Adaptive Constrained Optimization for Neural Vehicle Routing

Anonymous Author(s)
Affiliation
Address
email

Abstract

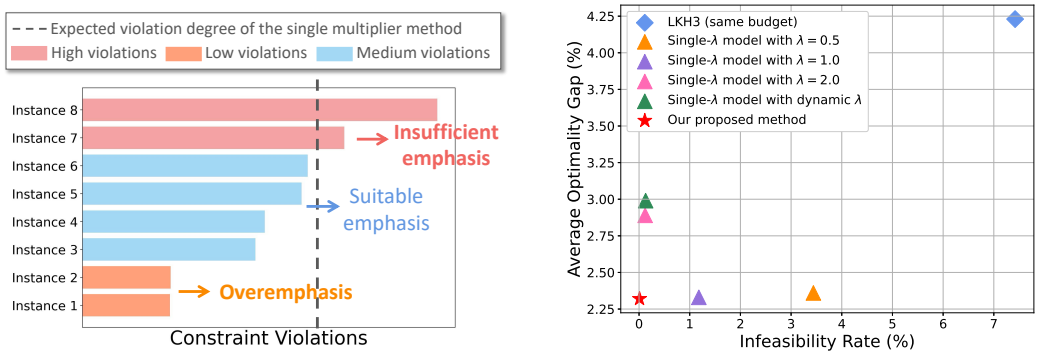
Neural solvers have shown remarkable success in tackling Vehicle Routing Problems (VRPs). However, their application to scenarios with complex real-world constraints is still at an early stage. Recent works successfully employ variants of the Lagrange multiplier method to handle such constraints, but their limitation lies in the use of a uniform multiplier across all problem instances, overlooking the fact that the difficulty of satisfying constraints varies significantly across instances. To address this limitation, we propose an instance-level adaptive constrained optimization framework that reformulates the Lagrangian dual problem by assigning each instance its own multiplier. To efficiently optimize this new problem, we design a multiplier-conditioned policy that solves instances with a controllable level of constraint awareness, which effectively decouples policy optimization from the optimization of multipliers. By leveraging this conditioned policy, we customize the optimization of multipliers for each test instance by adapting to its particular constraint violations. Experimental results on the Travelling Salesman Problem with Time Window (TSPTW), and TSP with Draft Limit (TSPDL) show that our method exhibits advantages compared to the strong solver LKH3 and significantly outperforms state-of-the-art neural methods. Our code is available at https://anonymous.4open.science/r/ICO-E52F.

1 Introduction

The Vehicle Routing Problem (VRP) is a classic kind of NP-hard combinatorial optimization problem with broad real-world applications in manufacturing [62], transportation [57], and logistics [39]. VRP solvers in the Operational Research (OR) community, which are typically based on heuristic search [29] and integer programming [4], have achieved remarkable success in the past but are often limited by high computational overheads. To address this, neural networks have been leveraged to develop efficient, data-driven heuristics for solving VRPs [64, 35, 40, 49, 37, 33, 11, 69, 45], demonstrating faster solving speeds and competitive solution quality against strong OR solvers. A prominent approach among these neural solvers is utilizing reinforcement learning-based policies to sequentially construct solutions [5], which has shown effectiveness on canonical problems like TSP and Capacitated VRP (CVRP) [41, 19, 47].

Real-world applications of VRP, however, often involve constraints that are more complex than those in the canonical problems. For example, in many business scenarios such as public transportation [12, 56] and dial-a-ride systems [16], the arrival time of vehicle must fall into a customer-requested time window, known as the time window constraint. This constraint significantly restricts the feasible region such that even finding a feasible solution is proved to be NP-complete [55], which can pose great challenges to most existing solvers. Other examples of complex constraints in VRPs include the global priority rule in disaster relief [52] and the draft limits in maritime transportation [26]. To

Submitted to 39th Conference on Neural Information Processing Systems (NeurIPS 2025). Do not distribute.



(a) (b)

Figure 1: (a) Illustration of the drawback inherent in single-multiplier (λ) methods. Constraint violations of different instances are plotted. The single-λ methods tend to overemphasize (insufficiently emphasize) constraints on some instances with relatively low (high) constraint violations. (b) Performance comparison of LKH3, single-λ models and our proposed instance-level adaptive method, on TSPDL with 50 nodes.

handle these hard constraints, classical OR solvers often employ techniques like penalty functions to incorporate constraint violations into the objective function. In the strong solver LKH3 [30], the penalty function is prioritized over the original distance cost, highlighting its emphasis on handling constraints. However, as shown in pervious works [9] and our experiments (see Table 1), the feasibility rate obtained by the traditional solvers is still unsatisfactory when runtime budgets are limited.

Neural solvers have achieved remarkable performance on various VRPs, even surpassing LKH3 on large-scale problems [48] and specific problem variants [72]. However, the research of their extension to VRPs with complex constraints is still at an early stage. To better handle complex constraints, existing studies have refined neural methods from several perspectives, including constraint-aware feature design [15], improvement in network architecture [21], modifications to the objective function [71, 14, 60], and development of novel masking mechanisms [9]. For instance, Chen et al. [15] introduced a multi-step look-ahead strategy, integrating the future time window information to enhance constraint-related features. Similarly, Bi et al. [9] designed a look-ahead-based mask mechanism to proactively exclude actions that would violate constraints in future steps. From the perspective of constrained optimization, Tang et al. [60] adopted the Lagrange multiplier method to explicitly optimize constraint violations together with the route distance. Notably, the most recent Lagrange multiplier-based implementation [9] has achieved state-of-the-art performance on common benchmarks, regarded as a general and effective solution for complex VRPs. However, these Lagrangian-based methods directly extend the canonical formulation to the optimization of neural solvers by employing a uniform multiplier across all problem instances, thereby neglecting the disparity in constraint violations among instances, as illustrated in Figure 1a. This limitation can significantly hinder the adaptability of neural models, resulting in suboptimal performance. More related works about neural solvers and constrained optimization are introduced in Appendix C.

To address this issue, we introduce a new formulation of the Lagrangian dual problem that assigns each instance a specific multiplier, enabling adaptive constrained optimization at the instance level. Compared to the methods that rely on a single multiplier, this instance-specific formulation offers greater flexibility by optimizing the trade-off between solution quality and constraint satisfaction for each individual instance. However, directly optimizing instance-specific multipliers for millions of training instances poses significant computational challenges. To address this issue, we develop a multiplier-conditioned policy that decouples the optimization of the policy from that of the multipliers, effectively reformulating the dual problem into two separate subproblems. By leveraging this conditioned policy, the outer subproblem of optimizing multipliers can be efficiently solved independently during the inference stage.

We conduct experiments on two challenging constrained VRPs: Travelling Salesman Problems with Time Window (TSPTW), and TSP with Draft Limit (TSPDL). Notably, these two problems pose greater challenges in satisfying constraints compared to CVRPTW and CVRPDL, as the

constraint violations of the latter can be addressed more easily by assigning additional vehicles to the violated nodes. The experimental results demonstrate that our adaptive optimization approach significantly surpasses the state-of-the-art neural method [9] that relies on a single multiplier. For instance, Figure 1b compares the optimality gap and infeasibility rate on TSPDL50 (TSPDL with 50 nodes), where our proposed method has clear advantages. Moveover, compared to the strong solver LKH3 under the same runtime budget, our neural method reduces the infeasibility rate by $95.56\% - 1.33\% = 94.23\%$ on TSPTW100 and $7.02\% - 0.91\% = 6.11\%$ on TSPDL100, while achieving competitive optimality gap. These results highlight neural methods as a promising alternative to OR solvers for addressing constrained VRPs.

2 Background

2.1 Constrained VRPs

The objective of VRPs [18] is to determine a tour that minimizes the total travel distance while visiting all the customer nodes. Formally, a VRP instance is defined on a graph $G = (V, E)$, where V represents the set of all customer nodes along with a depot node, and E denotes the set of directed edges between each pair of nodes (i.e., the graph is fully connected). The vehicles are required to start and end their tours at the depot node. In this paper, we focus on two types of challenging constraints: Time window constraint and draft limit constraint.

Time window. The time window constraint nartually arises in many business scenarios that require flexible time scheduling [61]. In this context, each node is accosiated with a time window $[l_i, u_i]$ that defines the earlist time l_i and the latest time u_i of visiting that node. The constraint ensures that the arrival time at each node does not exceed the end of its designated time window. If the arrival time t_i is earlier than the start time (i.e., $t_i < l_i$), the vehicle must wait until the time window starts. Formally, a TSPTW instance I is expressed as:

$$\min_{\tau} f_I(\tau) = \sum_{(n,v)\in\tau} d_I(n, v), \qquad \text{s.t.} \quad g_I(\tau) = \sum_{i=0}^{n-1} \max\{t_i - u_i, 0\} \leq 0,$$

where τ denotes the tour, and $d_I(n, v)$ is the distance between nodes n and v. The goal is to find a tour τ that minimizes the total distance $f_I(\tau)$ while satisfying the time window constraint $g_I(\tau) \leq 0$.

Draft limit. The draft limit in ports is an important factor that influences the routing actions in maritime transportation [26]. The draft of a ship is the distance between the waterline and the bottom of the ship, affected by the cumulative load. The draft limits in ports are designed to avoid overloaded ships entering these ports. In this context, each node represents a port with a maximum draft m_i and a non-negative demand δ_i. The constraint requires that the cumulative load, $c_i = \sum_{j=1}^{i-1} \delta_{\tau_j}$, over the last $i - 1$ steps must not exceed the maximum draft m_i of the i-th visited port. Formally, this can be expressed as $g_I(\tau) = \sum_{i=0}^{n-1} \max\{c_i - m_i, 0\} \leq 0$.

2.2 Lagrange Multiplier Method

To solve constrained VRPs, the constraint violation can be integrated into the objective function through the formulation of the Lagrangian dual problem [7]:

$$\max_{\lambda\geq 0} \min_{\tau} [f_I(\tau) + \lambda \cdot g_I(\tau)],$$

where λ is a non-negative dual variable (i.e., multiplier), quantifing the impact of a constraint on the objective function. The Lagrangian dual problem can be optimized by alternatively updating the primal and dual variables. This involves solving the primal problem for a fixed dual variable, which can be addressed using a classical VRP solver, followed by updating the dual variable based on the observed constraint violations [38]. The update of the dual variable is often realized using subgradient ascent as:

$$\lambda \leftarrow \lambda + \alpha \cdot g_I(\tau),$$

where α is the learning rate. Through the iterative adjustment, the dual variable is continuously refined according to the current level of constraint violation, enabling a better balance between solution

quality and constraint satisfaction. More iterative update methods for the dual variable include quadratic method [31] and proportional-integral-derivative control [58]. Compared to traditional penalty-based approaches, the Lagrange multiplier method avoids reliance on fixed penalty parameters and has the potential to yield optimal solutions if the strong duality holds [10]. However, the Lagrange multiplier method is designed to optimize an individual problem instance. A gap arises nartually when it is applied to the training process involving a large number of instances.

2.3 Lagrange Multiplier-based Training Methods for Neural Vehicle Routing

When reinforcement learning (RL) is employed to train neural networks for constructing solutions to VRPs [5], the expected return of the RL policy π_θ on a given instance I is defined as $\mathcal{J}(\pi_\theta, I) = \mathbb{E}_{\tau\sim\pi_\theta(\cdot|I)}[-f_I(\tau)]$, and the expected constraint violation is given by $\mathcal{J}_C(\pi_\theta, I) = \mathbb{E}_{\tau\sim\pi_\theta(\cdot|I)}[-g_I(\tau)]$. Using these definitions, the Lagrangian dual problem of policy optimization is formulated as

$$\min_{\lambda\geq 0}\max_{\theta}\mathbb{E}_{I\sim\mathcal{D}}[\mathcal{J}(\pi_\theta, I) + \lambda\cdot\mathcal{J}_C(\pi_\theta, I)].$$

Unlike typical constrained RL [1, 68, 28], where the focus is on solving a specific instance, the trained policy in this framework is designed to generalize to unseen instances from the same problem class. To achieve this, the training objective involves maximizing the expected performance over a distribution $\mathcal{D}$ of instances. In practice, the training process is conducted on a dataset D_I that contains a large number of synthetic problem instances.

To optimize this (or a similar) dual problem, Tang et al. [60] proposed an approach that alternatively updates the policy π_θ and the multiplier λ. Specifically, the policy π_θ is optimized by policy gradient algorithms such as REINFORCE [67], while the multiplier λ is optimized by subgradient ascent. More recently, Bi et al. [9] chose to fix the value of λ as a pre-defined constant for efficiency and scalability. In our experiments (see Table 1), we observe that dynamically updating the policy-level single λ is inferior to the fixed λ setting in most cases.

Limitations of Lagrangian-based training. The Lagrange multiplier method was originally designed for optimizing a single problem instance. However, existing approaches directly extend this method to the training of neural solvers and ties multipliers to the RL policy, forming a *policy-level* dual approach, where λ updates with policy changes but remains invariant across instances. This simple adaptation overlooks the fact that different instances can exhibit significantly varying levels of constraint violations, as demonstrated in Figure 1a, thereby resulting in suboptimal performance.

3 Method

To address the aforementioned limitations, we propose an Instance-level adaptive Constrained Optimization (ICO) method. In this section, we first provide an overview of the proposed ICO approach, followed by a detailed description of its training process and network architecture.

3.1 Instance-level Adaptive Constrained Optimization

We leverage instance-specific multipliers to effectively handle the varying degrees of constraint violations across instances, which can enable a more flexible trade-off between optimizing the objective and satisfying the constraints. Formally, the new dual problem is formulated as

$$\min_{\{\lambda_i\}_{i=1}^N}\max_{\theta}\sum_{i=1}^{N}[\mathcal{J}(\pi_\theta, I_i) + \lambda_i\cdot\mathcal{J}_C(\pi_\theta, I_i)], \tag{1}$$

where N is the number of training instances and λ_i is the dual variable specific to instance I_i. This dual formulation has the potential to simultaneously improve solution quality and enhance constraint satisfaction, provided that both the primal and dual variables are effectively optimized. However, it is extermely challenging and computationally expensive to optimize the instance-specific dual variables for **millions of training instances**. In the common training method of neural solvers [41], more than one hundred million training instances are generated on the fly, and each instance is only used once during training without additional iterations to refine its corresponding multiplier. This training process necessitates an efficient and scalable approach to adaptively manage instance-specific

multipliers. Therefore, we discard the expensive alternating update method and decouple the original bi-level optimization problem into two separate subproblems: Solve the inner subproblem of Eq. (1) as phase 1 and solve the outer subproblem based on the inner results as phase 2.

Phase 1: Solve the inner subproblem. In the first phase, we solve the inner maximization problem separately while considering varying values of λ, aiming to obtain a manifold of policies capable of solving instances with continuously varying levels of constraint awareness. To achieve this, we propose training a λ-conditioned policy $\pi_\theta(\cdot|\lambda)$ that takes λ as input and performs as trained using the specified λ, i.e.,

$$\pi_\theta(\cdot|\lambda) \approx \arg\max_{\pi} \sum_{i=1}^{N} [\mathcal{J}(\pi, I_i) + \lambda \cdot \mathcal{J}_C(\pi, I_i)],$$

where the right side represents the optimal policy corresponding to the given λ. With this condition mechanism, the constraint sensitivity of the policy can be seamlessly controlled by adjusting the input value of λ, without requiring any modification to the network parameters. This can effectively decouple the policy optimization process from the optimization of the multipliers, thereby enhancing scalability of the Lagrangian-based training method. The detailed training algorithm and network architecture for the λ-conditioned policy are provided in Section 3.2.

Phase 2: Solve the outer subproblem. The second phase is performed during the inference stage, where instance-specific λ values are optimized based on the feedback provided by the trained λ-conditioned policy. For each new instance, we iteratively update λ by subgradient ascent to minimize its specific constraint violations, thereby adjusting the policy to achieve an appropriate trade-off. This process alternates between sampling a solution using the policy $\pi_\theta(\cdot|\lambda)$ and updating λ based on the observed constraint violations. Formally, the process is described as follows:

$$\tau_{t-1} \sim \pi_\theta(\cdot|\lambda_{t-1}, I), \quad \lambda_t = \lambda_{t-1} + \alpha \cdot g_I(\tau_{t-1}),$$

where t denotes the iteration timestep, and $g_I(\tau_{t-1})$ is the constraint violation of the sampled solution. Note that we initialize all λ values using an identical λ_0. Furthermore, we also explore to utilize Proportional-Integral-Derivative (PID) control to adjust the λ-value as proposed by Stooke et al. [58], detailed in Appendix F.2.

3.2 Multiplier-Conditioned Policy

The λ-conditioned policy serves as a key component in optimizing the decoupled dual problem. We design a two-stage training algorithm for the λ-conditioned policy, consisting of a pre-training stage for efficient convergence and a fine-tuning stage to achieve a precise alignment between λ values and instance hardness, which is schematically illustrated in Figure 2. Detailed description of the two training stages is as follows.

Pre-training stage. The pre-training stage is conducted on randomly sampled λ values, which is computationally efficient and can effectively enable the generalization ability across varying λ conditions. The training objective can be expressed as

$$\max_{\theta} \mathbb{E}_{I\sim\mathcal{D}} \mathbb{E}_{\lambda\sim\mathcal{D}_\lambda} [\mathcal{J}(\pi_\theta(\cdot|\lambda), I) + \lambda \cdot \mathcal{J}_C(\pi_\theta(\cdot|\lambda), I)].$$

Specifically, we randomly sample λ_i from a pre-defined distribution $\mathcal{D}_\lambda$ for each training instance I_i, constituting a pair sample (λ_i, I_i). The reward function of the instance I_i is reweighted by its own multiplier λ_i. Following the shared baseline method [41], we sample multiple solutions $\{\tau^j\}_{j=1}^P$ for each (λ_i, I_i) pair and estimate the baseline by the average reward of these solutions. Then, we compute the policy gradient $\nabla_\theta J(\theta)$ using the REINFORCE [67] algorithm as

$$R^j = -(f_{I_i}(\tau^j) + \lambda_i \cdot (g_{I_i}(\tau^j) + c_{I_i}(\tau^j))), \forall j \in [P],$$

$$\nabla_\theta J(\theta) = \frac{1}{P} \sum_{j=1}^{P} (R^j - \frac{1}{P} \sum_{k=1}^{P} R^k) \log \pi_\theta(\tau^j|\lambda_i, I_i),$$

where $[P]$ denotes the set $\{1, ..., P\}$, and $c_{I_i}(\tau^j)$ is the number of timeout nodes, which we use as a heuristic penalty reward, following the reward design of [9]. The factor $R^j - \frac{1}{P}\sum_{k=1}^{P} R^k$ represents

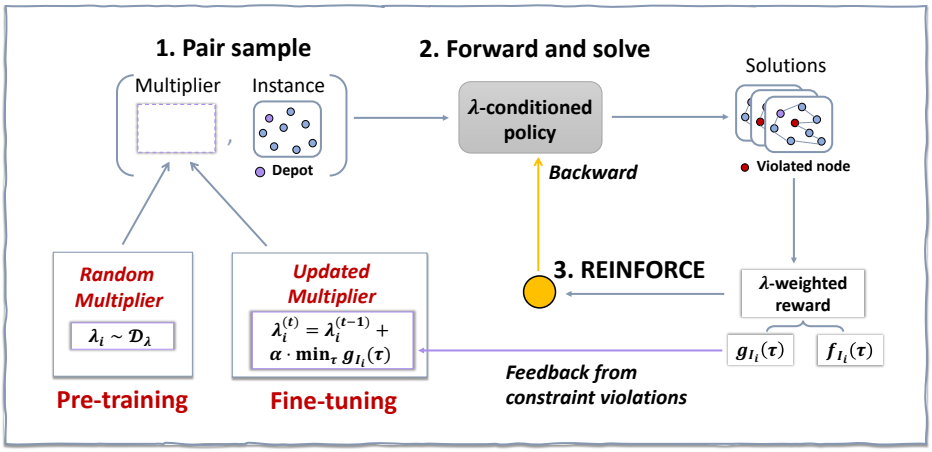


Figure 2: A sketch of the two training stages. In both stages, the λ-conditioned policy is trained using REINFORCE. The primary distinction lies in the handling of the multiplier λ. During the pre-training stage, λ is randomly sampled to facilitate early convergence and to enhance the policy's adaptability across diverse λ values. In contrast, the fine-tuning stage employs an iterative update mechanism for λ, ensuring that its values are precisely adjusted to account for constraint violations.

the *advantage* that measures relative reward improvement over the shared baseline. Intuitively, the training algorithm reinforces the probability of generating positive advantage trajectories (i.e., solutions) while decreasing the probability of generating negative ones. The pseudo code of the pre-training process is provided in Appendix B. Through this training process with random λ, the conditioned policy obtains the adaptability to different levels of constraint awareness. Additionally, this pre-training phase ensures sufficient convergence of the policy, effectively reducing the occurrence of infeasible instances to a manageable level. Once these objectives are achieved, the training transitions to the subsequent stage, where instance-specific λ values are iteratively optimized.

Fine-tuning stage. To achieve an effective alignment between λ values and instance hardness, we further fine-tune the pre-trained policy using iteratively updated λ values. In this stage, we initialize a uniform and small initial value $\lambda^{(0)}$ for all instances and alternate between optimizing the policy and updating the multipliers. For policy optimization, we continue to employ the REINFORCE algorithm with an average baseline, as used in the pre-training stage. For updating the multipliers, the subgradient is computed based on the minimal constraint violation value across a set of sampled solutions $\{\tau^j\}_{j=1}^P$. Formally, the λ values are updated by the following rule:

$$\lambda_i^{(t)} = \lambda_i^{(t-1)} + \alpha \min_{j \in [P]} (g_{I_i}(\tau^j) + c_{I_i}(\tau^j)),$$

where α is the learning rate. After each iteration, we retain the infeasible instances and their corresponding λ values in the batch while replacing the feasible instances with new ones. It is important to note that the pre-trained policy is already capable of finding feasible solutions for the majority of instances. Therefore, the proportion of infeasible instances in each batch is typically small. Moreover, to further enhance training efficiency and avoid excessive focus on particularly hard instances, we impose a maximum iteration limit and a cap on the infeasible instance ratio. The pseudo code of the fine-tuning process is provided in Appendix B.

Network architecture. The λ-conditioned policy solves instances with a controllable level of constraint awareness, determined by the condition variable λ. Similar conditioned policies have been explored in related works, particularly for multi-objective optimization [44, 66] and latent space search [13]. Among them, there are two possible ways to incorporate the target variable into the policy network: (1) embedding it into the initial input features or (2) embedding it into the decoder's context. In this paper, we adopt the λ-conditioned initial embedding, which empirically demonstrates superior performance in adjusting trade-off behaviors (see Appendix F.4). Specifically, building on the POMO model [41], we incorporate a linear transformation of λ into the original initial embeddings. The embedding is computed as:

$$\boldsymbol{h}_i^{(0)} = W^\lambda \lambda + W^h [x_i, y_i, l_i, u_i]^\top,$$

where $W^\lambda \in \mathbb{R}^{d\times 1}$ and $W^h \in \mathbb{R}^{d\times 4}$ are trainable parameters, and $[x_i, y_i, l_i, u_i]$ represents the concatenation of the node's coordinates (x_i, y_i) and its time window bounds (l_i, u_i). This concatenated feature vector serves as the input representation for each node. The output $\boldsymbol{h}_i^{(0)}$ is then used as the initial embedding for the encoder network, which employs the multi-head attention mechanism [63] to perform message passing and update node embeddings. Intuitively, the λ-conditioned embedding adjusts the relative importance of distance-related features (e.g., node coordinates) and constraint-related features (e.g., time window bounds) based on the value of λ, thereby enabling a controllable level of constraint awareness. The rest of the architecture closely follows the standard model [41].

4 Experiments

In this section, we evaluate the effectiveness of our ICO method through comparison experiments and ablation studies. Additional results are included in Appendix F due to space limitation.

4.1 Experimental Settings

Problem instance generation. Following prior works [40], we randomly sample node coordinates (x_i, y_i) from a uniform distribution $U(0, 1)$ within a square. For generating the time windows of TSPTW and draft limits of TSPDL, we utilize the code from Bi et al. [9] and adopt the **hard** settings, which are sufficiently challenging to examine state-of-the-art neural and OR solvers.

Implementation details. Our model is implemented based on the POMO framework [41], incorporating the PI mask [9] to restrict the search space. We only employ the PIP decoder to predict masks during the training process on instances with the number of nodes $n = 100$. The prior distribution of λ in the pre-training stage, i.e., $\mathcal{D}(\lambda)$, is set to a triangular distribution $T(0.1, 0.5, 2.0)$. The learning rate for updating λ is set to 0.5 for TSPTW and 0.2 for TSPDL. The common hyperparameters shared between our method and prior works follow their default settings [41, 9]. In evaluation, our method employs $\times$ 8 instance augmentation and 16 iterations to update λ during the inference stage. To align the runtime consumption, we use sampling strategies for PIP. More implementation details are provided in Appendix E due to space limitation.

Baselines. We compare our proposed method against state-of-the-art neural methods and OR solvers. For OR solvers, we include LKH3 [30], one of the strongest solver specifically designed for VRPs; and OR-Tools [20], a general-purpose solver capable of handling various constraints. For neural methods, we consider the state-of-the-art PIP framework [9]. For TSPTW(DL)100, we report the results of the models with the PIP decoder. Our experiments **encompass four configurations of PIP**: $\lambda = 0.5$, $\lambda = 1.0$, $\lambda = 2.0$, and a dynamically updated λ. Specifically, in the dynamic setting, the value of λ is periodically adjusted using subgradient ascent every 1000 epochs. The subgradient is estimated based on the average constraint violation observed on the validation dataset.

Metrics. We evaluate performance and efficiency using four metrics: infeasibility rate, average optimality gap, normalized HyperVolume (HV) and runtime. Among these, the HV serves as a comprehensive indicator, capturing both feasibility and solution quality. A detailed explanation of these metrics is provided in Appendix E.3.

4.2 Main Results

Comparison with single-λ models. The performance comparisons on TSPTW and TSPDL across different problem scales are presented in Table 1. On TSPTW100, the proposed ICO method reduces the infeasibility rate from 4.33% (achieved by POMO+PIP with $\lambda = 1.0$) to an impressive 1.33%, representing a substantial reduction of 3.00%. Similarly, on TSPTW50, the infeasibility rate is lowered from 1.56% to just 0.50%. Even when the λ value in single-λ models is increased to 2.0, these models still lag behind the ICO method in terms of feasibility, with the sole exception being TSPDL100. In addition to improving feasibility rates, the ICO method consistently outperforms single-λ models in terms of optimality gaps. For instance, the ICO method achieves a smaller gap of 9.22% on TSPDL100, compared to 10.77% achieved by the best POMO+PIP model. Moreover, the ICO method showcases the highest HV scores on all benchmarks, indicating its superior trade-off performance in balancing solution quality and constraint satisfaction.

Table 1: Experimental results on TSPTW and TSPDL. Test instances are generated using the hard settings [9]. LKH3 (less time) and OR-Tools (less time) denote the OR methods with reduced runtime budgets to align with neural solvers. The best and the runner-up results are highlighted in **Blue** and **Violet**, respectively.

Methods	TSPTW50				TSPTW100			
	Inf. Rate ↓	Avg. Gap ↓	HV ↑	Time ↓	Inf. Rate ↓	Avg. Gap ↓	HV ↑	Time ↓
LKH3	0.12%	0.0%	1.00	7h	0.07%	0.0%	1.00	1.4d
OR-Tools	65.72%	0.0%	0.34	2.4h	89.07%	0.0%	0.11	1.6d
LKH3 (less time)	57.34%	**0.01%**	0.43	100s	95.56%	**0.03%**	0.04	8m
OR-Tools (less time)	65.72%	**0.02%**	0.34	99s	89.07%	0.51%	0.10	8m
AM + PIP ($\lambda = 1.0$)	2.99%	0.34%	0.90	105s	7.80%	0.70%	0.79	8m
POMO + PIP ($\lambda = 0.5$)	1.95%	0.08%	**0.96**	108s	4.90%	0.17%	**0.92**	9m
POMO + PIP ($\lambda = 1.0$)	1.56%	0.16%	0.95	108s	**4.33%**	0.25%	0.91	9m
POMO + PIP ($\lambda = 2.0$)	1.41%	0.19%	0.95	108s	4.71%	0.39%	0.88	9m
POMO + PIP (dynamic λ)	**0.98%**	0.13%	0.93	108s	4.94%	0.45%	0.87	9m
ICO (Ours)	**0.50%**	0.07%	**0.98**	91s	**1.33%**	**0.14%**	**0.96**	8m

Methods	TSPDL50				TSPDL100			
	Inf. Rate ↓	Avg. Gap ↓	HV ↑	Time ↓	Inf. Rate ↓	Avg. Gap ↓	HV ↑	Time ↓
LKH3	0.0%	0.0%	1.00	6.8h	0.0%	0.0%	1.00	1.2d
OR-Tools	100.0%	/	/	10.6s	100.0%	/	/	56.8s
LKH3 (less time)	7.42%	4.23%	0.20	70s	7.02%	**6.76%**	0.20	6m
OR-Tools (less time)	100.0%	/	/	3s	100.0%	/	/	29s
POMO + PIP ($\lambda = 0.5$)	3.44%	2.36%	0.58	71s	62.94%	20.95%	/	5m
POMO + PIP ($\lambda = 1.0$)	1.18%	**2.33%**	0.78	71s	3.23%	10.77%	0.31	5m
POMO + PIP ($\lambda = 2.0$)	**0.12%**	2.89%	**0.85**	71s	**0.11%**	12.24%	**0.38**	5m
POMO + PIP (dynamic λ)	0.13%	2.99%	0.84	71s	**0.01%**	14.78%	0.26	5m
ICO (Ours)	**0.01%**	**2.32%**	**0.88**	69s	0.91%	**9.22%**	**0.49**	5m

Comparion with strong OR solvers. In Table 1, we also compare our neural methods with strong OR solvers, LKH3 and OR-Tools, under aligned runtime conditions. The results show that our ICO method achieves a dramatic improvement in infeasibility rates, reducing them from 95.56% to 1.33% (94.23% reduction) on TSPTW100 and from 7.02% to 0.91% (6.11% reduction) on TSPDL100. Regarding solution quality, our method significantly outperforms OR-Tools on TSPTW100 and even surpasses LKH3 on TSPDL50. While the solution quality of our neural approach on the other three benchmarks still lags behind LKH3, the substantial improvements in feasibility and competitive performance overall underscore the strengths of our neural method.

Comparison under different inference strategies. In Table 2, we extend the scope of our comparative experiments to incorporate additional inference strategies, including *Greedy* (vs. $T = 1$), *Sampling* (vs. $T > 1$), and *Efficient Active Search (EAS)* [32]. The results consistently demonstrate that ICO outperforms the best-performing PIP model (denoted as PIP*) in most scenarios. In particular, our ICO integrates well with *EAS*, achieving near-zero infeasibility rates and gaps. The only exception on ICO ($T = 1$) can be attributed to the small initial λ value. Notably, we observe that ICO ($T = 2$) even surpasses PIP* (*Sampling* 16) while consuming much less runtime, which highlights the superiority of our proposed ICO. To defense the prolonged runtime of ICO, we further compare ICO ($T = 16$) with LKH3 post search. The results indicate that even adding a strong post-search such as LKH3 to the baseline, our ICO method remains superior in reducing infeasibility.

Analysis of anytime performance. During inference, our ICO method iteratively samples solutions and updates λ, making the anytime performance throughout the iterative process a critical factor. Figure 3 shows the convergence curves of infeasibility rate and average optimality gap on TSPTW50 and TSPTW100. The results indicate that, while ICO starts with a higher infeasibility rate, it converges rapidly and outperforms single-λ models in later iterations. In terms of optimality gap, ICO consistently achieves better results throughout the process.

Extension to more problem variants. Our proposed method can be seamlessly extended to solve more VRP variants. In Appendix F.1, we conduct comparison experiments on two kinds of CVRPTW. The results show that our proposed ICO method still have advantages on CVRPTW.

Table 2: Comparisons under different inference strategies, including *Greedy*, *Sampling*, *Efficient Active Search (EAS)* [32] and *LKH3 post search*. When *EAS* is integrated, the number of parallel solutions (i.e., POMO size) is increased to 10 for estimating the average baseline. We select the best PIP model from the four configurations according to the HV metric, denoted by PIP*. For LKH3 post search, the solutions generated by PIP*(*Greedy*) are used as the initial solutions of LKH3. *Sampling/EAS* 16 refers to conducting 16 iterations, while T represents the iteration count of our ICO method. The best and the runner-up results are highlighted in **Blue** and **Violet**, respectively.

Methods	TSPTW50 (10k instances)				TSPTW100 (1k instances)			
	Inf. Rate ↓	Avg. Gap ↓	HV ↑	Time ↓	Inf. Rate ↓	Avg. Gap ↓	HV ↑	Time ↓
PIP* (*Greedy*)	3.05%	0.22%	0.927	9s	9.00%	0.23%	0.868	4s
PIP* (*Sampling* 2)	2.53%	0.10%	0.955	14s	7.20%	0.22%	0.887	7s
PIP* (*Sampling* 16)	2.11%	0.09%	0.961	63s	5.80%	0.19%	0.906	43s
PIP* (*EAS* 16)	1.22%	0.05%	0.978	11m	**0.50%**	0.04%	**0.987**	9m
PIP* (*Greedy*) + LKH3	1.40%	**0.01%**	**0.984**	100s	6.17%	**-0.07%**	0.951	50s
ICO ($T = 1$)	2.14%	0.10%	0.959	9s	14.10%	0.15%	0.833	4s
ICO ($T = 2$)	1.67%	0.09%	0.966	14s	3.60%	0.17%	0.931	7s
ICO ($T = 16$)	**0.50%**	0.07%	**0.981**	90s	1.10%	0.14%	0.961	48s
ICO (*EAS* 16)	**0.17%**	**0.03%**	**0.993**	11m	**0.20%**	**0.02%**	**0.994**	9m

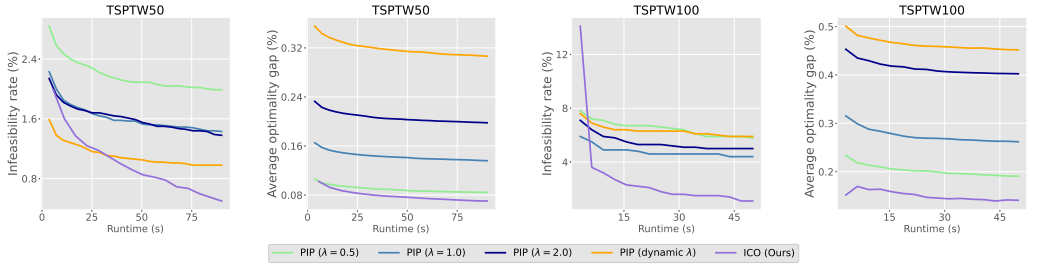


Figure 3: Anytime performance comparion between our ICO method and the single-λ methods.

4.3 Ablation Study

In this subsection, we present a series of experiments to investigate the impact of each component. Detailed results and analyses are provided in Appendix F due to space limitation.

- **Analysis of update rules for λ in inference stage.** See Appendix F.2.
- **Analysis of training strategies.** See Appendix F.3.
- **Analysis of the λ-conditioned network architecture.** See Appendix F.4.
- **Analysis of the pre-defined λ distribution.** See Appendix F.5.
- **Sensitivity analysis of λ-related hyperparameters.** See Appendix F.6.

5 Conclusion

In this paper, we propose a novel approach ICO to address the limitations of existing Lagrangian-based neural methods in solving complex constained VRPs. Unlike prior methods that rely on a single, uniform multiplier across all problem instances, ICO leverages instance-specific multipliers to improve adaptability and better optimize the trade-off between solution quality and constraint satisfaction for every problem instance. Experimental results on two challenging constrained VRP benchmarks, TSPTW and TSPDL, demonstrate that ICO significantly reduces infeasibility rates compared to both state-of-the-art neural methods and strong OR solvers like LKH3. These empirical findings suggest that our ICO framework can be a promising alternative for strong OR solvers when tackling constrained combinatorial problems. **One limitation of this study** lies in the fact that the proposed ICO framework necessitates a minimum of two iterations to update the λ values, resulting in an extended inference runtime. Future research could explore methods for directly predicting optimal λ values, improving the training strategies of the conditioned policy, and enabling generalization across diverse sets of constraints.

References

[1] Joshua Achiam, David Held, Aviv Tamar, and Pieter Abbeel. Constrained policy optimization. In *Proceedings of the 34th International Conference on Machine Learning (ICML)*, pages 22–31, Sydney, Australia, 2017.

[2] Utku Umur Acikalin, Aaron M. Ferber, and Carla P. Gomes. Learning to explore and exploit with gnns for unsupervised combinatorial optimization. In *The 13th International Conference on Learning Representations (ICLR)*, Singapore, 2025.

[3] Sungsoo Ahn, Younggyo Seo, and Jinwoo Shin. Learning what to defer for maximum independent sets. In *Proceedings of the 37th International Conference on Machine Learning, (ICML)*, pages 134–144, Virtual, 2020.

[4] David Applegate, Robert Bixby, Vasek Chvatal, and William Cook. Concorde TSP solver. `http://www.math.uwaterloo.ca/tsp/concorde/m`, 2006.

[5] Irwan Bello, Hieu Pham, Quoc V Le, Mohammad Norouzi, and Samy Bengio. Neural combinatorial optimization with reinforcement learning. In *Proceedings of the 5th International Conference on Learning Representations (ICLR)*, Toulon, France, 2017.

[6] Yoshua Bengio, Andrea Lodi, and Antoine Prouvost. Machine learning for combinatorial optimization: A methodological tour d'horizon. *European Journal of Operational Research*, 290(2):405–421, 2021.

[7] Dimitri P Bertsekas. *Constrained optimization and Lagrange multiplier methods*. Academic press, 2014.

[8] Jieyi Bi, Yining Ma, Jiahai Wang, Zhiguang Cao, Jinbiao Chen, Yuan Sun, and Yeow Meng Chee. Learning generalizable models for vehicle routing problems via knowledge distillation. In *Advances in Neural Information Processing Systems 35 (NeurIPS)*, pages 31226–31238, New Orleans, LA, 2022.

[9] Jieyi Bi, Yining Ma, Jianan Zhou, Wen Song, Zhiguang Cao, Yaoxin Wu, and Jie Zhang. Learning to handle complex constraints for vehicle routing problems. In *Advances in Neural Information Processing Systems 37 (NeurIPS)*, 2024.

[10] Stephen P. Boyd and Lieven Vandenberghe. *Convex Optimization*. Cambridge University Press, 2014.

[11] Quentin Cappart, Didier Chételat, Elias B Khalil, Andrea Lodi, Christopher Morris, and Petar Veličković. Combinatorial optimization and reasoning with graph neural networks. *Journal of Machine Learning Research*, 24(130):1–61, 2023.

[12] Diego Cattaruzza, Nabil Absi, Dominique Feillet, and Jesús González-Feliu. Vehicle routing problems for city logistics. *EURO Journal on Transportation and Logistics*, 6(1):51–79, 2017.

[13] Félix Chalumeau, Shikha Surana, Clément Bonnet, Nathan Grinsztajn, Arnu Pretorius, Alexandre Laterre, and Tom Barrett. Combinatorial optimization with policy adaptation using latent space search. In *Advances in Neural Information Processing Systems 36 (NeurIPS)*, pages 7947–7959, New Orleans, LA, 2023.

[14] Jinbiao Chen, Huanhuan Huang, Zizhen Zhang, and Jiahai Wang. Deep reinforcement learning with two-stage training strategy for practical electric vehicle routing problem with time windows. In *Proceedings of the 17th International Conference on Parallel Problem Solving from Nature (PPSN)*, volume 13398, pages 356–370, Dortmund, Germany, 2022.

[15] Jingxiao Chen, Ziqin Gong, Minghuan Liu, Jun Wang, Yong Yu, and Weinan Zhang. Looking ahead to avoid being late: Solving hard-constrained traveling salesman problem. *arxiv:2403.05318*, 2024.

[16] Jean-François Cordeau and Gilbert Laporte. The dial-a-ride problem (DARP): Variants, modeling issues and algorithms. *Quarterly Journal of the Belgian, French and Italian Operations Research Societies*, 1:89–101, 2003.

[17] Rodrigo Ferreira da Silva and Sebastián Urrutia. A general VNS heuristic for the traveling salesman problem with time windows. *Discrete Optimization*, 7(4):203–211, 2010.

[18] George B Dantzig and John H Ramser. The truck dispatching problem. *Management Science*, 6 (1):80–91, 1959.

[19] Darko Drakulic, Sofia Michel, Florian Mai, Arnaud Sors, and Jean-Marc Andreoli. BQ-NCO: Bisimulation quotienting for generalizable neural combinatorial optimization. In *Advances in Neural Information Processing Systems 36 (NeurIPS)*, pages 77416–77429, New Orleans, LA, 2023.

[20] Jonas K Falkner and Lars Schmidt-Thieme. OR-Tools routing library. URL `https://developers.google.com/optimization/routing/`.

[21] Jonas K. Falkner and Lars Schmidt-Thieme. Learning to solve vehicle routing problems with time windows through joint attention. *arXiv:2006.09100*, 2020.

[22] Han Fang, Zhihao Song, Paul Weng, and Yutong Ban. INViT: A generalizable routing problem solver with invariant nested view transformer. In *Proceedings of the 41st International Conference on Machine Learning (ICML)*, pages 12973–12992, Vienna, Austria, 2024.

[23] Zhang-Hua Fu, Kai-Bin Qiu, and Hongyuan Zha. Generalize a small pre-trained model to arbitrarily large TSP instances. In *Proceedings of the 35th AAAI Conference on Artificial Intelligence (AAAI)*, pages 7474–7482, Virtual, 2021.

[24] Chengrui Gao, Haopu Shang, Ke Xue, Dong Li, and Chao Qian. Towards generalizable neural solvers for vehicle routing problems via ensemble with transferrable local policy. In *Proceedings of the 33rd International Joint Conference on Artificial Intelligence (IJCAI)*, pages 6914–6922, Jeju, Korea, 2024.

[25] Chengrui Gao, Haopu Shang, Ke Xue, and Chao Qian. Neural solver selection for combinatorial optimization. In *Proceedings of the 42nd International Conference on Machine Learning (ICML)*, Vancouver, Canada, 2025.

[26] Jørgen Glomvik Rakke, Marielle Christiansen, Kjetil Fagerholt, and Gilbert Laporte. The traveling salesman problem with draft limits. *Computers & Operations Research*, 39(9):2161–2167, 2012.

[27] Nathan Grinsztajn, Daniel Furelos-Blanco, Shikha Surana, Clément Bonnet, and Tom Barrett. Winner takes it all: Training performant RL populations for combinatorial optimization. In *Advances in Neural Information Processing Systems 36 (NeurIPS)*, pages 48485–48509, New Orleans, LA, 2023.

[28] Shangding Gu, Long Yang, Yali Du, Guang Chen, Florian Walter, Jun Wang, and Alois Knoll. A review of safe reinforcement learning: Methods, Theories, and Applications. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 46(12):11216–11235, 2024.

[29] Keld Helsgaun. An effective implementation of the Lin–Kernighan traveling salesman heuristic. *European Journal of Operational Research*, 126(1):106–130, 2000.

[30] Keld Helsgaun. An extension of the Lin-Kernighan-Helsgaun TSP solver for constrained traveling salesman and vehicle routing problems. Technical report, 2017.

[31] Magnus R Hestenes. Multiplier and gradient methods. *Journal of optimization theory and applications*, 4(5):303–320, 1969.

[32] André Hottung, Yeong-Dae Kwon, and Kevin Tierney. Efficient active search for combinatorial optimization problems. In *Proceedings of the 10th International Conference on Learning Representations (ICLR)*, Virtual, 2022.

[33] Yuan Jiang, Yaoxin Wu, Zhiguang Cao, and Jie Zhang. Learning to solve routing problems via distributionally robust optimization. In *Proceedings of the 36th AAAI Conference on Artificial Intelligence (AAAI)*, pages 9786–9794, Virtual, 2022.

[34] Yuan Jiang, Zhiguang Cao, Yaoxin Wu, Wen Song, and Jie Zhang. Ensemble-based deep reinforcement learning for vehicle routing problems under distribution shift. In *Advances in Neural Information Processing Systems 36 (NeurIPS)*, pages 53112–53125, New Orleans, LA, 2023.

[35] Chaitanya K. Joshi, Thomas Laurent, and Xavier Bresson. An efficient graph convolutional network technique for the travelling salesman problem. *arXiv:1906.01227*, 2019.

[36] Nikolaos Karalias and Andreas Loukas. Erdos goes neural: an unsupervised learning framework for combinatorial optimization on graphs. In *Advances in Neural Information Processing Systems 33 (NeurIPS)*, Vancouver, Canada, 2020.

[37] Minsu Kim, Jinkyoo Park, and joungho kim. Learning collaborative policies to solve NP-hard routing problems. In *Advances in Neural Information Processing Systems 34 (NeurIPS)*, pages 10418–10430, Virtual, 2021.

[38] Niklas Kohl and Oli BG Madsen. An optimization algorithm for the vehicle routing problem with time windows based on lagrangian relaxation. *Operations Research*, 45(3):395–406, 1997.

[39] Grigorios D Konstantakopoulos, Sotiris P Gayialis, and Evripidis P Kechagias. Vehicle routing problem and related algorithms for logistics distribution: A literature review and classification. *Operational Research*, 22(3):2033–2062, 2022.

[40] Wouter Kool, Herke van Hoof, and Max Welling. Attention, learn to solve routing problems! In *Proceedings of the 7th International Conference on Learning Representations (ICLR)*, New Orleans, LA, 2019.

[41] Yeong-Dae Kwon, Jinho Choo, Byoungjip Kim, Iljoo Yoon, Youngjune Gwon, and Seungjai Min. POMO: Policy optimization with multiple optima for reinforcement learning. In *Advances in Neural Information Processing Systems 33 (NeurIPS)*, pages 21188–21198, Virtual, 2020.

[42] Yao Lai, Yao Mu, and Ping Luo. MaskPlace: Fast chip placement via reinforced visual representation learning. In *Advances in Neural Information Processing Systems 35 (NeurIPS)*, New Orleans, LA, 2022.

[43] Yang Li, Jinpei Guo, Runzhong Wang, and Junchi Yan. T2T: From distribution learning in training to gradient search in testing for combinatorial optimization. In *Advances in Neural Information Processing Systems 36 (NeurIPS)*, pages 50020–50040, New Orleans, LA, 2023.

[44] Xi Lin, Zhiyuan Yang, and Qingfu Zhang. Pareto set learning for neural multi-objective combinatorial optimization. In *Proceedings of the 10th International Conference on Learning Representations (ICLR)*, Virtual, 2022.

[45] Fei Liu, Xi Lin, Zhenkun Wang, Qingfu Zhang, Xialiang Tong, and Mingxuan Yuan. Multi-task learning for routing problem with cross-problem zero-shot generalization. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD)*, pages 1898–1908, Barcelona, Spain, 2024.

[46] Hao Lu, Xingwen Zhang, and Shuang Yang. A learning-based iterative method for solving vehicle routing problems. In *Proceedings of the 7th International Conference on Learning Representations (ICLR)*, New Orleans, LA, 2019.

[47] Fu Luo, Xi Lin, Fei Liu, Qingfu Zhang, and Zhenkun Wang. Neural combinatorial optimization with heavy decoder: Toward large scale generalization. In *Advances in Neural Information Processing Systems 36 (NeurIPS)*, pages 8845–8864, New Orleans, LA, 2023.

[48] Fu Luo, Xi Lin, Yaoxin Wu, Zhenkun Wang, Tong Xialiang, Mingxuan Yuan, and Qingfu Zhang. Boosting neural combinatorial optimization for large-scale vehicle routing problems. In *Proceedings of the 13th International Conference on Learning Representations (ICLR)*, Singapore, 2025.

[49] Yining Ma, Jingwen Li, Zhiguang Cao, Wen Song, Le Zhang, Zhenghua Chen, and Jing Tang. Learning to iteratively solve routing problems with dual-aspect collaborative transformer. In *Advances in Neural Information Processing Systems 34 (NeurIPS)*, pages 11096–11107, Virtual, 2021.

[50] Yining Ma, Zhiguang Cao, and Yeow Meng Chee. Learning to search feasible and infeasible regions of routing problems with flexible neural k-Opt. In *Advances in Neural Information Processing Systems 36 (NeurIPS)*, pages 49555–49578, New Orleans, LA, 2023.

[51] Mohammadreza Nazari, Afshin Oroojlooy, Martin Takáč, and Lawrence V Snyder. Reinforcement learning for solving the vehicle routing problem. In *Advances in Neural Information Processing Systems 31 (NeurIPS)*, pages 9861–9871, Montréal, Canada, 2018.

[52] Kiran Venkata Panchamgam. *Essays in retail operations and humanitarian logistics*. PhD thesis, Robert H. Smith School of Business, University of Maryland, College Park, 2011.

[53] Seonho Park and Pascal Van Hentenryck. Self-supervised primal-dual learning for constrained optimization. In *Proceedings of the 37th AAAI Conference on Artificial Intelligence (AAAI)*, pages 4052–4060, Washington, DC, 2023.

[54] Sebastian Sanokowski, Sepp Hochreiter, and Sebastian Lehner. A diffusion model framework for unsupervised neural combinatorial optimization. In *Proceedings of the 41st International Conference on Machine Learning (ICML)*, Vienna, Austria, 2024.

[55] Martin WP Savelsbergh. Local search in routing problems with time windows. *Annals of Operations Research*, 4:285–305, 1985.

[56] Reza Shahin, Pierre Hosteins, Paola Pellegrini, Pierre-Olivier Vandanjon, and Luca Quadrifoglio. A survey of flex-route transit problem and its link with vehicle routing problem. *Transportation Research Part C: Emerging Technologies*, 158:104437, 2024.

[57] David M Stein. Scheduling dial-a-ride transportation systems. *Transportation Science*, 12(3): 232–249, 1978.

[58] Adam Stooke, Joshua Achiam, and Pieter Abbeel. Responsive safety in reinforcement learning by PID lagrangian methods. In *Proceedings of the 37th International Conference on Machine Learning (ICML)*, volume 119, pages 9133–9143, Virtual Event, 2020.

[59] Zhiqing Sun and Yiming Yang. DIFUSCO: Graph-based diffusion solvers for combinatorial optimization. In *Advances in Neural Information Processing Systems 36 (NeurIPS)*, pages 3706–3731, New Orleans, LA, 2023.

[60] Qiaoyue Tang, Yangzhe Kong, Lemeng Pan, and Choonmeng Lee. Learning to solve soft-constrained vehicle routing problems with lagrangian relaxation. *arXiv:2207.09860*, 2022.

[61] Paolo Toth and Daniele Vigo. *Vehicle routing: problems, methods, and applications*. SIAM, 2014.

[62] Stefan Treitl, Pamela C Nolz, and Werner Jammernegg. Incorporating environmental aspects in an inventory routing problem. a case study from the petrochemical industry. *Flexible Services and Manufacturing Journal*, 26:143–169, 2014.

[63] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems 30 (NeurIPS)*, pages 5998–6008, Long Beach, CA, 2017.

[64] Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. Pointer networks. In *Advances in Neural Information Processing Systems 28 (NeurIPS)*, pages 2692–2700, Montreal, Canada, 2015.

[65] Haoyu Wang, Nan Wu, Hang Yang, Cong Hao, and Pan Li. Unsupervised learning for combinatorial optimization with principled objective relaxation. In *Advances in Neural Information Processing Systems 35 (NeurIPS)*, New Orleans, LA, 2022.

[66] Zhenkun Wang, Shunyu Yao, Genghui Li, and Qingfu Zhang. Multiobjective combinatorial optimization using a single deep reinforcement learning model. *IEEE Transactions on Cybernetics*, 54(3):1984–1996, 2024.

[67] Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8(3):229–256, 1992.

[68] Yihang Yao, Zuxin Liu, Zhepeng Cen, Jiacheng Zhu, Wenhao Yu, Tingnan Zhang, and Ding Zhao. Constraint-conditioned policy optimization for versatile safe reinforcement learning. In *Advances in Neural Information Processing Systems 36 (NeurIPS)*, New Orleans, LA, 2023.

[69] Haoran Ye, Jiarui Wang, Helan Liang, Zhiguang Cao, Yong Li, and Fanzhang Li. GLOP: Learning global partition and local construction for solving large-scale routing problems in real-time. In *Proceedings of the 38th AAAI Conference on Artificial Intelligence (AAAI)*, pages 20284–20292, Vancouver, Canada, 2024.

[70] Cong Zhang, Wen Song, Zhiguang Cao, Jie Zhang, Puay Siew Tan, and Xu Chi. Learning to dispatch for job shop scheduling via deep reinforcement learning. In *Advances in Neural Information Processing Systems 33 (NeurIPS)*, pages 1621–1632, Vancouver, Canada, 2020.

[71] Rongkai Zhang, Anatolii Prokhorchuk, and Justin Dauwels. Deep reinforcement learning for traveling salesman problem with time windows and rejections. In *Proceedings of the International Joint Conference on Neural Networks (IJCNN)*, pages 1–8, Glasgow, United Kingdom, 2020.

[72] Zhi Zheng, Shunyu Yao, Zhenkun Wang, Tong Xialiang, Mingxuan Yuan, and Ke Tang. DPN: Decoupling partition and navigation for neural solvers of min-max vehicle routing problems. In *Proceedings of the 41st International Conference on Machine Learning (ICML)*, Vienna, Austria, 2024.

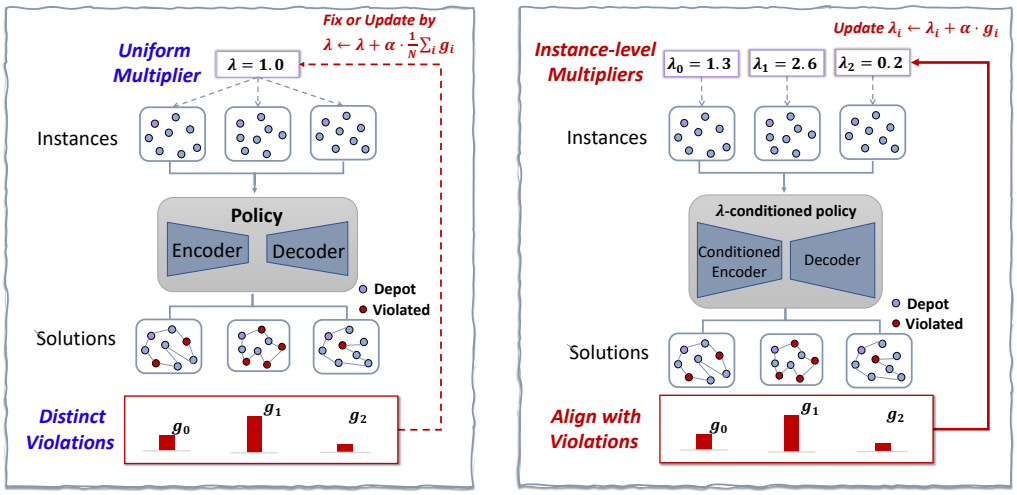
[73] Changliang Zhou, Xi Lin, Zhenkun Wang, Xialiang Tong, Mingxuan Yuan, and Qingfu Zhang. Instance-conditioned adaptation for large-scale generalization of neural combinatorial optimization. *arXiv:2405.01906*, 2024.

[74] Jianan Zhou, Yaoxin Wu, Wen Song, Zhiguang Cao, and Jie Zhang. Towards omni-generalizable neural methods for vehicle routing problems. In *Proceedings of the 40th International Conference on Machine Learning (ICML)*, pages 42769–42789, Honolulu, HI, 2023.

[75] Jianan Zhou, Zhiguang Cao, Yaoxin Wu, Wen Song, Yining Ma, Jie Zhang, and Chi Xu. MVMoE: Multi-task vehicle routing solver with mixture-of-experts. In *Proceedings of the 41th International Conference on Machine Learning (ICML)*, pages 61804–61824, Vienna, Austria, 2024.

[76] Jianan Zhou, Yaoxin Wu, Zhiguang Cao, Wen Song, Jie Zhang, and Zhiqi Shen. Collaboration! towards robust neural methods for routing problems. In *Advances in Neural Information Processing Systems 37 (NeurIPS)*, Vancouver, Canada, 2024.

A Illustration of our proposed method



(a) Previous policy-level multiplier methods **(b) Our proposed instance-level adaptive method**

Figure 4: An illustration is presented to compare previous *policy-level* multiplier methods with our proposed *instance-level* adaptive approach. The *policy-level* methods are limited in their ability to address distinct constraint violations across diverse instances, as they apply uniform multipliers irrespective of instance-specific variations. In contrast, our *instance-level* method inherently aligns multiplier values with the specific constraint violations of each instance, thereby achieving more precise and adaptive handling of constraints.

B Pseudo Code of the Training Process

Algorithm 1 Pre-training of the λ-conditioned policy

Input: Distribution D_λ, number of batches T, batch size B, number of parallel sampling P
Initialize policy network parameters θ
for $t = 0$ **to** $T - 1$ **do**
  Generate a batch of instances $\{I_i\}_{i=1}^{B}$
  Sample multipliers $\lambda_i \sim D_\lambda, \quad \forall i \in \{1, ..., B\}$
  Sample multiple solutions $\{\tau_i^j\}_{j=1}^{P} \sim \pi_\theta(\cdot|\lambda_i, I_i), \quad \forall i \in \{1, ..., B\}$
  Compute baseline $b_i \leftarrow \frac{1}{P}\sum_{j=1}^{P} -(f_{I_i}(\tau_i^j) + \lambda_i(g_{I_i}(\tau_i^j) + c_{I_i}(\tau_i^j))), \quad \forall i \in \{1, ..., B\}$
  Compute policy gradient $\nabla_\theta J(\theta) \leftarrow \frac{1}{BP}\sum_{i=1}^{B}\sum_{j=1}^{P}(-(f_{I_i}(\tau_i^j) + \lambda_i(g_{I_i}(\tau_i^j) + c_{I_i}(\tau_i^j))) - b_i)\nabla_\theta \log \pi_\theta(\tau_i^j|\lambda_i, I_i)$
  Update parameters $\theta \leftarrow \theta + \alpha\nabla_\theta J(\theta)$
end for
Output: θ

C Related Works

Prevalent paradigms of neural VRP. Many researchers have focused on end-to-end neural methods that learn to generate solutions through deep neural networks [6, 11]. These neural solvers can be categorized into three paradigms [50]: (1) **Learn-to-Construct (L2C) methods** sequentially extends solutions from scratch in an autoregressive manner, typically trained via reinforcement learning [51] or imitation learning [19]. These L2C methods have proven to be applicable to a variety of combinatorial problems [70] and industrial applications [42]. (2) **Learn-to-Predict (L2P) methods** operate under a variable-independent assumption, directly predicting the entire solution

Algorithm 2 Fine-tuning of the λ-conditioned policy

Input: Number of batches T, batch size B, number of parallel sampling P, multiplier learning rate α_λ, policy learning rate α, maximum number of itertations K, maximum infeasible ratio δ
Initialize policy network parameters θ
Generate a batch of instances $\{I_i\}_{i=1}^{B}$
Initialize multipliers $\lambda_i \leftarrow \lambda^{(0)}, \quad \forall i \in \{1, ..., B\}$
Initialize iteration counts $k_i \leftarrow 0, \quad \forall i \in \{1, ..., B\}$
for $t = 0$ **to** $T - 1$ **do**
  Sample multiple solutions $\{\tau_i^j\}_{j=1}^{P} \sim \pi_\theta(\cdot|\lambda_i, I_i), \quad \forall i \in \{1, ..., B\}$
  Compute baseline $b_i \leftarrow \frac{1}{P}\sum_{j=1}^{P} -(f_{I_i}(\tau_i^j) + \lambda_i(g_{I_i}(\tau_i^j) + c_{I_i}(\tau_i^j))), \quad \forall i \in \{1, ..., B\}$
  Compute policy gradient $\nabla_\theta J(\theta) \leftarrow \frac{1}{BP}\sum_{i=1}^{B}\sum_{j=1}^{P}(-(f_{I_i}(\tau_i^j) + \lambda_i(g_{I_i}(\tau_i^j) + c_{I_i}(\tau_i^j))) - b_i)\nabla_\theta \log \pi_\theta(\tau_i^j|\lambda_i, I_i)$
  Update parameters $\theta \leftarrow \theta + \alpha\nabla_\theta J(\theta)$
  Adjust the maximum number of iterations K according to the current infeasibility ratio, ensuring the ratio of retained infeasible instances does not exceed the maximum ratio δ
  for each instance I_j without feasible solutions **do**
    Update $\lambda_j \leftarrow \lambda_j + \alpha_\lambda \min_{m\in[P]}(g_{I_j}(\tau_j^m) + c_{I_j}(\tau_j^m))$
    Increment $k_j \leftarrow k_j + 1$
  end for
  for each instance I_j with zero k_j or $k_j > K$ **do**
    Generate a new instance to replace I_j
    Initialize $\lambda_j \leftarrow \lambda^{(0)}$ and $k_j \leftarrow 0$
  end for
end for
Output: θ

without conditional dependence [35]. While computationally efficient, L2P methods often suffer from limited expressiveness. To address this issue, recent research has introduced diffusion models to enhance the L2P paradigm by leveraging their ability to generate multimodal distributions of optimal solutions [59, 43]. (3) **Learn-to-Search (L2S) methods** adopt the iterative framework of traditional search heuristics. During the search process, L2S methods usually leverage a RL policy to control or select search operators [49, 46], thereby guiding the search directions towards near-optimal solutions.

Recent advances in neural VRP. Recent advancements in neural methods for solving VRPs focus on improving scalability and robustness through innovative architectures and learning strategies. For example, the large-scale performance is improved by employing divide-and-conquer strategies [23, 69], leveraging heavy decoder architectures [47], incorporating distance-related bias [73], and exploiting local transferability [24, 22]; the robustness against distribution shifts is improved by distributional robust optimization [33], multi-distribution knowledge distillation [8], meta learning [74] and ensemble learning [34]. Furthermore, it is observed that the performance of neural solvers can be enhanced by utilizing a population of complementary models [27, 76, 25]. Moveover, Liu et al. [45] proposed to develop a foundation model for a class of VRP variants, leveraging the shared problem structure to achiece better performance. Building on this, Zhou et al. [75] further improved model capability by introducing the mixture-of-experts structure. Besides these efforts, this paper focuses on complex constrained VRPs, which are common in real-world applications [12, 26] but have not received much attention in the research community. Only a few works [60, 15, 9] try to address it through feature enhancement or Lagrange multiplier method. In this context, we introduce a novel instance-level adpative framework for Lagrangian-based neural methods, reducing the infeasiblity rate significantly.

Learning for constrained optimization in other domains. Most neural solvers for constrained optimization problems rely on expert-designed rules to prevent constraint violations during the decoding process [3, 59, 36]. For instance, Ahn et al. [3] proposed a clean-up phase that rolls back invalid actions, thereby enforcing feasibility through a hard constraint mechanism. However, these expert-driven rules often fail in complex scenarios, such as the constrained VRPs studied in this work, as well as in many continuous optimization problems. To address constraint violations in such cases,

optimization techniques based on penalty functions and Lagrange multipliers have been integrated into unsupervised learning [2] and self-supervised learning pipelines [53]. However, a common limitation arises when a single multiplier or penalty factor is applied across diverse problem instances with varying degrees of constraint violations. This challenge, though important, has been largely overlooked in prior studies [2, 65, 54]. Only a related study on continuous optimization [53] noticed this issue and introduced a primal-dual network to predict instance-specific dual variables during training. While this approach improves upon the single multiplier method, it remains constrained by the computational overhead associated with alternating primal-dual ascent and the limited accuracy of dual variable predictions.

D Instance Generation

In our experiments, we consider two categories of problem, TSPTW and TSPDL. Following prior works [40], we randomly sample coordinates (x_i, y_i) for each node i (including the depot) from a uniform distribution $U(0, 1)$ within a square. For generating the time windows and draft limits, we utilize the code of Bi et al. [9] and adopt the **hard** settings, which are sufficiently challenging to examine state-of-the-art neural and OR solvers. The generation process of time windows and draft limits is detailed as follows.

Time windows. After generating the node coordinates, the pairwise travel times are calculated based on the Euclidean distance between any two nodes. For the generation of time windows, we adopt the configuration of a widely recognized benchmark [17] in our experiments. Specifically, the process begins with the construction of a random tour τ (i.e., a random permutation of the nodes). Subsequently, the time window $[l_i, u_i]$ for each node i is iteratively generated, where the lower bound l_i and upper bound u_i are uniformly sampled from a range determined by the cumulative travel distance ϕ_i of the partial solution up to node i and the maximum window size 2η. More formally, $l_i \sim U[\phi_i - \eta, \phi_i]$ and $u_i \sim U[\phi_i, \phi_i + \eta]$. This procedure guarantees the existence of at least one feasible solution for each instance, and the tight coupling between the time windows and the randomized tours introduces significant complexity to the problem, thereby increasing the computational difficulty of satisfying constraints. In this paper, the maximum window size η is set to 50, and we employ a scale factor $\rho = 100$ to normalize the node coordinates and time windows according to [9].

Draft limits. In the context of TSPDL, each node is associated with a demand value and a maximum draft limit, which is designed to avoid overloaded ships entering these ports (i.e., nodes). From an initial feasible setting, the draft limit of each node is set to the summarized demands of other nodes, thereby ensuring that any node demand can not exceed its own draft limit. Subsequently, a fraction parameter, denoted as $p\%$, is introduced to adjust the draft limits of non-depot nodes. Specifically, $p\%$ of the non-depot nodes are randomly selected, and each of them is assigned a draft limit drawn as a random integer from the range $[\delta_i, \sum_{i=1}^{n} \delta_i]$, where δ_i is the demand of the i-th node. Finally, a feasibility validation is conducted (e.g., utilizing bin-counting constraints) to ensure that the assigned draft limits do not lead to instances without feasible solutions. In our experiment, the node demands are set to 1 and the fraction parameter $p\%$ is set to 90%.

E Implementation Details

E.1 Training Details

The training procedure of our ICO method contains two stages: a pre-training stage and a fine-tuning stage. The pre-training stage involves a total of $10,000$ epochs, while the fine-tuning stage comprises $1,000$ epochs. Each training epoch processes $10,000$ synthetic problem instances. For both stages, we select the model checkpoint that achieves the best inference performance on a validation dataset as the final model. It is worth noting that the training process of our ICO method includes $1,000$ more epochs compared to the training process of POMO+PIP. To ensure a fair comparison, we extend the training of the provided POMO+PIP checkpoints by an additional $1,000$ epochs.

The fine-tuning stage involves the iterative updating of λ values. In this process, the initial values $\lambda^{(0)}$ is uniformly set to 0.1 for all problem instances. If the policy fails to find feasible solutions

on a specific instance, the λ value corresponding to this instance is updated based on the constraint violation, where the learning rate of λ is set to 0.5 for TSPTW and 0.2 for TSPDL, since the scales of constraint violations on TSPTW and TSPDL are different. These hyperparameters in updating λ are aligned with the corresponding hyperparameters in the inference stage, narrowing the gap of training and inference. To improve computational efficiency and mitigate the risk of overfocusing on challenging instances, the number of iterations is limited to a maximum of 4, and the ratio of infeasible instances within a batch must not exceed 25%. During the fine-tuning on TSPDL50, we observe that the fine-tuned policy tends to overemphasize the constraints, resulting in a near zero infeasibility rate but a significant deterioration in objective values. To mitigate this issue, we adjust the learning rate of fine-tuning process on TSPDL50 to 1×10^{-6}, while learning rates of other training process remain the default setting (i.e., 1×10^{-4}).

E.2 Inference Details

The instance-specific λ values are iteratively updated based on constraint violations during the inference stage. In this process, the λ values are initialized as 0.1 for all instances except instances of TSPDL100, since it is observed that the conditioned policy fails to obtain feasible solutions for most instances of TSPDL100 when using $\lambda = 0.1$. Consequently, the intial λ value for TSPDL100 is increased to 0.5. During the updating process of λ, the learning rate is configured as 0.5 for TSPTW and 0.2 for TSPDL. These different learning rates are to accommodate the different scales of constraint violations on these two problem types. In the comparison experiments, the number of iterations for updating λ is set to 16.

E.3 Experimental Settings

Metrics. Four metrics are applied: Infeasibility rate, average optimality gap, normalized Hyper-Volume (HV) and runtime. The instance-level infeasibility rate measures the proportion of instances where the solver fails to find any feasible solution. These metrics are calculated on a test dataset containing 10,000 instances. To compute the optimality gap, we use the solutions obtained by LKH3 through full-time search as reference solutions. Unlike some prior works that compute the optimality gap directly from the average objective [40], we calculate the optimality gap on an instance-by-instance basis and then average these values. It is important to note that the calculation of objective values and optimality gaps only includes instances with feasible solutions. Therefore, the average objective value may not serve as a fully reliable metric for performance comparison, as the sets of instances with feasible solutions can vary across different methods. To measure the comprehensive performance of both solution quality and feasibility, we further compute the normalized HV based on the infeasibility rate and average optimality gap. The reference point for computing HV is set to $(100\%, 5\%)$ for TSPTW and $(10\%, 20\%)$ for TSPDL, where the first number represent the infeasibility rate and the other denotes the average gap. To evaluate the computational efficiency, we compare the total runtime of solving 10,000 instances with batch parallelism on a single GPU (NVIDIA RTX 4090 Ti). For OR solvers like LKH3 and OR-Tools, we record the runtime of parallel computation on 16 CPU cores.

Evalution configurations of baselines. To align the runtime consumption, POMO+PIP employs $\times 28$ sampling for intances with $n = 50$ and $\times 20$ sampling for instances with $n = 100$, where AM+PIP adopts $\times 200$ sampling for both $n = 50$ and $n = 100$ instances. These different sampling configurations are to align with the additional runtime caused by the computation of λ-conditioned embeddings in our ICO method. The evaluation batch sizes for both POMO-PIP and our ICO method are set to 2,500 for instances with $n = 50$ and 1000 for instances with $n = 100$.

F Additional Results

F.1 Extension to more problem variants.

The idea of instance-level adaptive dual variables is not specially designed for TSPTW and TSPDL; rather, it can be extended to other domains that simultaneously require constraint handling and cross-instance (or cross-environment) generalization of the RL policy, with domain-specific adaptations. To demonstrate generality, we extend our method to more VRP variants. After summarizing the

hard-constrained VRPs addressed in prior works [15, 21, 9, 60, 14], we find that CVRPTW is the only problem not addressed in our experiments. While the decision space of CVRPTW appears more complex, it is, in fact, easier to satisfy its constraints compared to TSPTW and TSPDL. This is because its time window constraints can be easily satisfied by a shortcut: Add more vehicles.

To construct a challenging benchmark, we propose to **set a maximum limit on the number of vehicles**, which also aligns more closely with real-world applications. We conduct new experiments on CVRPTW50 with limited vehicles using JAMPR's time window generation code [21]. Since PIP has not been extended to this problem, we used POMO as the backbone to implement ICO. Experimental results in Table 3 show that our ICO significantly outperforms the POMO baseline, especially in infeasibility rate.

Table 3: Experimental results on new problem variants: CVRPTW50 and CVRPTW50 with limited vehicles. To compute HV, we use reference point (1%, 15) for CVRPTW50 and (10%, 15) for CVRPTW50 with limited vehicles. The best results are highlighted in **bold**.

	CVRPTW50				**CVRPTW50 with limited vehicles**			
Method	Inf. rate	Obj.	HV	Time	Inf. rate	Obj.	HV	Time
POMO ($\lambda = 0.5$)	0.69%	**13.99**	0.021	39s	4.35%	**14.05**	0.036	38s
POMO ($\lambda = 1.0$)	0.25%	14.22	0.039	40s	3.26%	14.28	0.033	38s
POMO ($\lambda = 2.0$)	0.31%	14.49	0.023	39s	2.51%	14.51	0.025	38s
ICO	**0.10%**	14.00	**0.060**	40s	**1.16%**	14.09	**0.054**	40s

F.2 Analysis of Different Update Rules for λ

Proportional-Integral-Derivative (PID) control for updating λ. From the perspective of control theory, the subgradient ascent process of λ behaves as *integral* control, while Stooke et al. [58] proposed to further incorporate *proportional* and *derivative* control into the update rule, avoiding oscillations encountered by the integral-only controller. The proportional control is to hasten the constraint satisfaction in response to the immediate constraint violation. The derivative control prevents the oscillations by monitoring the variation tendency of constraint violations. By adding the terms of proportional, integral and derivative control, the update rule of PID control is expressed as:

$$\begin{aligned}
\Delta_t &= g_I(\tau_t), \\
I_t &= I_{t-1} + g_I(\tau_t), \\
\delta_t &= \max\{g_I(\tau_t) - g_I(\tau_{t-1}), 0\}, \\
\lambda_t &= K_P \cdot \Delta_t + K_I \cdot I_t + K_D \cdot \delta_t,
\end{aligned}$$

where Δ_t represents the proportional term of time step t, I_t denotes the t-th step integral term that accumulates the constraint violations of previous steps, δ_t computes the derivative term of the constraint violation, and K_P, K_I, K_D are tuning parameters that measure the weights of three terms. Intuitively, this PID method provides a richer set of controllers than subgradient ascent, but it also introduces more hyperparameters that require manual tuning. In our experiments, K_P is set to 0.1 and K_D is set to 1.0 on both problem types, and K_I is set to 0.5 on TSPTW and 0.01 on TSPDL.

In Table 4, we compare the performance of different update rules of λ in inference stage: fixed λ values ($\lambda \in \{0.5, 1.0, 2.0\}$), randomly sampled λ values, the subgradient ascent method and the PID control method [58]. For the random sampling strategy, λ values are drawn randomly from the uniform distribution $U(0.1, 2.0)$ at each iteration.

The results in the last three rows indicate that both the subgradient ascent method and the PID control method generally outperform the random sampling strategy, with particularly improvements in reducing the infeasibility rate. As evidenced in the first three rows, employing fixed λ values leads to significantly inferior performance compared to the adaptive variation of λ, underscoring the critical importance of dynamically adjusting λ for each instance. It is worth noting that the random sampling approach also demonstrates competitive performance, indicating that simply varying the λ values randomly for each instance has a high probability of identifying effective λ values. By comparing

the results of the last two rows, it is observed that the PID control method does not achieve superior performance as expected, which can be attributed to two factors: (1) the hyperparameters of PID are challenging to tune; (2) the subgradient ascent method is already involved in the fine-tuning process, while the PID control is not integrated into the training, limiting its effectiveness.

Table 4: Additional results of different update rules of λ on TSPTW and TSPDL. The best results are highlighted in **bold**.

Methods	TSPTW ($n = 50$)		TSPTW ($n = 100$)		TSPDL ($n = 50$)		TSPDL ($n = 100$)	
	Inf. rate	Avg. Gap	Inf. rate	Avg. Gap	Inf. rate	Avg. Gap	Inf. rate	Avg. Gap
ICO ($\lambda = 0.5$)	1.43%	0.19%	4.34%	0.26%	2.63%	2.50%	42.14%	13.16%
ICO ($\lambda = 1.0$)	1.52%	0.23%	4.03%	0.36%	0.23%	2.77%	2.01%	10.79%
ICO ($\lambda = 2.0$)	1.55%	0.24%	4.27%	0.38%	0.07%	3.15%	0.38%	11.62%
ICO (random)	0.55%	0.07%	2.40%	0.14%	0.12%	**2.28%**	0.40%	10.73%
ICO (subgradient)	**0.51%**	0.07%	**1.33%**	0.14%	**0.01%**	2.32%	0.91%	**9.22%**
ICO (PID control)	0.55%	0.07%	1.39%	0.14%	0.05%	2.36%	**0.26%**	9.25%

F.3 Analysis of training strategies

Figure 5 illustrates the performance of POMO+PIP (with $\lambda = 1$), the pre-trained policy, and the fine-tuned policy. The comparison between the pre-trained and fine-tuned policies reveals that the fine-tuning process leads to a substantial reduction in both infeasibility rate and average gap, except the average gap on TSPDL50. Notably, even the pre-trained policy alone surpasses the single-λ POMO+PIP, further highlighting the advantages of the proposed approach.

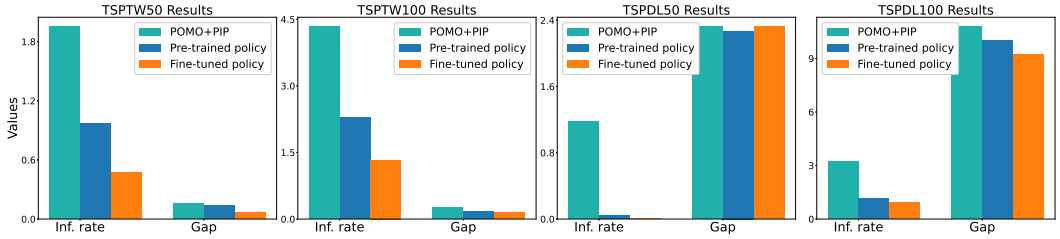


Figure 5: Comparison of the pre-trained policy and fine-tuned policy.

F.4 Analysis of Network Architectures

The λ-conditioned policy network takes λ as the condition varibable and adjust the constraint awareness according to the varying value of λ. Among existing network architectures in other domains [66, 44], there are two alternative approaches to implement the conditioned policy: (1) condition λ in the initial embeddings; (2) condition λ in the decoder's context. The second approach, referred to as the λ-conditioned context method, is detailed as follows.

λ-conditioned context. Building upon the POMO model [41], the conditioned context method integrates a linear embedding of λ into the decoder's *context* embedding, formulated as $\boldsymbol{q} = W^{\lambda}\lambda + W^{q}[\boldsymbol{h}^{c}, t^{c}]$. Here, $W^{\lambda} \in \mathbb{R}^{d\times 1}$ and $W^{q} \in \mathbb{R}^{d\times d}$ are trainable parameters, and $[\boldsymbol{h}^{c}, t^{c}]$ denotes the concatenation of the current node embedding $\boldsymbol{h}^{c}$ and the current time t^{c}, together forming the *context* used for selecting candidate nodes. The resulting output, $\boldsymbol{q}$, functions as the query input for the subsequent multi-head attention layer in the decoder. This conditioned context approach incorporates the information of λ into the core component of the decoder, enabling an efficient adjustment of the policy's behavior.

In Table 5, we compare the performance of the network with λ-conditioned context and network with λ-conditioned embeddings on TSPTW100 and TSPDL100. Here we report the results of the pre-trained policies. The experimental results demonstrate that the λ-conditioned embedding method achieves significantly superior performance in both infeasibility rate and average optimality gap. This performance advantage can be attributed to the fact that the λ-conditioned embedding utilizes the

Table 5: Additional results of different network architectures on TSPTW and TSPDL. The best results are highlighted in **bold**.

Methods	TSPTW ($n = 100$)		TSPDL ($n = 100$)	
	Inf. rate	Avg. Gap	Inf. rate	Avg. Gap
Network with λ-conditioned context	2.83%	0.30%	2.31%	13.34%
Network with λ-conditioned embeddings	**2.28%**	**0.17%**	**1.14%**	**10.01%**

full capacity of the entire network to process λ-related information, while the conditioned context approach restricts the λ-related information to the decoder, thereby limiting its effectiveness.

F.5 Analysis of the distribution $D(\lambda)$ in training stage

In the pre-training stage of the conditioned policy, random values of λ are sampled from a pre-defined distribution $D(\lambda)$ for training. Empirically, the distribution $D(\lambda)$ has a non-negligible influence on the performance of the pre-trained policy. A natural and straightforward option for $D(\lambda)$ is the uniform distribution within an appropriate range. However, as shown in Figure 6, the trained policy just silghtly violates constraints on the majority of instances, where only a small subset of instances in the long tail experience significant constraint violations. Therefore, we adopt a triangular distribution $T(0.1, 0.5, 2.0)$, which biases the sampling towards smaller λ values, thereby prioritizing the optimization of instances with low constraint violations. Figure 7 compares the performance of the policy trained with a uniform distribution $U(0.1, 2.0)$ and the policy trained with a triangular distribution $T(0.1, 0.5, 2.0)$ on the TSPTW50 dataset. The results demonstrate that the triangular distribution leads to superior overall performance as expected.

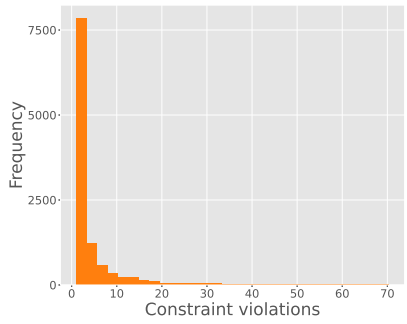


Figure 6: Histogram of constraint violation statistics on the validation dataset.

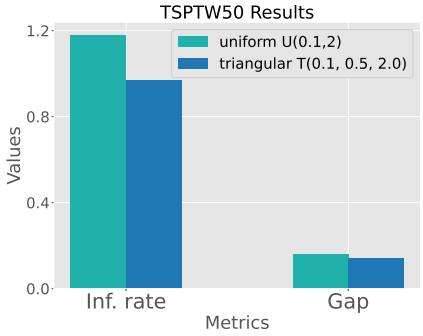


Figure 7: Performance of using two different $D(\lambda)$ configurations during the pre-training stage.

F.6 Sensitivity of λ-related hyperparameters in inference stage

Since the optimization landscape for λ is typically non-convex due to the hardness of combinatorial optimization, the initial value and learning rate of λ are both important for the optimization performance. Here we conduct a sensitivity analysis of λ from these two perspectives, including the initial value of λ (denoted as λ_0) and the learning rate for updating λ (denoted as α). During the inference stage, we evaluated performance across $\lambda_0 \in \{0.1, 0.15, 0.20, 0.5, 1.0\}$ and $\alpha \in \{0.1, 0.2, 0.5, 0.7, 1.0\}$ on TSPTW50 and TSPDL50. Each hyperparameter was varied while keeping the other fixed at its default value. Results in Table 6 and 7 show that:

- In 16 out of 18 settings, our ICO method surpasses the best-performing PIP model in hypervolume (HV), showing its robustness.
- Although the performance variance (shown in the last row) is relatively small, it is not negligible. This underscores the importance of carefully tuning λ-related hyperparameters to achieve optimal performance.

Interestingly, some settings (e.g., $\alpha = 0.7$ for TSPTW50) slightly outperform the default, suggesting that advanced hyperparameter optimization techniques could further enhance performance.

Table 6: Sensitivity analysis in inference stage on TSPTW50. λ_0 denotes the initial value of λ and α represents the learning rate of λ.

	Inf. rate	Gap	HV	Better HV than PIP
PIP with the best HV	1.95%	0.08%	0.965	-
$\lambda_0 = 0.1$ (default)	0.50%	0.07%	0.981	Yes
$\lambda_0 = 0.15$	0.47%	0.08%	0.979	Yes
$\lambda_0 = 0.2$	0.48%	0.08%	0.979	Yes
$\lambda_0 = 0.5$	0.84%	0.19%	0.954	No
$\lambda_0 = 1.0$	0.97%	0.23%	0.945	No
$\alpha = 0.1$	0.59%	0.07%	0.980	Yes
$\alpha = 0.2$	0.49%	0.07%	0.981	Yes
$\alpha = 0.5$ (default)	0.50%	0.07%	0.981	Yes
$\alpha = 0.7$	0.48%	0.07%	0.981	Yes
$\alpha = 1.0$	0.55%	0.07%	0.981	Yes
Avg $\pm$ Std	0.59% $\pm$ 0.17%	0.10% $\pm$ 0.06%	0.974 $\pm$ 0.0133	-

Table 7: Sensitivity analysis in inference stage on TSPDL50. λ_0 denotes the initial value of λ and α represents the learning rate of λ.

	Inf. rate	Gap	HV	Better HV than PIP
PIP with the best HV	0.12%	2.89%	0.845	-
$\lambda_0 = 0.1$ (default)	0.01%	2.32%	0.883	Yes
$\lambda_0 = 0.15$	0.01%	2.33%	0.883	Yes
$\lambda_0 = 0.2$	0.01%	2.33%	0.883	Yes
$\lambda_0 = 0.5$	0.01%	2.51%	0.874	Yes
$\lambda_0 = 1.0$	0.01%	2.79%	0.859	Yes
$\alpha = 0.1$	0.06%	2.23%	0.883	Yes
$\alpha = 0.2$ (default)	0.01%	2.32%	0.883	Yes
$\alpha = 0.5$	0.00%	2.47%	0.877	Yes
$\alpha = 0.7$	0.00%	2.58%	0.871	Yes
$\alpha = 1.0$	0.00%	2.80%	0.860	Yes
Avg $\pm$ Std	0.01% $\pm$ 0.02%	2.47% $\pm$ 0.20%	0.876 $\pm$ 0.009	-

G Licenses

Table 8: List of licenses for the codes and datasets we used in this work.

Resource	Type	Link	License
OR-Tools [20]	Code	https://github.com/google/or-tools	Apache License 2.0
LKH3 [30]	Code	http://webhotel4.ruc.dk/ keld/research/LKH-3/	Available for academic research use
AM [40]	Code	https://github.com/wouterkool/attention-learn-to-route	MIT License
POMO [41]	Code	https://github.com/yd-kwon/POMO	MIT License
EAS [32]	Code	https://github.com/ahottung/EAS	Available online
JAMPR [21]	Code	https://github.com/jokofa/JAMPR	MIT License
PIP [9]	Code	https://github.com/jieyibi/PIP-constraint	MIT License

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: Our contributions are clearly presented in the abstract and introduction.

   Guidelines:

   - The answer NA means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A No or NA answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: We discussed the primary limitation of this work in the conclusion section.

   Guidelines:

   - The answer NA means that the paper has no limitation while the answer No means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [NA]

Justification: This paper is an empirical study.

Guidelines:

- The answer NA means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: We provided all the key information to reproduce our results.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - If the paper includes experiments, a No answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
     (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
     (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
     (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
     (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

Answer: [Yes]

Justification: The code and README files will be provided in the supplemental materials.

Guidelines:

- The answer NA means that paper does not include experiments requiring code.
- Please see the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- While we encourage the release of code and data, we understand that this might not be possible, so "No" is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
- The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
- The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer, etc.) necessary to understand the results?

Answer: [Yes]

Justification: Important settings and details are provided in the experiment section. Other necessary details are provided in the appendix.

Guidelines:

- The answer NA means that the paper does not include experiments.
- The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
- The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

Answer: [No]

Justification: Training consumption of our neural models is relatively high.

Guidelines:

- The answer NA means that the paper does not include experiments.
- The authors should answer "Yes" if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).

- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g. negative error rates).
- If error bars are reported in tables or plots, The authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: Provided in the appendix.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: No ethics issues.

   Guidelines:

   - The answer NA means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer No, they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [No]

    Justification: This paper studies a general method for optimizing classical problems, which is not directly related to any societal impact.

    Guidelines:

    - The answer NA means that there is no societal impact of the work performed.
    - If the authors answer NA or No, they should explain why their work has no societal impact or why the paper does not address societal impact.
    - Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.

- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pretrained language models, image generators, or scraped datasets)?

    Answer: [NA]

    Justification: The studied classical optimization problems have no risk to affect safety.

    Guidelines:

    - The answer NA means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: We properly cited the used papers and codes. Their licenses are included in the appendix.

    Guidelines:

    - The answer NA means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
    - If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
    - For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.

- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [Yes]

    Justification: Our code and models are well documented and anomymized.

    Guidelines:

    - The answer NA means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [NA]

    Justification: Datasets are synthetic.

    Guidelines:

    - The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [NA]

    Justification: Not applicable.

    Guidelines:

    - The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
    - We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
    - For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does not impact the core methodology, scientific rigorousness, or originality of the research, declaration is not required.

Answer: [NA]

Justification: We only use LLMs for writing.

Guidelines:

- The answer NA means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
- Please refer to our LLM policy (`https://neurips.cc/Conferences/2025/LLM`) for what should or should not be described.