I. Sanchez-Diaz, “Electric Vehicle Routing Problem With Machine Learning for Energy Prediction,” *Transportation Research Part B: Methodological* 145 (2021): 24–55, <https://doi.org/10.1016/j.trb.2020.12.007>.
16. G. M. Hurovich, N. A. Agatz, and L. Veelenturf, “Real-Time Routing Cost Estimations for Time Slot Management,” (2025), <https://doi.org/10.2139/ssrn.5497838>.
17. A. Nalbant and B. Yildiz, “A Machine Learning Approach for Marginal Fulfillment Cost Estimation in Last Mile Delivery,” *Transportation Research Part C, Emerging Technologies* 178 (2025): 105163, <https://doi.org/10.1016/j.trc.2025.105163>.
18. E. Larsen, S. Lachapelle, Y. Bengio, E. Frejinger, S. Lacoste-Julien, and A. Lodi, “Predicting Tactical Solutions to Operational Planning Problems Under Imperfect Information,” *INFORMS Journal on Computing* 34 (2022): 227–242, <https://doi.org/10.1287/ijoc.2021.1091>.
19. F. Akkerman and M. Mes, “Distance Approximation to Support Customer Selection in Vehicle Routing Problems,” *Annals of*

- Operations Research* 350 (2022): 269–297, <https://doi.org/10.1007/s10479-022-04674-8>.
20. A. Kononenko, L. M. Hvattum, and M. K. Msakni, “Using Machine Learning to Identify Hidden Constraints in Vehicle Routing Problems,” *Computers & Operations Research* 179 (2025): 107029, <https://doi.org/10.1016/j.cor.2025.107029>.
 21. J. Gückel and P. Fontaine, “Fast Shapley Value Approximation Through Machine Learning With Application in Routing Problems,” *Networks* 86 (2025): 402–427, <https://doi.org/10.1002/net.70003>.
 22. W. Qin, Z. Zhuang, Z. Huang, and H. Huang, “A Novel Reinforcement Learning-Based Hyper-Heuristic for Heterogeneous Vehicle Routing Problem,” *Computers & Industrial Engineering* 156 (2021): 107252, <https://doi.org/10.1016/j.cie.2021.107252>.
 23. M. E. Bruni, E. Fadda, S. Fedorov, and G. Perboli, “A Machine Learning Optimization Approach for Last-Mile Delivery and Third-Party Logistics,” *Computers & Operations Research* 157 (2023): 106262, <https://doi.org/10.1016/j.cor.2023.106262>.
 24. A. Baldo, M. Boffa, L. Cascioli, E. Fadda, C. Lanza, and A. Ravera, “The Polynomial Robust Knapsack Problem,” *European Journal of Operational Research* 305 (2023): 1424–1434, <https://doi.org/10.1016/j.ejor.2022.06.029>.
 25. U. Ermağan, B. Yildiz, and F. S. Salman, “A Learning Based Algorithm for Drone Routing,” *Computers & Operations Research* 137 (2022): 105524, <https://doi.org/10.1016/j.cor.2021.105524>.
 26. F. Arnold and K. Sörensen, “What Makes a VRP Solution Good? The Generation of Problem-Specific Knowledge for Heuristics,” *Computers & Operations Research* 106 (2019): 280–288, <https://doi.org/10.1016/j.cor.2018.02.007>.
 27. F. Arnold and K. Sörensen, “Knowledge-Guided Local Search for the Vehicle Routing Problem,” *Computers & Operations Research* 105 (2019): 32–46, <https://doi.org/10.1016/j.cor.2019.01.002>.
 28. M. R. D. H. Maia, A. Plastino, and U. D. S. Souza, “An Improved Hybrid Genetic Search With Data Mining for the CVRP,” *Networks* 83 (2024): 692–711, <https://doi.org/10.1002/net.22213>.
 29. B. Herdianto, R. Billot, F. Lucas, and M. Sevaux, “Feature-Guided Metaheuristic With Diversity Management for Solving the Capacitated Vehicle Routing Problem,” *European Journal of Operational Research* 333 (2025): 414–428, <https://doi.org/10.1016/j.ejor.2025.12.029>.
 30. M. Karimi-Mamaghan, M. Mohammadi, P. Meyer, A. M. Karimi-Mamaghan, and E. G. Talbi, “Machine Learning at the Service of Meta-Heuristics for Solving Combinatorial Optimization Problems: A State-Of-The-Art,” *European Journal of Operational Research* 296 (2022): 393–422, <https://doi.org/10.1016/j.ejor.2021.04.032>.
 31. R. Asín Achá, A. Espinoza, O. Goldschmidt, D. S. Hochbaum, and I. I. Huerta, “Selecting Fast Algorithms for the Capacitated Vehicle Routing Problem With Machine Learning Techniques,” *Networks* 84 (2024): 465–480, <https://doi.org/10.1002/net.22244>.
 32. S. Chamurraly and J. Rieck, “A Practical and Robust Approach for Solving the Multi-Compartment Vehicle Routing Problem Under Demand Uncertainty Using Machine Learning,” *Networks* 84 (2024): 300–325, <https://doi.org/10.1002/net.22231>.
 33. M. Morabit, G. Desaulniers, and A. Lodi, “Machine-Learning Based Column Selection for Column Generation,” *Transportation Science* 55 (2021): 815–831, <https://doi.org/10.1287/trsc.2021.1045>.
 34. K. Xu, L. Shen, and L. Liu, “Enhancing Column Generation by Reinforcement Learning-Based Hyper-Heuristic for Vehicle Routing and Scheduling Problems,” *Computers & Industrial Engineering* 206 (2025): 111138, <https://doi.org/10.1016/j.cie.2025.111138>.
 35. X. Zhang, L. Chen, M. Gendreau, and A. Langevin, “Learning-Based Branch-And-Price Algorithms for the Vehicle Routing Problem With Time Windows and Two-Dimensional Loading Constraints,” *INFORMS Journal on Computing* 34 (2022): 1419–1436, <https://doi.org/10.1287/ijoc.2021.1110>.
 36. I. Mitrai and P. Daoutidis, “Computationally Efficient Solution of Mixed Integer Model Predictive Control Problems via Machine Learning Aided Benders Decomposition,” *Journal of Process Control* 137 (2024): 103207, <https://doi.org/10.1016/j.jprocont.2024.103207>.
 37. A. M. Alvarez, Q. Louveaux, and L. Wehenkel, “A Machine Learning-Based Approximation of Strong Branching,” *INFORMS Journal on Computing* 29 (2017): 185–195, <https://doi.org/10.1287/ijoc.2016.0723>.
 38. L. Scavuzzo, K. Aardal, A. Lodi, and N. Yorke-Smith, “Machine Learning Augmented Branch and Bound for Mixed Integer Linear Programming,” *Mathematical Programming* 217 (2024): 123–166, <https://doi.org/10.1007/s10107-024-02130-y>.
 39. N. Rashedi, N. Sankey, V. Vaze, and K. Wei, “A Machine Learning Approach for Solution Space Reduction in Aircraft Disruption Recovery,” *European Journal of Operational Research* 323 (2025): 297–308, <https://doi.org/10.1016/j.ejor.2024.11.025>.
 40. Z. Xiao, J. Zhi, and B. B. Keskin, “Towards a Machine Learning-Aided Metaheuristic Framework for a Production/Distribution System Design Problem,” *Computers & Operations Research* 146 (2022): 105897, <https://doi.org/10.1016/j.cor.2022.105897>.
 41. Y. Sun, X. Li, and A. Ernst, “Using Statistical Measures and Machine Learning for Graph Reduction to Solve Maximum Weight Clique Problems,” *IEEE Transactions on Pattern Analysis and Machine Intelligence* 43 (2021): 1746–1760, <https://doi.org/10.1109/tpami.2019.2954827>.
 42. Z. Li, Q. Chen, and V. Koltun, “Combinatorial Optimization With Graph Convolutional Networks and Guided Tree Search,” in *Proceedings of the 32nd International Conference on Neural Information Processing Systems* (Curran Associates Inc, 2018), 537–546.
 43. J. Y. Ding, C. Zhang, L. Shen, et al., “Accelerating Primal Solution Findings for Mixed Integer Programs Based on Solution Prediction,” (2019), <https://doi.org/10.48550/ARXIV.1906.09575>.
 44. L. Huang, F. Xiao, J. Zhou, Z. Duan, H. Zhang, and Z. Liang, “A Machine Learning Based Column-And-Row Generation Approach for Integrated Air Cargo Recovery Problem,” *Transportation Research Part B: Methodological* 178 (2023): 102846, <https://doi.org/10.1016/j.trb.2023.102846>.
 45. P. Koutecká, P. Šoucha, J. Høula, and B. Maenhout, “A Machine Learning Approach to Rank Pricing Problems in Branch-And-Price,” *European Journal of Operational Research* 320 (2025): 328–342, <https://doi.org/10.1016/j.ejor.2024.07.029>.
 46. W. Cook, K. Helsgaun, S. Hougardy, and R. T. Schroeder, “Local Elimination in the Traveling Salesman Problem,” *ArXiv:2307.07054*, (2023).
 47. T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning* (Springer, 2009), <https://doi.org/10.1007/978-0-387-84858-7>.
 48. Y. LeCun, Y. Bengio, and G. Hinton, “Deep Learning,” *Nature* 521 (2015): 436–444, <https://doi.org/10.1038/nature14539>.
 49. L. Breiman, “Random Forests,” *Machine Learning* 45 (2001): 5–32, <https://doi.org/10.1023/a:1010933404324>.
 50. T. Chen and C. Guestrin, “XGBoost: A Scalable Tree Boosting System,” in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (ACM, 2016), 785–794, <https://doi.org/10.1145/2939672.2939785>.
 51. P. Toth and D. Vigo, “The Granular Tabu Search and Its Application to the Vehicle-Routing Problem,” *INFORMS Journal on Computing* 15 (2003): 333–346, <https://doi.org/10.1287/ijoc.15.4.333.24890>.
 52. T. Vidal, T. G. Crainic, M. Gendreau, N. Lahrichi, and W. Rei, “A Hybrid Genetic Algorithm for Multidepot and Periodic Vehicle Routing Problems,” *Operations Research* 60 (2012): 611–624, <https://doi.org/10.1287/opre.1120.1048>.

53. T. Vidal, “Hybrid Genetic Search for the CVRP: Open-Source Implementation and SWAP* Neighborhood,” *Computers & Operations Research* 140 (2022): 105643, <https://doi.org/10.1016/j.cor.2021.105643>.
54. E. Queiroga, R. Sadykov, E. Uchoa, and T. Vidal, “10,000 Optimal CVRP Solutions for Testing Machine Learning Based Heuristics,” in *AAAI-22 Workshop on Machine Learning for Operations Research (ML4OR)* (2022), <https://openreview.net/forum?id=yHiMXKN6nTl>.
55. N. A. Wouda, L. Lan, and W. Kool, “PyVRP: A High-Performance VRP Solver Package,” *INFORMS Journal on Computing* 36 (2024): 943–955, <https://doi.org/10.1287/ijoc.2023.0055>.
56. T. O’Malley, E. Bursztein, J. Long, et al., “KerasTuner,” (2019), <https://github.com/keras-team/keras-tuner>.
57. F. Pedregosa, G. Varoquaux, A. Gramfort, et al., “Scikit-Learn: Machine Learning in Python,” *Journal of Machine Learning Research* 12 (2011): 2825–2830.
58. S. M. Lundberg and S. I. Lee, “A Unified Approach to Interpreting Model Predictions,” in *Proceedings of the 31st International Conference on Neural Information Processing Systems* (Curran Associates Inc, 2017), 4768–4777.

Appendix A: Mathematical Model Formulations

TSP Formulation

We consider the TSP on a complete graph $G = (\mathcal{N}, \mathcal{E})$, where $\mathcal{N}$ is the set of nodes and $\mathcal{E}$ the set of edges. Each edge $(i, j) \in \mathcal{E}$ is associated with a nonnegative travel cost c_{ij} . The binary decision variable x_{ij} equals 1 if the tour traverses edge (i, j) and 0 otherwise. To eliminate subtours, we use the Miller–Tucker–Zemlin (MTZ) formulation with continuous variables u_i for $i \in \mathcal{N} \setminus \{0\}$. The TSP can then be written as:

$$\begin{aligned}
 \min \quad & \sum_{(i,j) \in \mathcal{E}} c_{ij} x_{ij} \\
 \text{s.t.} \quad & \sum_{j \in \mathcal{N} \setminus \{i\}} x_{ij} = 1 & \forall i \in \mathcal{N} \\
 & \sum_{i \in \mathcal{N} \setminus \{j\}} x_{ij} = 1 & \forall j \in \mathcal{N} \\
 & u_i - u_j + (|\mathcal{N}| - 1) x_{ij} \leq |\mathcal{N}| - 2 & \forall i, j \in \mathcal{N} \setminus \{0\}, i \neq j \\
 & 1 \leq u_i \leq |\mathcal{N}| - 1 & \forall i \in \mathcal{N} \setminus \{0\} \\
 & x_{ij} \in \{0, 1\} & \forall (i, j) \in \mathcal{E}, i \neq j.
 \end{aligned}$$

CVRP Formulation

The CVRP is defined on the same complete graph $G = (\mathcal{N}, \mathcal{E})$, where each edge $(i, j) \in \mathcal{E}$ incurs a nonnegative cost c_{ij} . Each customer node $i \in \mathcal{N}$ is associated with a demand d_i , and vehicles have a fixed capacity Q . The binary decision variable x_{ij} indicates whether edge (i, j) is traversed, while the continuous variable u_i represents the cumulative load upon arrival at node i . Using this notation, the CVRP can be expressed as the following mixed-integer formulation:

$$\begin{aligned}
 \min \quad & \sum_{i \in \mathcal{N}} \sum_{j \in \mathcal{N} \setminus \{i\}} c_{ij} x_{ij} \\
 \text{s.t.} \quad & \sum_{j \in \mathcal{N} \setminus \{i\}} x_{ij} = 1 & \forall i \in \mathcal{N} \setminus \{0\} \\
 & \sum_{i \in \mathcal{N} \setminus \{j\}} x_{ij} = 1 & \forall j \in \mathcal{N} \setminus \{0\} \\
 & d_i \leq u_i \leq Q & \forall i \in \mathcal{N} \setminus \{0\} \\
 & u_i - u_j + Q x_{ij} \leq Q - d_j & \forall i, j \in \mathcal{N} \setminus \{0\}, i \neq j \\
 & x_{ij} \in \{0, 1\} & \forall (i, j) \in \mathcal{E}, i \neq j.
 \end{aligned}$$

Appendix B: Hyperparameter Configurations

See Tables B1–B4.

TABLE B1 | Neural network hyperparameter configuration for TSP and CVRP instances.

Hyperparameter	TSP (small)	TSP (medium)	TSP (large)	CVRP (small)	CVRP (medium)	CVRP (large)
Number of layers	1	2	2	1	2	2
Units per layer	[160]	[416, 288]	[352, 320]	[192]	[192, 64]	[254, 192]
Batch size	1024	1024	1024	1024	1024	1024
Learning rate	0.0032	0.0006	0.0029	0.0018	0.0025	0.0012
Activation function	relu	relu	relu	relu	relu	relu
Dropout	0.2	0.2	0.1	0.3	0.2	0.3

TABLE B2 | XGBoost hyperparameter configuration for TSP and CVRP instances.

Hyperparameter	TSP (small)	TSP (medium)	TSP (large)	CVRP (small)	CVRP (medium)	CVRP (large)
colsample_bytree	0.72	0.79	0.80	0.71	0.79	0.60
gamma	0.59	0.48	0.55	0.57	0.48	0.42
learning_rate	0.03	0.06	0.04	0.05	0.06	0.05
max_depth	7	18	16	10	6	8
min_child_weight	7	1	1	4	7	4
n_estimators	470	655	237	533	886	368
reg_lambda	6.69	7.76	3.04	3.26	3.00	3.25
subsample	0.62	0.66	0.66	0.71	0.67	0.76

TABLE B3 | Random Forest hyperparameter configuration for TSP and CVRP instances.

Hyperparameter	TSP (small)	TSP (medium)	TSP (large)	CVRP (small)	CVRP (medium)	CVRP (large)
n_Estimators	477	500	230	236	174	421
max_Features	sqrt	sqrt	log2	none	none	sqrt
max_Depth	20	20	20	16	17	16
min_Samples_Leaf	1	1	6	3	2	1
min_Samples_Split	11	11	12	9	8	4
Bootstrap	false	false	false	true	true	false

TABLE B4 | KNN hyperparameter configuration for TSP and CVRP instances.

Hyperparameter	TSP (small)	TSP (medium)	TSP (large)	CVRP (small)	CVRP (medium)	CVRP (large)
K (n_neighbors)	222	208	401	241	245	241
Distance metric (p)	2	2	1	1	1	1
Weights	uniform	distance	distance	distance	distance	distance

Appendix C: Relative Gap Analysis

This appendix reports the average relative gap (in percent) between the optimal baseline solution and the solution obtained under preprocessing. Only instances for which optimality is lost through the preprocessing step are included.

Table C1 presents the results for TSP, and Table C2 for CVRP instances. An empty entry (–) indicates that optimality was preserved for all baseline-optimal instances under the respective preprocessing configuration. The results indicate that even when optimality is lost due to preprocessing, the resulting gaps remain small.

Table C2 reports the corresponding analysis for CVRP instances. Although preprocessing occasionally leads to deviations beyond the tolerance threshold, the average deterioration remains moderate and below 2% across all reported configurations.

TABLE C1 | Average relative gap (in %) compared to the lower bound of the baseline-optimal TSP instances without preprocessing.

Nodes	Instances	θ							n					
		0.005	0.01	0.03	0.05	0.1	0.2	0.3	13	11	9	7	5	3
20–50	296	—	0.84	0.58	0.59	0.99	0.86	1.12	0.16	0.21	0.63	0.72	0.79	1.89
51–80	209	—	—	0.41	0.31	0.30	0.41	0.57	—	—	—	0.24	0.34	1.67
81–100	53	—	—	0.10	0.16	0.19	0.40	0.46	—	—	—	0.11	0.31	1.36

TABLE C2 | Average relative gap (in %) compared to the lower bound of the baseline-optimal CVRP instances without preprocessing.

Nodes	Instances	θ							n					
		0.005	0.01	0.03	0.05	0.1	0.2	0.3	13	11	9	7	5	3
20–50	38	—	—	—	0.84	1.27	1.82	1.02	—	—	—	—	0.71	1.30

Appendix D: Results for KNN Confidence Scores**TSP****TABLE D1** | Number of baseline-optimal TSP instances that remain optimal under preprocessing using KNN scores.

Nodes	Instances	θ							n					
		0.005	0.01	0.03	0.05	0.1	0.2	0.3	13	11	9	7	5	3
20–50	296	295	295	294	283	281	263	238	295	292	284	258	179	13
51–80	209	205	201	198	195	190	184	163	207	206	197	173	73	1
81–100	53	53	52	50	43	43	39	35	52	51	49	45	24	0

TABLE D2 | Average solver runtimes (in seconds) for optimally solved TSP instances under different preprocessing configurations using KNN scores.

Nodes	Baseline	θ							n					
		0.005	0.01	0.03	0.05	0.1	0.2	0.3	13	11	9	7	5	3
20–50	10.7	7.9	7.5	7.4	7.6	7.1	5.4	5.4	8.7	8.2	8.0	6.6	6.1	1.4
51–80	48.2	29.4	29.4	27.3	25.9	22.9	18.4	16.1	29.7	28.9	27.1	22.0	19.3	9.4
81–100	86.8	48.3	45.5	35.3	36.8	30.5	28.1	25.1	40.8	39.5	30.6	29.7	26.9	19.6

TABLE D3 | Average optimality gaps (in %) at the time limit, measured relative to the lower bound obtained without preprocessing (TSP instances not solved to optimality) using KNN scores.

Nodes	Baseline	θ							n					
		0.005	0.01	0.03	0.05	0.1	0.2	0.3	13	11	9	7	5	3
20–50	1.98	1.95	1.95	1.96	1.92	1.97	2.09	2.83	1.94	1.94	1.99	2.03	2.18	3.91
51–80	1.86	1.66	1.71	1.69	1.66	1.64	1.73	1.83	1.73	1.67	1.70	1.66	1.91	4.79
81–100	3.78	2.39	2.33	2.23	2.24	2.21	2.18	2.19	2.24	2.25	2.22	2.24	2.46	5.19

CVRP**TABLE D4** | Number of baseline-optimal CVRP instances that remain optimal under preprocessing using KNN scores.

Nodes	Instances	θ							n					
		0.005	0.01	0.03	0.05	0.1	0.2	0.3	13	11	9	7	5	3
20–50	38	38	38	36	36	31	30	29	37	37	37	36	31	11

TABLE D5 | Average solver runtimes (in seconds) for optimally solved CVRP instances under different preprocessing configurations using KNN scores.

Nodes	Baseline	θ							n					
		0.005	0.01	0.03	0.05	0.1	0.2	0.3	13	11	9	7	5	3
20–50	68.0	51.2	55.7	56.3	56.5	48.6	36.6	28.3	70.6	65.0	59.2	51.0	26.4	1.8

TABLE D6 | Average optimality gaps (in %) at the time limit, measured relative to the lower bound obtained without preprocessing (CVRP instances not solved to optimality) using KNN scores.

Nodes	Baseline	θ							n					
		0.005	0.01	0.03	0.05	0.1	0.2	0.3	13	11	9	7	5	3
20–50	13.79	13.35	13.28	13.24	13.25	13.18	13.04	13.05	13.30	13.34	13.23	13.25	13.24	14.46
51–80	29.87	25.92	25.82	25.76	25.72	25.59	25.29	24.87	25.69	25.85	25.68	25.66	25.88	26.81
81–100	38.44	32.88	32.57	32.30	32.10	31.85	31.71	31.19	32.25	32.23	32.16	32.00	32.15	33.22

Appendix E: Results for Distance-Based Pruning

Note: Results for $n = 3$ are omitted because the distance-based pruning frequently removed edges required to construct a Hamiltonian cycle, preventing the solver from finding a feasible solution. For $n = 5$, reported average gaps are computed only over instances for which a feasible solution was obtained. This behavior was not observed for the proposed machine learning-based pruning strategy.

TABLE E1 | Number of baseline-optimal TSP instances that remain optimal under preprocessing using top- n distance-based candidate lists.

Nodes	Instances	n				
		13	11	9	7	5
20–50	296	293	285	276	239	122
51–80	209	207	205	193	143	48
81–100	53	53	52	48	37	7

TABLE E2 | Average solver runtimes (in seconds) for baseline-optimal TSP instances under preprocessing using top- n distance-based candidate lists.

Nodes	Baseline	n				
		13	11	9	7	5
20–50	10.7	7.9	7.4	7.9	7.4	9.7
51–80	48.2	25.2	24.0	21.2	24.3	34.9
81–100	86.8	33.3	26.7	26.8	32.6	39.2

TABLE E3 | Average optimality gaps (in %) at the time limit, measured relative to the lower bound obtained without preprocessing, for TSP instances not solved to optimality using top- n distance-based candidate lists.

Nodes	Baseline	n				
		13	11	9	7	5
20–50	1.98	1.96	1.95	1.97	2.44	2.89
51–80	1.86	1.74	1.74	1.75	1.87	2.54
81–100	3.78	2.50	2.57	2.90	3.14	3.17