Learning to Handle Constraints in Routing Problems via a Construct-and-Refine Framework

Jieyi Bi[1], **Zhiguang Cao**[2], **Jianan Zhou**[1],
Wen Song[3], **Yaoxin Wu**[4], **Jie Zhang**[1], **Yining Ma**[5*], **Cathy Wu**[5]

[1]Nanyang Technological University
[2]Singapore Management University
[3]Shandong University
[4]Eindhoven University of Technology
[5]Massachusetts Institute of Technology

Abstract

Neural solvers have achieved impressive progress on simple routing problems via data-driven training, but often struggle with complex constraints. We rethink the popular single-paradigm neural solvers and identify paradigm-inherent limitations: construction solvers suffer from inflexible stepwise feasibility, and improvement solvers easily get stuck in infeasible searches with long runtimes. However, these paradigms are naturally complementary: construction efficiently provides strong initial solutions that help improvement rapidly reach feasible, high-quality solutions. Motivated by this, we propose Construct-and-Refine (CaR), the first generic neural framework for efficient constraint handling, compatible with existing construction and improvement solvers. To promote synergistic paradigm integration, we introduce a joint training framework with bespoke losses to generate diverse, high-quality, (near)-feasible solutions that are refined by a light improvement process (e.g., only 10 steps down from 5k). We also present the first study of a shared encoder for cross-paradigm representation learning in handling complex constraints. Extensive experiments on hard-constrained TSPTW and CVRPBLTW demonstrate that CaR achieves superior feasibility, solution quality, and efficiency compared to both traditional and neural state-of-the-art solvers.

1 Introduction

Vehicle routing problems (VRPs) often involve complex real-world constraints [1, 2]. For decades, the Operations Research (OR) community has put remarkable efforts into designing hand-crafted heuristics to handle these constraints and approximate near-optimal solutions, leading to powerful solvers such as OR-Tools [3] and LKH3 [4]. Recently, neural combinatorial optimization (NCO) solvers [5–7] have enriched the landscape for solving VRPs, which learn deep models to solve VRPs in a data-driven manner, thereby reducing dependence on domain knowledge. Benefiting from GPU parallelism, they are often fast while maintaining competitive solution quality. In general, existing neural solvers fall into two paradigms: 1) *Construction solvers* (e.g. [8, 9]), which construct solutions node by node from scratch, excelling in efficiency but prone to local optima; 2) *Improvement solvers* (e.g. [10, 11]), which iteratively refine a complete solution, exploring a broader search space but facing longer runtime. However, when they come to handling complex VRP constraints, both paradigms face critical inherent limitations that significantly hinder both feasibility and optimality.

*Yining Ma is the corresponding author (yiningma@mit.edu).

39th Conference on Neural Information Processing Systems (NeurIPS 2025) Workshop: DiffCoALG: Differentiable Learning of Combinatorial Algorithms.

Construction solvers are inherently built upon a *stepwise feasibility satisfaction* process, where constraints must be enforced at each node selection step. While feasibility masking addresses this by excluding the invalid candidates and works well for simple cases like the Capacitated VRP (CVRP), it fails to handle more complex ones such as the Traveling Salesman Problem with Time Windows (TSPTW), where computing accurate masks itself is NP-hard thus intractable [12]. Recent attempts, such as lookahead strategies [13] and learnable approximated masking [12], alleviate this but come with high computational costs and still yield large infeasibility rates. Moreover, as discussed in Section 4, even for complex VRPs where feasibility masking is tractable, such as CVRP with backhaul, duration limit, and time window constraints (CVRPBLTW), enforcing multiple constraints through such stepwise masking can overly restrict learning, thereby degrading model performance and making it more prone to local optima. Hence, achieving stepwise feasibility satisfaction while maintaining optimality remains a critical challenge in current construction solvers.

On the other hand, the performance of improvement solvers on complex-constrained VRPs remains largely unexplored. Akin to the construction methods, they struggle with accurate feasibility masking for complex local search operators (e.g., k-opt). While prior work [11] showed that learning-guided infeasible region exploration improves performance in simpler constrained CVRP, it lacks feasibility guarantees and remains potentially time-consuming for complex VRPs. In more complex VRPs like CVRPBLTW, these improvement solvers face prolonged searches in infeasible regions (see Table 1), further amplifying the inherent inefficiencies in exploring complex search spaces.

As a result, both solver paradigms struggle with constraint handling when used alone, often exhibiting inefficiencies, poor adaptability to complex constraints, or high infeasibility and suboptimality. However, they offer complementary strengths: improvement solvers excel at exploring local neighborhoods to repair infeasible or suboptimal solutions, while construction solvers provide strong initializations that allow improvement to rapidly reach feasible, high-quality solutions. This raises our research question: *can we unify the strengths of the two paradigms to handle constraints without sacrificing efficiency?*

Motivated by this, we present **Construct-and-Refine (CaR)**, the first generic neural framework that promotes efficient constraint handling through paradigm integration. Unlike existing methods that are often paradigm-limited, inefficient or lack generality across VRPs with varying constraint complexities, CaR unites the efficiency of construction with the flexibility of improvement, enabling efficient constraint handling for general complex VRPs where feasibility masking is either intractable (TSPTW) or overly restrictive (CVRPBLTW). To enhance the synergy between paradigms, CaR adopts an end-to-end joint training framework with bespoke loss functions that enable the construction module to generate diverse, high-quality, (near)-feasible initial solutions, which are then passed to a light refinement module, reducing the number of improvement steps required, e.g., from 5k to 10. Moreover, to further balance computational cost and enhance synergy, we take the first step toward exploring how a unified encoder across construction and refinement modules can enable more effective cross-paradigm representation learning. Lastly, we show that CaR is a generic framework compatible with existing state-of-the-art construction and improvement methods for efficient constraint handling.

Our contributions are: 1) We introduce Construct-and-Refine (CaR), the first neural framework that enables efficient constraint handling through paradigm integration, excelling in solving complex VRPs where existing neural solvers fail to solve effectively; 2) We explore cross-paradigm representation learning by unifying encoders across the two paradigms, yielding notable benefits for hard-constrained scenarios; 3) We conduct extensive experiments on representative hard-constrained TSPTW and CVRPBLTW, where masking is intractable or overly restrictive. Notably, CaR is the first generic neural Construct-and-Refine framework for efficient constraint handling in VRPs, achieving substantial reduction in infeasibility, especially where construction alone fails, while simultaneously improving solution quality and runtime efficiency via light refinement.

2 Preliminaries

VRP variants. VRP is defined over a directed graph $\mathcal{G} = \{\mathcal{V}, \mathcal{E}\}$, where $\mathcal{V}$ contains n customer nodes $\{v_1, v_2, \ldots, v_n\}$ (for both TSP and VRP variants) and a depot $\{v_0\}$ (for VRP only), and $e(v_i, \rightarrow v_j)$ (or e_{ij}) $\in \mathcal{E}$ represents an edge from node v_i to v_j $(i \neq j)$ weighted by the 2D Euclidean distances. It aims to minimize the total cost of a solution while satisfying variant-specific constraints. This paper focuses on two representative complex VRPs: TSPTW with time window constraints,

and CVRPBLTW with backhaul, duration limit, and time window constraints. See Appendix B for more details on the constraints. Neural VRP solvers usually handle these constraints by feasibility masking, which is tractable and effective in simpler cases like CVRP. However, they struggle with harder variants, e.g., CVRPBLTW with multiple constraints, or TSPTW, where masking is NP-hard since computing accurate masks requires evaluating future time-related feasibility for all possible actions. We further discuss the limitations of the existing feasibility masking in Appendix F.1.

MDP formulations. We model construction and improvement as Constrained Markov Decision Processes (CMDPs), defined by the tuple $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \mathcal{C})$, where $\mathcal{S}$ is the state space, $\mathcal{A}$ the action space, $\mathcal{P}$ the transition function, $\mathcal{R}$ the reward, and $\mathcal{C}$ the constraint function over m constraints. The objective is to learn a policy π_θ maximizing expected reward while satisfying constraints:

$$\max_\theta \mathcal{J}(\pi_\theta) = \mathbb{E}_{\tau\sim\pi_\theta}\left[\mathcal{R}\left(\tau|\mathcal{G}\right)\right], \text{s.t. } \pi_\theta \in \Pi_F,\ \Pi_F \quad = \{\pi \in \Pi \,|\mathcal{J}_\mathcal{C}(\pi_\theta) = \mathbf{0}^m\}, \tag{1}$$

where Π_F denotes the feasible policy set. *Construction* solvers build solutions sequentially, with state s_t encoding partial routes, vehicle status, and unvisited nodes; action a_t selects the next node; and the reward is the negative tour cost, $\mathcal{R}(\tau \mid \mathcal{G}) = -C(\tau)$. In contrast, *improvement* solvers refine complete solutions, where s_t includes the current and best-so-far solutions; actions apply local operators (e.g., k-opt, remove-and-reinsert); and the reward is $\mathcal{R}_t = \min[C(\tau_t^*) - C(\tau_{t+1}), 0]$ [14].

Relaxation of CMDP. Following [12], we adopt Lagrangian relaxation to facilitate training by penalizing constraint violations. The objective in Eq. (1) is augmented with a cost term:

$$C(\tau) = \sum_{e_{ij}\in\tau}\left[C_L\left(e_{ij}\right) + \sum_{\eta=1}^{m} C_V^\eta\left(e_{ij}\right)\right], \tag{2}$$

where C_L denotes the objective cost (e.g., tour length), and C_V captures violations of m constraints. For instance, if a vehicle arrives at node v_j later than the time window upper bound u_j, the violation cost is $C_V(e_{ij}) = t_j - u_j$. C_V also incorporates the count of violated nodes.

3 Methodology

As discussed above, single-paradigm methods have inherent limitations for constraint handling but can complement each other (see Appendix C for details). On one hand, improvement can rapidly repair construction's infeasible or suboptimal solutions, due to its strength in exploring local neighborhoods; on the other hand, construction is highly efficient and can provide strong initializations that help improvement rapidly reach feasible, high-quality solutions, complementing improvement's limitation in inefficiency. To further promote the synergic effects, we present our unified CaR framework for complex VRPs where masking is *intractable* (TSPTW) or *restrictive* (CVRPBLTW).

3.1 Joint training framework of CaR

CaR aims to transcend simple paradigm merging to handle constraints more efficiently than either paradigm alone. We develop a unified training framework to jointly optimize the two policies in CaR.

Relaxation of CMDP in CaR. Building on the success of relaxed CMDP formulation for construction [12], we extend it to refinement to better balance objective and feasibility. Unlike prior infinite-horizon approaches [11, 15], we adopt a fixed rollout limit T_R, treating steps equally to align with CaR's lightweight and efficient refinement design.

Collaborative training framework. To integrate construction and refinement effectively, we design a joint training framework that optimizes both processes simultaneously in each gradient step, allowing them to co-evolve. As illustrated in Figure 1, for each batch of training instances $\mathcal{G}$, the construction module first generates a small set of diverse, high-quality initial solutions in parallel. These solutions are then refined by a lightweight neural improvement process within T_R steps (i.e., $T_R = 10$ significantly fewer than the thousands of iterations in conventional improvement methods), enabling rapid enhancement of high-potential candidates. The refined outputs then supervise construction, promoting collaborative correction of infeasibility and sub-optimality.

Construction module loss. The policy π_θ^{C} is trained via REINFORCE [16], using the loss function:

$$\mathcal{L}_{\mathrm{RL}}^{\mathrm{C}} = \frac{1}{S}\sum_{i=1}^{S}\left[\left(\mathcal{R}(\tau_i) - \frac{1}{S}\sum_{j=1}^{S}\mathcal{R}(\tau_j)\right)\log \pi_\theta^{\mathrm{C}}(\tau_i)\right], \tag{3}$$

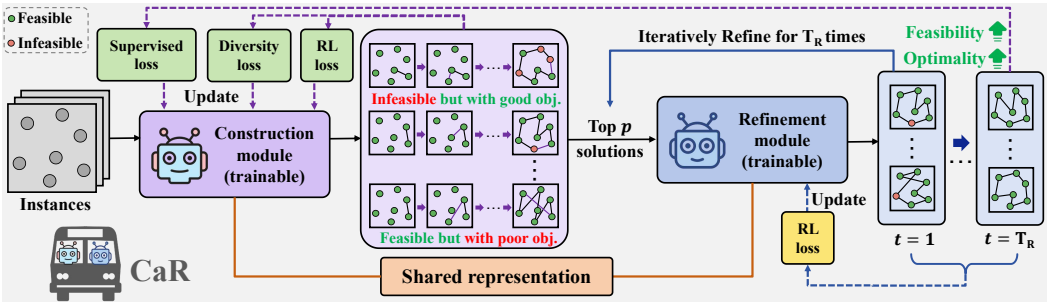

Figure 1: Overall framework of CaR, where the construction module provides *diverse* near-feasible and/or high-quality solutions for the refinement module to generate better solutions.

where the solution probability is factorized as $\pi_\theta^{\mathrm{C}}(\tau) = \prod_{t=1}^{|\tau|} \pi_\theta^{\mathrm{C}}(e_t \mid \tau_{<t})$, with $\tau_{<t}$ denoting the partial solution prior to selecting edge e_t at step t. We employ a group baseline with diverse rollouts to reduce REINFORCE variance. For simpler variants like CVRP, S solutions are generated via POMO's multi-start strategy [9], while for time-constrained variants (e.g., TSPTW), we sample S solutions to avoid infeasibility (see Appendix F.2). To compensate for reduced diversity due to the removal of the multi-start mechanism and to enhance the diversity of initial constructed solutions for refinement, we introduce an auxiliary entropy-based diversity loss:

$$\mathcal{L}_{\mathrm{DIV}} = -\sum_{t=1}^{|\tau|} \pi_\theta^{\mathrm{C}}(e_t \mid \tau_{<t}) \log \pi_\theta^{\mathrm{C}}(e_t \mid \tau_{<t}), \tag{4}$$

which largely encourages policy exploration during RL training. To avoid inefficiency, we evaluate candidates using the cost in Eq. (2), and only feed the top p high-quality candidates to subsequent refinement. If the refinement module improves a constructed solution (indicated by $\mathbb{I} = 1$), the best-refined solution τ^* is used as a pseudo ground truth to supervise π_θ^{C}:

$$\mathcal{L}_{\mathrm{SL}} = -\mathbb{I} \cdot \sum_{t=1}^{|\tau^*|} \log \pi_\theta^{\mathrm{C}}(e_t^* \mid \tau_{<t}^*), \tag{5}$$

where $\mathbb{I}$ indicates whether such refinement led to improvement. The final construction loss integrates three components, i.e., $\mathcal{L}(\theta^{\mathrm{C}}) = \mathcal{L}_{\mathrm{RL}}^{\mathrm{C}} + \alpha_1 \mathcal{L}_{\mathrm{DIV}} + \alpha_2 \mathcal{L}_{\mathrm{SL}}$.

Refinement module loss. The refinement policy π_θ^{R} iteratively improves solutions over T_R steps, with probability of the refined solution at step t is factorized as $\pi_\theta^{\mathrm{R}}(\tau_t) = \prod_{\kappa=1}^{\mathbf{K}} \pi_\theta^{\mathrm{R}}(a_\kappa | a_{<\kappa}, \tau_{t-1})$, where $\mathbf{K}$ denotes the total number of sequential refinement moves/actions, with further details in Appendix D. The RL loss $\mathcal{L}_{\mathrm{RL}}^{\mathrm{R}}(t)$ for refinement is computed at each step t using the REINFORCE algorithm in Eq. (3), where S is replaced by p, since only p solutions are refined. The final refinement loss is defined as the average across all T_R steps: $\mathcal{L}(\theta^{\mathrm{R}}) = \frac{1}{T_R} \sum_{t=1}^{T_R} \mathcal{L}_{\mathrm{RL}}^{\mathrm{R}}(t)$, encouraging each refinement step to contribute meaningfully and improving overall refinement efficiency.

Joint training loss. The joint training loss combines the above two losses, i.e., $\mathcal{L}(\theta) = \mathcal{L}(\theta^{\mathrm{C}}) + \omega \mathcal{L}(\theta^{\mathrm{R}})$, where ω balances their scales. Such joint loss promotes information exchange between modules , enhancing synergy in collaboratively handling complex constraints.

3.2 Unified model architecture

To reduce overhead and enhance synergy, we take the first step in exploring shared encoders for more effective cross-paradigm representation learning for hard-constrained VRPs.

Encoder. Given an instance batch $\{\mathcal{G}_i\}_{i=1}^B$, both paradigms learns to obtain high-dimensional node embeddings h_i via encoders. For CVRPBLTW, each node v_i is represented by its coordinates, demand (i.e., linehaul or backhaul), time window, and duration limit, i.e., $f_i^{\mathrm{n}} = \{x_i, y_i, q_i, l_i, u_i, \ell\}$. Unlike construction, refinement also incorporates solution features by encoding the sequential structure via positional information. To support both paradigms, we use a shared 6-layer Transformer encoder [9] with multi-head attention. Positional encoding, required only in refinement, is injected using cyclic positional encoding (CPE) via the synthesis-attention (Syn-Att) mechanism [17]. Afterwards, a

multi-layer perceptron (MLP) fuses node-level attention scores a^{n} and solution-level scores a^{s} from positional embedding vectors through element-wise aggregation (see details in Figure 4).

Decoder. The decoder generates action probabilities from node representations, selecting the next node for construction or the modification for refinement. In CaR, we retain the original neural solver designs when applied to *one* construction and *one* improvement at a time. To validate generality, we experiment with two construction backbones, POMO [9] and PIP [12], and two refinement backbones, NeuOpt [11] and N2S [17]. To adapt improvement solvers for new variants (e.g., TSPTW, CVRPBLTW), we follow their original design and introduce variant-specific features, such as refinement history and node-level feasibility information (see Appendix D for details), to enhance constraint awareness. While we also explored a unified decoder, results in Figure 7 show degraded performance, suggesting that while a shared encoder benefits representation learning, separate decoders remain important for paradigm-specific optimization – an insight for future reference.

4 Experiments

We now evaluate our proposed Construct-and-Refine (CaR) framework in handling hard-constrained TSPTW and CVRPBLTW instances. Additional results are provided in Appendix F.

Experimental settings. Training instances are generated on the fly as in [12, 18] (see Appendix B). For TSPTW, we mainly focus on the hard variants in [12]. All experiments are conducted on problem sizes $n = 50/100$, following established benchmarks [8, 10]. Models are trained with 20,000 instances per epoch for 5,000 epochs with a batch size of 128 [18]. We set $T_{\text{R}} = 5$ during training. During inference, $8\times$ augmentation [9] is used to construct initial solutions, followed by T_{R}-step refinement.

Baseline. We compare CaR with *SoTA* traditional solvers (LKH3 [4], OR-Tools [3], and Greedy heuristics) and neural solvers, including construction methods (POMO [9]+EAS [19]+SGBS [20], PIP [12]) and the improvement solver NeuOpt-GIRE [11]. To ensure fairness, we upgrade NeuOpt-GIRE and POMO-based solvers with our relaxed CMDP formulation (*) and solution-level features (‡; see Appendix D). All models are trained to convergence with comparable training budgets.

Evaluation metrics. We evaluate performance using: 1) average solution length (Obj.), the mean length of best feasible solutions; 2) average optimality gap (Gap), the difference from (near-)optimal solutions found by top traditional solvers (LKH3 for TSPTW, OR-Tools for CVRPBLTW, HGS for CVRP, marked with $\diamond$); 3) total inference time (Time) for solving 10,000 TSPTW or 1,000 CVRP/CVRPBLTW instances using single-GPU parallelization; and 4) average infeasible solution rate per instance (Infsb%) after construction and refinement.

4.1 Model performance on constrained VRPs

TSPTW results. We first test CaR on TSPTW, where most neural solvers fail due to the lack of feasibility masking. To verify generality, we use POMO* and PIP as construction backbones, with CaR-POMO training 1.42× (at n=50) and 1.65× (at n=100) faster than CaR-PIP. As shown in Table 1, CaR-POMO consistently outperforms PIP in both quality and feasibility. On TSPTW-50, CaR reduces infeasibility to 0.00%, outperforming PIP (1.87%) and NeuOpt* (0.02%). On TSPTW-100, CaR lowers PIP's 4.67% infeasibility to 0.02% and reduces the gap from 1.030% to 0.146%. While NeuOpt* improves with longer runtimes, CaR achieves competitive results with an 8× speedup within a runtime budget of 10 minutes (Figure 2), which aligns with CaR's aim of efficiency. Notably, CaR surpasses LKH3 and finds feasible solutions even when it fails, highlighting the strength of our cross-paradigm framework under complex constraints. .

CVRPBLTW results. On complex CVRPBLTW, feasibility masking filters out over 60% of nodes, severely limiting the search space (Figure 6). Interestingly, removing these masks and applying the relaxed CMDP in POMO significantly improves performance (e.g., CVRPBLTW-100: from 7.004% to -1.645% in Table 1). Unlike TSPTW, NeuOpt* fails in CVRPBLTW with 27-56% infeasibility, while CaR guarantees feasibility as other construction solvers. We compare CaR with the best single-paradigm solvers in Figure 2. CaR achieves best area under the curve, indicating superior efficiency and effectiveness. We also validate CaR with k-opt and R&R, where R&R performs better (-2.448% vs. -1.724%) due to a finer-grained search better suited to multi-constraints variants.

Table 1: Results on constrained VRPs: best are **bolded**; best within 1 min are shaded to show solver efficiency.

	Method	#Params	Paradigm [§]	n=50 Obj. ↓	n=50 Gap ↓	n=50 Infsb% ↓	n=50 Time	n=100 Obj. ↓	n=100 Gap ↓	n=100 Infsb% ↓	n=100 Time
TSPTW	LKH3 (max trials = 100)	/	I	25.590	0.004%	11.88%	7m	46.625	0.103%	31.05%	27m
	LKH3 (max trials = 10000)	/	I	25.611	◇	0.12%	7h	46.858	◇	0.13%	1.4d
	OR-Tools[†]	/	I	25.763	-0.001%	65.72%	2.4h	46.424	0.026%	97.45%	12m
	Greedy-C	/	C	26.394	1.534%	72.55%	4.5s	51.945	9.651%	99.85%	11.4s
	POMO	1.25M	L2C	/	/	100.00%	4s	/	/	100.00%	14s
	POMO*	1.25M	L2C	26.222	1.635%	37.27%	4s	47.249	1.959%	38.22%	14s
	POMO* + PIP (greedy)	1.25M	L2C	25.657	0.177%	2.67%	7s	47.372	1.223%	6.96%	32s
	POMO* + PIP (sample 10)	1.25M	L2C	25.650	0.152%	1.87%	1m	47.294	1.030%	4.67%	5.2m
	NeuOpt-GIRE *‡ ($T = 1$k)	0.69M	L2I	25.627	0.061%	0.19%	2.3m	47.011	0.336%	0.13%	5.9m
	NeuOpt-GIRE *‡ ($T = 5$k)	0.69M	L2I	25.617	0.028%	0.02%	11.6m	46.913	**0.123%**	**0.02%**	30m
	CaR-POMO ($T_R = 5$)	1.64M	L2(C+I)	25.619	0.034%	0.02%	15s	47.278	1.065%	4.20%	36s
	CaR-POMO ($T_R = 20$)	1.64M	L2(C+I)	25.614	0.014%	0.01%	51s	47.001	0.406%	2.34%	2.1m
	CaR-PIP ($T_R = 5$)	1.64M	L2(C+I)	25.613	0.010%	0.02%	17s	47.000	0.315%	0.10%	58s
	CaR-PIP ($T_R = 20$)	1.64M	L2(C+I)	25.612	**0.005%**	**0.00%**	52s	46.923	0.146%	**0.02%**	2.4m
CVRPBLTW	OR-Tools (short)	/	I	14.890	1.402%	0.00%	10.4m	25.979	2.518%	0.00%	20.8m
	OR-Tools	/	I	14.677	◇	0.00%	1.7h	25.342	◇	0.00%	3.5h
	POMO	1.25M	L2C	15.999	9.169%	0.00%	2s	27.046	7.004%	0.00%	4s
	POMO+EAS+SGBS*	1.25M	L2C	15.156	3.263%	0.00%	10.3m	25.558	0.854%	0.00%	1h
	NeuOpt-GIRE *‡ ($T = 1$k)	0.69M	L2I	14.521	1.329%	33.80%	1.1m	24.832	4.290%	56.30%	2.9m
	NeuOpt-GIRE *‡ ($T = 5$k)	0.69M	L2I	14.201	**-1.163%**	27.30%	5.5m	24.237	-0.533%	41.20%	15m
	POMO*	1.25M	L2C	14.873	2.310%	0.00%	2s	24.592	-1.645%	0.00%	4s
	CaR (k-opt) ($T_R = 5$)	1.64M	L2(C+I)	14.872	2.271%	0.00%	3s	24.597	-1.674%	0.00%	5s
	CaR (k-opt) ($T_R = 20$)	1.64M	L2(C+I)	14.844	2.114%	0.00%	8s	24.585	-1.724%	0.00%	17s
	CaR (R&R) ($T_R = 5$)	1.72M	L2(C+I)	14.725	1.328%	0.00%	3s	24.552	-1.835%	0.00%	6s
	CaR (R&R) ($T_R = 20$)	1.72M	L2(C+I)	14.601	0.463%	**0.00%**	10s	24.400	**-2.448%**	**0.00%**	19s

[§] The abbreviations refer to: **I** – Improvement; **L2C** – Learning to Construct; **L2I** – Learning to Improve.
[†] OR-Tools presolves before search; if it detects infeasibility, it terminates immediately, making runtime shorter than preset.

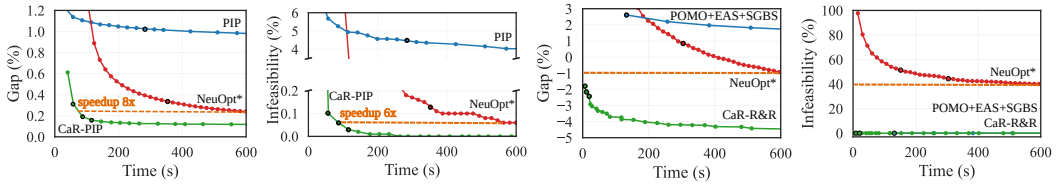

Figure 2: Performance over time on TSPTW-100 (left) and CVRPBLTW-100 (right). For CVRPBLTW, POMO+EAS+SGBS and CaR always achieve 0% infeasibility due to feasibility guarantee by masking. Dots with black circles represent results reported in Table 1.

4.2 Effects of the unified representation on handling complex VRPs

We study the effect of a unified encoder. As shown in Table 2, CaR with shared representation performs better on both hard-constrained cases (TSPTW-50 Hard and TSPTW-100 Medium), indicating improved knowledge transfer across paradigms. Compared with using separate encoders and decoders, which share only the constructed solutions, CaR also reduces the number of learnable parameters. See Appendix F.3 for further analysis.

Table 2: Effects of unified encoder on TSPTW.

Shared Rep.	#Params	TSPTW-50 Hard Infsb% ↓	TSPTW-50 Hard Gap ↓	TSPTW-100 Medium Infsb% ↓	TSPTW-100 Medium Gap ↓
×	2.8M	0.675%	0.199%	0.000%	7.589%
✓	1.6M	**0.010%**	**0.015%**	**0.000%**	**5.815%**

5 Conclusion

This paper proposes Construct-and-Refine (CaR), the first neural framework to handle constraints via paradigm integration. CaR jointly learns to construct diverse, high-quality solutions and refine them with a lightweight improvement module, enabling efficient constraint satisfaction. We also explore shared encoders for cross-paradigm representation learning. CaR performs strongly across constrained VRPs, offering key insights: 1) single-paradigm solvers struggle with complex constraints; 2) strict feasibility masking may hurt performance, while relaxed penalties help; 3) combining construction with light refinement efficiently recovers feasible, high-quality solutions; and 4) shared representations boost performance particularly for complex cases. Future work includes applying CaR to more VRPs, integrating diverse solvers, improving scalability, and developing foundation NCO models.

References

[1] Lu Duan, Yang Zhan, Haoyuan Hu, Yu Gong, Jiangwen Wei, Xiaodong Zhang, and Yinghui Xu. Efficiently solving the practical vehicle routing problem: A novel joint learning approach. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 3054–3063, 2020.

[2] Lixia Wu, Haomin Wen, Haoyuan Hu, Xiaowei Mao, Yutong Xia, Ergang Shan, Jianbin Zhen, Junhong Lou, Yuxuan Liang, Liuqing Yang, et al. Lade: The first comprehensive last-mile delivery dataset from industry. *arXiv preprint arXiv:2306.10675*, 2023.

[3] Vincent Furnon and Laurent Perron. Or-tools routing library, 2024. URL https://developers.google.com/optimization/routing/.

[4] Keld Helsgaun. LKH-3 (version 3.0.7), 2017. URL http://webhotel4.ruc.dk/~keld/research/LKH-3/.

[5] Federico Berto, Chuanbo Hua, Junyoung Park, Laurin Luttmann, Yining Ma, Fanchen Bu, Jiarui Wang, Haoran Ye, Minsu Kim, Sanghyeok Choi, Nayeli Gast Zepeda, André Hottung, Jianan Zhou, Jieyi Bi, Yu Hu, Fei Liu, Hyeonah Kim, Jiwoo Son, Haeyeon Kim, Davide Angioni, Wouter Kool, Zhiguang Cao, Jie Zhang, Kijung Shin, Cathy Wu, Sungsoo Ahn, Guojie Song, Changhyun Kwon, Lin Xie, and Jinkyoo Park. RL4CO: an extensive reinforcement learning for combinatorial optimization benchmark. *arXiv preprint arXiv:2306.17100*, 2023.

[6] Xuan Wu, Di Wang, Lijie Wen, Yubin Xiao, Chunguo Wu, Yuesong Wu, Chaoyu Yu, Douglas L Maskell, and You Zhou. Neural combinatorial optimization algorithms for solving vehicle routing problems: A comprehensive survey with perspectives. *arXiv preprint arXiv:2406.00415*, 2024.

[7] Daniela Thyssens, Tim Dernedde, Jonas K Falkner, and Lars Schmidt-Thieme. Routing arena: A benchmark suite for neural routing solvers. *arXiv preprint arXiv:2310.04140*, 2023.

[8] Wouter Kool, Herke van Hoof, and Max Welling. Attention, learn to solve routing problems! In *International Conference on Learning Representations*, 2018.

[9] 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*, volume 33, pages 21188–21198, 2020.

[10] Yaoxin Wu, Wen Song, Zhiguang Cao, Jie Zhang, and Andrew Lim. Learning improvement heuristics for solving routing problems. *IEEE Transactions on Neural Networks and Learning Systems*, 33(9):5057–5069, 2021.

[11] Yining Ma, Zhiguang Cao, and Yeow Meng Chee. Learning to search feasible and infeasible regions of routing problems with flexible neural k-opt. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023.

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

[13] 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 preprint arXiv:2403.05318*, 2024.

[14] Xinyun Chen and Yuandong Tian. Learning to perform local rewriting for combinatorial optimization. In *Advances in Neural Information Processing Systems*, volume 32, pages 6281–6292, 2019.

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

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

[17] Yining Ma, Jingwen Li, Zhiguang Cao, Wen Song, Hongliang Guo, Yuejiao Gong, and Yeow Meng Chee. Efficient neural neighborhood search for pickup and delivery problems. In *Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence*, pages 4776–4784, 7 2022.

[18] Jianan Zhou, Zhiguang Cao, Yaoxin Wu, Wen Song, Yining Ma, Jie Zhang, and Xu Chi. MVMoE: Multi-task vehicle routing solver with mixture-of-experts. In *International Conference on Machine Learning*, 2024.

[19] André Hottung, Yeong-Dae Kwon, and Kevin Tierney. Efficient active search for combinatorial optimization problems. In *International Conference on Learning Representations*, 2022.

[20] Jinho Choo, Yeong-Dae Kwon, Jihoon Kim, Jeongwoo Jae, André Hottung, Kevin Tierney, and Youngjune Gwon. Simulation-guided beam search for neural combinatorial optimization. In *Advances in Neural Information Processing Systems*, volume 35, pages 8760–8772, 2022.

[21] Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. Pointer networks. In *Advances in Neural Information Processing Systems*, volume 28, pages 2692–2700, 2015.

[22] Irwan Bello, Hieu Pham, Quoc V. Le, Mohammad Norouzi, and Samy Bengio. Neural combinatorial optimization with reinforcement learning. In *International Conference on Learning Representations Workshop Track*, 2017.

[23] 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*, pages 9861–9871, 2018.

[24] Haoran Sun, Katayoon Goshvadi, Azade Nova, Dale Schuurmans, and Hanjun Dai. Revisiting sampling for combinatorial optimization. In *International Conference on Machine Learning*, pages 32859–32874. PMLR, 2023.

[25] Minsu Kim, Junyoung Park, and Jinkyoo Park. Sym-NCO: Leveraging symmetricity for neural combinatorial optimization. In *Advances in Neural Information Processing Systems*, 2022.

[26] 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*, 2023.

[27] Felix Chalumeau, Shikha Surana, Clément Bonnet, Nathan Grinsztajn, Arnu Pretorius, Alexandre Laterre, and Thomas D Barrett. Combinatorial optimization with policy adaptation using latent space search. In *Advances in Neural Information Processing Systems*, 2023.

[28] Nathan Grinsztajn, Daniel Furelos-Blanco, Shikha Surana, Clément Bonnet, and Thomas D Barrett. Winner takes it all: Training performant RL populations for combinatorial optimization. In *Advances in Neural Information Processing Systems*, 2023.

[29] 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*, 2023.

[30] André Hottung, Mridul Mahajan, and Kevin Tierney. PolyNet: Learning diverse solution strategies for neural combinatorial optimization. In *International Conference on Learning Representations*, 2025.

[31] Fu Luo, Xi Lin, Zhenkun Wang, Tong Xialiang, Mingxuan Yuan, and Qingfu Zhang. Self-improved learning for scalable neural combinatorial optimization. *arXiv preprint arXiv:2403.19561*, 2024.

[32] Daisuke Kikuta, Hiroki Ikeuchi, Kengo Tajiri, and Yuusuke Nakano. Routeexplainer: An explanation framework for vehicle routing problem. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pages 30–42. Springer, 2024.

[33] Zefang Zong, Hansen Wang, Jingwei Wang, Meng Zheng, and Yong Li. Rbg: Hierarchically solving large-scale routing problems in logistic systems via reinforcement learning. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 4648–4658, 2022.

[34] Yan Jin, Yuandong Ding, Xuanhao Pan, Kun He, Li Zhao, Tao Qin, Lei Song, and Jiang Bian. Pointerformer: Deep reinforced multi-pointer transformer for the traveling salesman problem. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, pages 8132–8140, 2023.

[35] Qingchun Hou, Jingwei Yang, Yiqiang Su, Xiaoqing Wang, and Yuming Deng. Generalize learned heuristics to solve large-scale vehicle routing problems in real-time. In *International Conference on Learning Representations*, 2023.

[36] James Fitzpatrick, Deepak Ajwani, and Paula Carroll. A scalable learning approach for the capacitated vehicle routing problem. *Computers & Operations Research*, 171:106787, 2024.

[37] Simon Geisler, Johanna Sommer, Jan Schuchardt, Aleksandar Bojchevski, and Stephan Günnemann. Generalization of neural combinatorial solvers through the lens of adversarial robustness. In *International Conference on Learning Representations*, 2022.

[38] Pei Xiao, Zizhen Zhang, Jinbiao Chen, Jiahai Wang, and Zhenzhen Zhang. Neural combinatorial optimization for robust routing problem with uncertain travel times. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024.

[39] Zeyang Zhang, Ziwei Zhang, Xin Wang, and Wenwu Zhu. Learning to solve travelling salesman problem with hardness-adaptive curriculum. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2022.

[40] 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*, 2022.

[41] Yuan Jiang, Yaoxin Wu, Zhiguang Cao, and Jie Zhang. Learning to solve routing problems via distributionally robust optimization. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2022.

[42] Jianan Zhou, Yaoxin Wu, Wen Song, Zhiguang Cao, and Jie Zhang. Towards omni-generalizable neural methods for vehicle routing problems. In *International Conference on Machine Learning*, pages 42769–42789. PMLR, 2023.

[43] 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 Thirty-First International Joint Conference on Artificial Intelligence*, 2024.

[44] Han Fang, Zhihao Song, Paul Weng, and Yutong Ban. Invit: A generalizable routing problem solver with invariant nested view transformer. In *Forty-first International Conference on Machine Learning*, 2024.

[45] Han Lu, Zenan Li, Runzhong Wang, Qibing Ren, Xijun Li, Mingxuan Yuan, Jia Zeng, Xiaokang Yang, and Junchi Yan. ROCO: A general framework for evaluating robustness of combinatorial optimization solvers on graphs. In *International Conference on Learning Representations*, 2023.

[46] Chenguang Wang and Tianshu Yu. Efficient training of multi-task combinarotial neural solver with multi-armed bandits. *arXiv preprint arXiv:2305.06361*, 2023.

[47] Fei Liu, Xi Lin, Zhenkun Wang, Qingfu Zhang, Tong Xialiang, 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*, pages 1898–1908, 2024.

[48] Federico Berto, Chuanbo Hua, Nayeli Gast Zepeda, André Hottung, Niels Wouda, Leon Lan, Junyoung Park, Kevin Tierney, and Jinkyoo Park. Routefinder: Towards foundation models for vehicle routing problems. *arXiv preprint arXiv:2406.15007*, 2024.

[49] Zhuoyi Lin, Yaoxin Wu, Bangjian Zhou, Zhiguang Cao, Wen Song, Yingqian Zhang, and Senthilnath Jayavelu. Cross-problem learning for solving vehicle routing problems. *arXiv preprint arXiv:2404.11677*, 2024.

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

[51] Benjamin Hudson, Qingbiao Li, Matthew Malencia, and Amanda Prorok. Graph neural network guided local search for the traveling salesperson problem. In *International Conference on Learning Representations*, 2022.

[52] Ruizhong Qiu, Zhiqing Sun, and Yiming Yang. Dimes: A differentiable meta solver for combinatorial optimization problems. *Advances in Neural Information Processing Systems*, 35:25531–25546, 2022.

[53] Zhiqing Sun and Yiming Yang. Difusco: Graph-based diffusion solvers for combinatorial optimization. In *Advances in Neural Information Processing Systems*, 2023.

[54] Yimeng Min, Yiwei Bai, and Carla P Gomes. Unsupervised learning for solving the travelling salesman problem. *Advances in Neural Information Processing Systems*, 2023.

[55] Kexiong Yu, Hang Zhao, Yuhang Huang, Renjiao Yi, Kai Xu, and Chenyang Zhu. Disco: Efficient diffusion solver for large-scale combinatorial optimization problems. *arXiv preprint arXiv:2406.19705*, 2024.

[56] Paulo da Costa, Jason Rhuggenaath, Yingqian Zhang, and Alp Eren Akçay. Learning 2-opt heuristics for the traveling salesman problem via deep reinforcement learning. In *Asian Conference on Machine Learning*, pages 465–480, 2020.

[57] 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*, volume 34, pages 11096–11107, 2021.

[58] André Hottung and Kevin Tierney. Neural large neighborhood search for routing problems. *Artificial Intelligence*, page 103786, 2022.

[59] Minjun Kim, Junyoung Park, and Jinkyoo Park. Learning to cross exchange to solve min-max vehicle routing problems. In *The Eleventh International Conference on Learning Representations*, 2023.

[60] Minsu Kim, Jinkyoo Park, and joungho kim. Learning collaborative policies to solve np-hard routing problems. In *Advances in Neural Information Processing Systems*, volume 34, pages 10418–10430, 2021.

[61] 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 AAAI Conference on Artificial Intelligence*, 2024.

[62] Zhi Zheng, Changliang Zhou, Tong Xialiang, Mingxuan Yuan, and Zhenkun Wang. Udc: A unified neural divide-and-conquer framework for large-scale combinatorial optimization problems. *Advances in Neural Information Processing Systems*, 2024.

[63] Detian Kong, Yining Ma, Zhiguang Cao, Tianshu Yu, and jianhua Xiao. Efficient neural collaborative search for pickup and delivery problems. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2024.

[64] 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.

[65] Sirui Li, Zhongxia Yan, and Cathy Wu. Learning to delegate for large-scale vehicle routing. *Advances in Neural Information Processing Systems*, 34:26198–26211, 2021.

Appendix

A Detailed literature review

A.1 Neural VRP solvers

Generally, neural VRP solvers can be broadly categorized into two paradigms: construction solvers and improvement solvers.

1) *Construction-based solvers* learn to construct solutions from scratch in an end-to-end fashion. Vinyals et al. [21] introduced the Pointer Network (PtrNet), leveraging Recurrent Neural Networks (RNN) to solve the Traveling Salesman Problem (TSP) in a supervised manner. Building on this, Bello et al. [22] explored reinforcement learning (RL) training for PtrNet. Nazari et al. [23] extended the approach to solve CVRP in an autoregressive (AR) way. Among AR solvers, the Attention Model (AM) [8] stands out as a milestone to solve multiple VRPs. This was further advanced by the Policy Optimization with Multiple Optima (POMO) [9], which leverages diverse rollouts inspired by the symmetry properties of VRP solutions. Subsequently, numerous studies have advanced AR solvers in various perspectives, such as inference strategies [19, 20, 24], training paradigms [25–31], interpretability [32], scalability [33–36], robustness [37, 38], benchmarking [7], and generalization over different distributions [39–41], scales [42–44], and constraints [18, 45–49]. Beyond AR solvers, another line of research predicts heatmaps in a non-autoregressive (NAR) manner to represent edge probabilities for optimal solutions [50–55]. With the learned heatmap, these solvers can greatly reduce the search space. Despite showing better scalability, NAR solvers often depend on post-search procedures, which can be either time-consuming or ineffective in handling VRP constraints, even for the simple cases such as CVRP.

2) *Improvement-based solvers* learns to iteratively improve initial solutions, drawing inspiration from traditional (meta-)heuristics such as k-opt (e.g., 2-opt [10, 56, 57], extended to flexible k-opt [11]), ruin-and-repair [17, 58], and crossover [59]. In general, improvement-based methods can achieve

near-optimal solutions given prolonged search time, whereas construction-based methods typically offer a more efficient trade-off between performance and runtime.

A.2 Constraint handling for VRPs

Classic neural methods handle VRP constraints using feasibility masking to exclude invalid actions [9, 57]. While effective for simpler VRPs (e.g., CVRP), such masking fails on more complex VRPs. Firstly, calculating masking itself may be NP-hard [12], e.g., for TSPTW and TSPDL. While recent methods have explored Lagrangian-based problem reformulation [15], lookahead strategies [13], and learnable approximated masks to prevent infeasibility [12], they may incur high computational costs and still yield high infeasibility rates. Secondly, current methods struggle with multi-constraint VRPs, e.g., CVRPBLTW, where construction faces overly restrictive masks and improvement drifts into infeasible regions (see Section 4). Moreover, no feasibility handling scheme effectively addresses both types of the above challenges. Hence, an efficient, general, and effective constraint-handling framework for neural VRP solvers remains absent.

A.3 Hybrid VRP solvers

Recent research has explored hybridizing construction-based approaches with search mechanisms to enhance combinatorial optimization solvers. For instance, active search [19] and beam search [20] have been used to augment construction policies. However, these methods are not well-suited for hard-constrained vehicle routing problems (VRPs), where feasibility is critical. Other strategies, such as collaborative policies [60], random reconstruction [29], and neural divide-and-conquer frameworks (e.g., GLOP [61] and UDC [62]), have shown strong performance on CVRP but often struggle to maintain feasibility when solutions are decomposed or reconstructed, even with penalty-based guidance.

Beyond post-construction search, hybrid methods that integrate construction with intensive improvement phases have been proposed. RL4CO [5] combines pretrained POMO [9] and NeuOpt [11] at inference time, while NCS [63] employs a shared critic to jointly train the two modules. These approaches, however, depend on separately optimized components and often require extensive improvement steps (e.g., 5,000 iterations), limiting their efficiency and scalability.

Our approach distinguishes itself in three critical ways: 1) it adopts a joint training framework with unified representation learning, unlike prior works that treat construction and improvement as separate [5] or loosely coupled [63] components; 2) it achieves efficient and precise refinement in as few as 10 steps, rather than relying on prolonged improvement; and 3) it is the first to systematically explore the synergy between construction and improvement for effective constraint handling in complex VRPs.

Among the most relevant hybrid baselines, LCP [60], RL4CO [5], and NCS [63] are not directly applicable to complex constrained VRPs due to significant limitations: LCP's divide-and-conquer strategy impairs constraint handling; RL4CO lacks joint optimization; and NCS fails to generate feasible moves or predict value functions accurately under hard constraints, leading to 100% infeasibility on TSPTW in our preliminary evaluations.

B Problem selection and data generation

In this paper, we consider two representative VRPs: complex constrained VRPs where feasibility masking is NP-hard thus intractable (TSPTW) or tractable but ineffective (CVRPBLTW) Below, we introduce the detailed data generation process. Following the convention, all the node coordinates are generated under a uniform distribution, i.e., $x_i, yi \sim \mathcal{U}[0, 1]$.

B.1 Traveling Salesman Problem with Time Window (TSPTW)

We primarily follow the settings of a recent study [12], which introduced TSPTW with three difficulty levels: Easy [13], Medium, and Hard. Given our focus on complex constrained VRPs, we report results on the *Hard* variant in the main tables, consistent with the benchmark dataset in [64]. Except for the experiment in Table 2, where we use the Medium variant to show how CaR's unified encoder works as constraint hardness changes, all other TSPTW experiments default to the Hard setting.

NP-hardness of computing feasibility masking on TSPTW. During solution construction, neural solvers typically apply feasibility masking to exclude actions that violate constraints. In TSPTW, for instance, a node v_j is masked out if its arrival time t_j exceeds its time window end u_j, i.e., $t_j > u_j$. However, as highlighted in PIP [12], feasibility in TSPTW is not purely local: selecting a node affects the current time, which in turn influences the feasibility of all future selections due to interdependent time window constraints. For example, a node with a late time window might be locally feasible, but choosing it can delay the tour such that earlier nodes become unreachable, leading to irreversible infeasibility. Ensuring global feasibility thus requires evaluating whether a current decision allows for any feasible completions, i.e., a process that involves simulating all future possibilities. This renders feasibility masking itself NP-hard, compounding the inherent difficulty of solving TSPTW.

TSPTW Hard. We first generate a random permutation τ of the node set and sequentially assign time windows to ensure the existence of a feasible solution for each instance. The time windows are drawn from a uniform distribution, where the lower and upper bounds are given by $l_i \sim \mathcal{U}\left[C_L(\tau'_i) - \eta, C_L(\tau'_i)\right], u_i \sim \mathcal{U}\left[C_L(\tau'_i), C_L(\tau'_i) + \eta\right]$, where $C_L(\tau'_i)$ denotes the cumulative tour length of the partial sequence τ' up to step i, and η controls the time window width. To adaptively scale time windows with problem size, we set $\eta = n$, offering greater flexibility than the fixed value of 50 used in [12]. Time windows are then normalized following [8] to facilitate neural network training. Since TSP solutions form a Hamiltonian cycle, it is equivalent to setting any node as the starting point. Thus, we designate a specific starting node for TSPTW by redefining its upper bound u_0 as $u_0 = \max\left(u_i + C_L(e(v_i, v_0))\right), i \in [1, n]$.

TSPTW Medium. We generate the lower and upper bounds of the time windows following a uniform distribution: $l_i \sim \mathcal{U}[0, T_N]$, where T_N estimates the expected tour length for the given problem scale (e.g., $T_{20} \approx 10.9$ [13]). The upper bound u_i is derived from l_i as $u_i \sim l_i + T_N \cdot \mathcal{U}[0.1, 0.2]$. While this data generation rule does not guarantee instance feasibility, preliminary results show that the time windows overlap significantly, leading to a high feasibility rate in the generated instances.

For all the TSPTW instances, we normalize all l_i and u_i by u_0 to ensure their values fall within $[0, 1]$. Training data is generated on the fly, while inference uses the dataset from [12] for fair comparison. As noted in [12], all test instances are verified to be feasible, either empirically or theoretically.

B.2 Capacitated VRP with Backhaul, Duration Limit, and Time Window Constraints (CVRPBLTW)

CVRPBLTW follows the setting in [18, 47, 65] and includes four key constraints: 1) *Capacity (C).* Each node's demand q_i is sampled from $\mathcal{U}(1, \ldots, 9)$, and the vehicle's capacity Q varies by problem scale, with $Q^{50} = 40$ and $Q^{100} = 50$. 2) *Backhaul (B).* Node demands are sampled from a discrete uniform distribution $\{1, \ldots, 9\}$, with 20% of customers randomly designated as backhauls. Routes include both linehauls and backhauls without strict precedence constraints. 3) *Duration Limit (L).* The maximum route length is set to $\ell = 3$, ensuring feasible solutions in the unit square space $\mathcal{U}(0, 1)$. 4) *Time Window (TW).* The depot time window is defined as $[l_0, u_0] = [0, 3]$, and each customer node has a service time of $s_i = 0.2$. The time window for node v_i is computed as follows: the center is sampled as $\gamma_i \sim \mathcal{U}(l_0 + C_L(e(v_0, v_i)), u_0 - C_L(e(v_i, v_0)) - s_i)$, where $C_L(e(v_0, v_i)) = C_L(e(v_i, v_0))$ represents the travel time between the depot v_0 and node v_i. The half-width is drawn from $w_i \sim \mathcal{U}(\frac{s_i}{2}, \frac{u_0}{3}) = \mathcal{U}(0.1, 1)$. Finally, the time window is set as $[l_i, u_i] = [\max(l_0, \gamma_i - w_i), \min(u_0, \gamma_i + w_i)]$. Training data is generated on the fly, while inference uses the 1k-instance dataset from [18] for fair comparison.

C Limitations of individual paradigms for complex constraint handling

Figure 3 depicts the illustrative search behaviors of neural solvers on two hard-constrained VRPs: i) CVRPBLTW (a–c) where feasibility masking for construction is tractable but overly restrictive, leading to a fragmented feasible space due to multiple constraints; and ii) TSPTW (d-f), where computing feasibility masks is NP-hard and thus intractable [12], yielding a limited feasible region.

Both construction and improvement solvers may struggle with these hard constraints. In CVRPBLTW, overly restrictive masking limits the exploration of construction in the fragmented feasible space, causing early convergence to local optima (Figure 3(a)). This aligns with our findings in Section 4: removing hard masking for CVRPBLTW improves performance (POMO* vs. POMO in Table 1) but

still yields infeasible and suboptimal solutions (Appendix F.1). For TSPTW, the relaxed CMDP in Eq.(2) offers partial relief, but Bi et al. [12] show it fails in complex cases and propose a learnable mask to guide the policy toward near-feasible regions (Figure 3(d)). Still, the performance of construction methods alone remains limited due to the rigidity of inherent stepwise feasibility. For improvement methods with advanced operators (e.g., k-opt), masking is inapplicable and thus unavailable in both cases, often resulting in inefficient search trajectories through both feasible and infeasible regions. As shown in Figure 3(b) and Figure 3(e), while they can refine solutions, they are computationally inefficient, often requiring substantial time to find good initial solutions, and performance degrades significantly with multiple constraints and limited search steps (Table 1).

To address these limitations, we propose CaR, which unifies the strengths of both paradigms by using construction to generate diverse, high-quality, near-feasible initial solutions, thereby reducing the required refinement iterations. As shown in Figures 3(c) and (f), CaR applies lightweight refinement with only a few steps to help construction escape local optima and reach higher-quality local optima, enabling more effective and efficient constraint handling for complex VRPs.

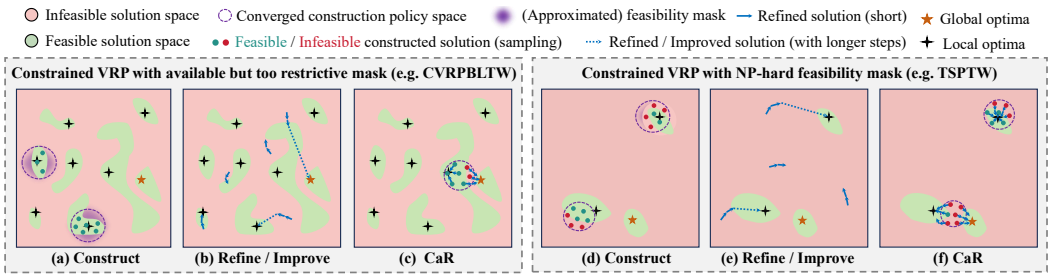

Figure 3: Illustration of the solution space and paradigm behaviors for two hard-constrained VRPs.

D Network architecture

The network architectures of mainstream neural construction and improvement solvers are typically based on a Transformer encoder with an attention-based decoder, enabling unification across both paradigms. In this paper, we adopt a unified encoder shared by the construction and refinement decoders. Specifically, we use a 6-layer Transformer encoder, following POMO [9], while retaining the original decoders from NeuOpt [11] and N2S [17], corresponding to two representative local search operators: flexible k-opt [11] and remove-and-reinsertion (R&R) [17], respectively. For the unexplored variants TSPTW and CVRPBLTW, we design new constraint-related features analogous to the contextual features used in the original decoders. The concrete forward processes are introduced below.

D.1 Encoder

As shown in Figure 4, the construction module first takes node features f_i^{n}—including coordinates (x_i, y_i) and constraint-related features (e.g., time windows $[l_i, u_i]$ and demand q_i)—as input. For each node v_i, these features are projected into a d-dimensional embedding $\boldsymbol{h}_i^{(0)} \in \mathbb{R}^d$ ($d = 128$) via a linear layer. The initial embedding is then passed through a 6-layer Transformer network [9]. At each layer j ($j = 1, \cdots, 6$), the embedding $\boldsymbol{h}_i^{(j-1)}$ is projected into query, key, and value vectors:

$$\boldsymbol{q}_i^{(j)} = W_q^{(j)} \boldsymbol{h}_i^{(j-1)}, \quad \boldsymbol{k}_i^{(j)} = W_k^{(j)} \boldsymbol{h}_i^{(j-1)}, \quad \boldsymbol{v}_i^{(j)} = W_v^{(j)} \boldsymbol{h}_i^{(j-1)}, \tag{6}$$

where $W_q^{(j)}, W_k^{(j)}, W_v^{(j)} \in \mathbb{R}^{d \times d}$. These are fed into a multi-head attention (MHA) layer, whose output is:

$$\tilde{\boldsymbol{h}}_i^{(j)} = \text{Softmax}\left(\frac{(\boldsymbol{q}_i^{(j)})^\top \boldsymbol{k}_i^{(j)}}{\sqrt{d}} \right) \boldsymbol{v}_i^{(j)}. \tag{7}$$

The MHA output is then linearly transformed:

$$\hat{\boldsymbol{h}}i^{(j)} = W_{\text{MHA}}^{(j)} \tilde{\boldsymbol{h}}i^{(j)}, \tag{8}$$

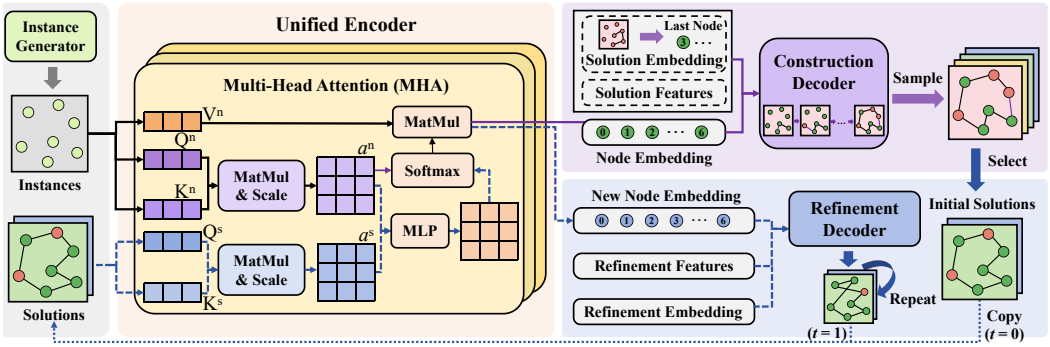

Figure 4: Overview of the unified network architecture in CaR. Blue dashed arrows indicate information flow specific to refinement, while purple dashed arrows indicate flow exclusive to construction.

where $W\text{MHA}^{(j)} \in \mathbb{R}^{d \times d}$. This passes through instance normalization with residual connection:

$$\boldsymbol{h}'^{(j)}_i = \text{IN}\left(\hat{\boldsymbol{h}}^{(j)}_i + \boldsymbol{h}^{(j-1)}_i\right). \tag{9}$$

Next, a feed-forward (FF) layer refines the output:

$$\boldsymbol{h}''^{(j)}_i = W_2^{(j)} \cdot \text{ReLU}\left(W_1^{(j)} \boldsymbol{h}'^{(j)}_i\right), \tag{10}$$

where $W_1^{(j)} \in \mathbb{R}^{d \times d'}$, $W_2^{(j)} \in \mathbb{R}^{d' \times d}$, and $d' = 512$ as in [9]. The final embedding at layer j is:

$$\boldsymbol{h}^{(j)}_i = \text{IN}\left(\boldsymbol{h}''^{(j)}_i + \boldsymbol{h}'^{(j)}_i\right). \tag{11}$$

Thereafter, this process is repeated for 6 layers, and the final encoder output is $\boldsymbol{h}_i = \boldsymbol{h}^{(6)}_i$.

When it comes to the refinement module, the input changes from a node feature set to a linked list (i.e., the solution), which consists of the original node features plus positional information and a pointer to the next node in the solution. Notably, for CVRP variants, the depot node may appear multiple times, with each occurrence treated as a distinct node due to its different position. The refinement module shares the same operations as the construction module, except for Eq. (7). Specifically, we first incorporate the cyclic positional encoding (CPE) from [57] to obtain the positional embedding $p_i \in \mathbb{R}^d$. We then apply a self-attention mechanism:

$$\tilde{p}_i = (W_q^p p_i)^\top (W_k^p p_i), \tag{12}$$

where $W_q^p, W_k^p \in \mathbb{R}^{d \times d}$, to compute the positional attention matrix a^{s}. We replace Eq. (7) from the construction module with:

$$\tilde{\boldsymbol{h}}^{(j)}_i = \text{Softmax}\left(\frac{\text{MLP}\left(\left[(\boldsymbol{q}^{(j)}_i)^\top \boldsymbol{k}^{(j)}_i, \tilde{p}_i\right]\right)}{\sqrt{d}}\right) \boldsymbol{v}^{(j)}_i, \tag{13}$$

where the MLP reduces the dimension of the concatenated attention scores, i.e., $[a^{\text{n}}, a^{\text{s}}]$ in Figure 4, from 2 to 1, following the Syn-Att mechanism proposed in [17].

D.2 Decoder

The construction decoder primarily employs an MHA layer, where the key and value vectors are linearly transformed from the node embedding h_i. The query vector is computed using contextual information from the construction process, including: 1) the embedding of the partial solution h^{s} (i.e., the node embedding of the last node in the partial solution), and 2) the step-wise solution features f^{s}, which include the vehicle's remaining load, current time, and current sub-tour length. The MHA output is calculated as:

$$a = \sum_{i=0}^{n} \text{Softmax}\left(\frac{q_i^\top k_i}{\sqrt{d}} + \xi_i\right) v_i, \tag{14}$$

where ξ_i is the feasibility mask for node v_i. We mask out visited nodes to ensure solution validity across all variants. For CVRP, as discussed in Appendix F.1, we also mask nodes that violate capacity constraints at each construction step. The output of each head is then passed through a linear layer parameterized by W_o, yielding $h_a = W_o a$. Finally, the decoder computes selection probabilities for all candidate nodes using a single-head attention layer:

$$p_i = \text{Softmax}\left(\zeta \cdot \tanh\left(\frac{h_a^\top h_i}{\sqrt{d}}\right) + \xi_i\right), \tag{15}$$

where ζ scales the logits to encourage exploration [9].

The refinement decoder follows the original design in the NeuOpt [11] and N2S [17], which performs two representative local search operators: the flexible k-opt[2] and remove-and-reinsertion (R&R), respectively. Figure 5 illustrates the detailed improvement process in a single time step for different local search operators, where each node selection corresponds to a full computation of the network decoder. Note that although the original NeuOpt and N2S decoders adopt different architectures, their input and output formats are similar. At each refinement step $t = 1, \ldots, T_{\text{R}}$, the inputs include: (i) the updated *node embedding* h_i^t derived from the previous solution τ_{t-1} (with τ_0 as the top-p initial construction solutions); (ii) *refinement features* encoding feasibility transition history for k-opt or removal action history for R&R; and (iii) *refinement embedding* of the last action node, analogous to the solution embedding in construction. The output is the selection probability over candidate nodes for the next refinement operation (e.g., k-opt, removal, or insertion).

To adapt improvement solvers to new variants (e.g., TSPTW, CVRPBLTW), we follow their original design principles while integrating variant-specific features, such as refinement history and node-level feasibility information. Specifically, when using the NeuOpt decoder, we encode the feasibility of the most recent three refinement steps as a binary vector to represent refinement history and incorporate node-level feasibility features to enhance constraint awareness. For TSPTW, these features include the arrival time at each node, time window violation value, last node arrival time, and an indicator of whether the solution becomes infeasible after visiting the current node. For CVRPBLTW, node-level feasibility features include binary indicators for depot and backhaul nodes, violation values for each constraint, constraint-related attributes (e.g., arrival time and cumulative distance/demand), and infeasibility markers indicating whether the solution becomes infeasible before or after visiting the current node. Adapted NeuOpt variants are marked with ‡ in the main tables.

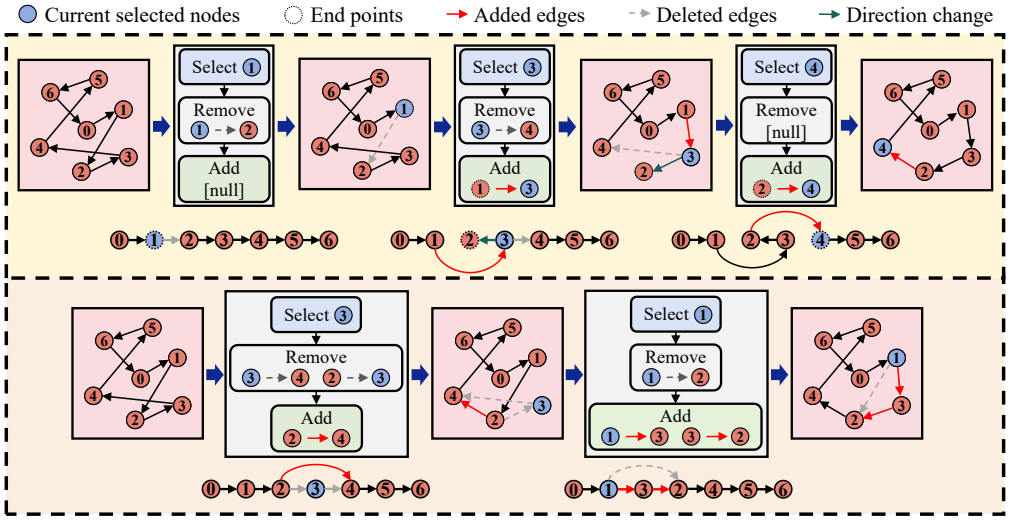

Figure 5: Illustration of actions and operations in k-opt (*top*) and remove-and-reinsertion (*bottom*).

E Experimental settings

We preset refinement steps $T_{\text{R}} = 5$ during training. Models are trained using Adam with a learning rate of 1×10^{-4}, weight decay of 1×10^{-6}. Follow the setting of [9], we use instance normalization

[2]Following [11], we set $k \leq 4$.

in the MHA and set the embedding dimension to 128. To prevent out-of-memory issues early in training, refinement is activated after a $\sim$10-epoch warmup of the construction module. To optimize GPU memory utilization, we adjust the number of refined solutions p separately for each problem size during training. We set $\alpha_1 = 0.01$ and $\alpha_2 = 1$ in the construction loss, and set $\omega = 100$ for TSPTW and CVRP, and 10 for CVRPBLTW based on the relative loss scales between modules observed during warm-up epochs. During inference, TSPTW and CVRPBLTW results are obtained without multi-starting, consistent with the training setup. For each augmented instance, we sample one solution for TSPTW and greedily generate one for CVRPBLTW ($p = 1$), which are then passed to the refinement module. For CVRP, Table 3 shows that multi-starting improves performance, so we apply it during construction but refine only the top p solutions for efficiency, setting $p = 2$ for $n = 50$ and $p = 1$ for $n = 100$ due to the GPU memory constraint. We use only one GPU for all problem sizes during inference to ensure fair comparison across variants. All experiments are ran on servers equipped with NVIDIA GeForce RTX 4090 GPUs and Intel(R) Core i9-10940X CPUs at 3.30GHz.

F Additional experiments and discussion

F.1 Discussion of the existing feasibility masking

We now present our insights into the limitations of existing feasibility masking mechanisms. As shown in Table 3, removing feasibility masking for CVRP hampers its performance, in contrast to more complex problems like CVRPBLTW, where its removal leads to improved results (see results of POMO* vs. POMO in Table 1). This suggests that *while existing feasibility masking is sufficient for simple problems, it is far from a silver bullet for handling complex constraints.* In fact, for highly constrained problems, feasibility masking is often intractable (e.g., TSPTW) or ineffective (e.g., CVRPBLTW), resulting in infeasibility and sub-optimality.

Table 3: Effects of the multi-starting strategy in [9] and feasibility masking during construction on CVRP-50.

w. multi-start	*w.* mask	Obj.↓	Gap↓	Infsb%↓
×	×	10.550	2.093%	0.00%
×	✓	10.520	1.793%	0.00%
✓	×	10.431	0.921%	0.00%
✓	✓	10.424	**0.857%**	0.000%

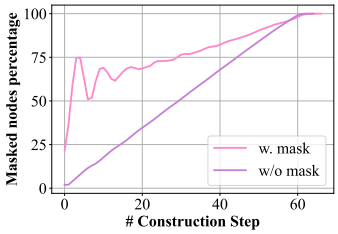

Figure 6: Masked node count in CVRPBLTW construction.

To address these challenges, we propose the CaR framework. For TSPTW, CaR leverages refinement to correct infeasibility and improve solution quality. For CVRPBLTW, we found that removing this restrictive mask significantly improves the performance (Gap: 9.17% vs. 2.31%), but it leads to unexpected infeasibility (2.6%). CaR addresses this by applying refinement, which further reduces infeasibility and improves solution quality. However, a small fraction of infeasible solutions can still remain. Since feasibility can ultimately be guaranteed through masking, we apply a final reconstruction step with feasibility masking during inference to ensure all reported solutions are valid (see Table 4). This strategy allows CaR to relax constraints during construction and refinement to enhance quality while maintaining overall feasibility through targeted post-processing. Note that in this paper, “without masks" does not imply removing all masks. All VRP variants must satisfy the fundamental constraint that each node (except depots) is visited exactly once. To ensure the validity of the generated solutions, we retain the masking of visited nodes during the construction process.

F.2 Discussion of the multi-start strategy and the diversification

Following the insights from [30] and [12], the multi-start strategy proposed in POMO [9] negatively impacts performance when training on VRP variants with TW constraints. Therefore, in this paper, we sample S start nodes instead of enumerating all nodes for constrained variants such as TSPTW and CVRPBLTW. Motivated by the advantage of diversification by multi-start, we add the diversity loss in Eq. (4), which is empirically proven to be effective (see Figure 10).

Table 4: CaR module impact on CVRPBLTW feasibility and quality (**Bold**: reported in Table 1).

Method	Module	n=50 Obj.↓	n=50 Gap↓	n=50 Infsb%↓	n=100 Obj.↓	n=100 Gap↓	n=100 Infsb%↓
CaR	Construction (w/o mask)	14.798	2.116%	3.50%	24.404	-2.050%	2.80%
(k-opt)	Refinement (k-opt)	14.759	1.682%	2.50%	24.360	-2.285%	2.60%
($T_{\mathrm{R}} = 20$)	Re-construction (w. mask)	**14.844**	**2.114%**	0.00%	**24.585**	**-1.724%**	0.00%
CaR	Construction (w/o mask)	14.843	2.235%	2.60%	24.460	-1.715%	3.10%
(R&R)	Refinement (R&R)	14.567	0.221%	1.10%	24.215	-2.886%	1.90%
($T_{\mathrm{R}} = 20$)	Re-construction (w. mask)	**14.601**	**0.463%**	0.00%	**24.400**	**-2.448%**	0.00%

Table 5: Effects of unified encoder on TSPTW.

Method	TSPTW-50 Medium Infsb%↓	TSPTW-50 Medium Gap↓	TSPTW-50 Hard Infsb%↓	TSPTW-50 Hard Gap↓	TSPTW-100 Medium Infsb%↓	TSPTW-100 Medium Gap↓
Construction-only	3.77%	5.230%	37.270%	1.635%	0.120%	10.931%
Separate	0.010%	**2.047%**	0.675%	0.199%	0.000%	7.589%
Unified (Ours)	**0.000%**	2.173%	**0.010%**	**0.014%**	**0.000%**	**5.815%**

Table 6: Model parameters and performance of different unifications on TSPTW-50 Hard.

Encoder	Decoder	#Params	Infsb%↓	Gap↓
Separate	Separate	2.8M	0.675%	0.199%
Unified	Unified	1.4M	**0.010%**	0.041%
Unified	**Separate**	1.6M	**0.010%**	**0.014%**

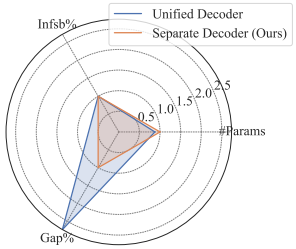

Figure 7: Effects of separate decoder.

F.3 Discussion of the unified model architecture

Recall that CaR employs a unified encoder shared across the construction and refinement decoders. As demonstrated in Table 2, this architecture significantly enhances performance. To further assess the impact of the unified encoder under varying constraint complexities, we present comprehensive results in Table 5 for *Medium* TSPTW-50, *Hard* TSPTW-50, and *Medium* TSPTW-100. Results indicate that as problem complexity increases, evidenced by higher infeasibility rates (e.g., 37.270% in *Hard* TSPTW-50) and greater optimality gaps (e.g., 10.931% in TSPTW-100), the performance improvements attributed to the unified encoder become more pronounced. This suggests that shared representations facilitate better generalization and constraint handling in more challenging scenarios.

We further evaluate this design on CVRPBLTW-100, where using separate encoders reduces the optimality gap slightly from -2.448% to -2.544% but increases model parameters by 1.7× compared to the unified encoder version of CaR (2.9M vs. 1.7M). To balance performance and computational efficiency, we advocate for the unified encoder design.

Moreover, we explore the unification of the encoder and decoder components. As shown in Table 6 and Figure 7, results on TSPTW-50 indicate that employing a separate decoder increases the number of model parameters by 1.1×, yet it further reduces the optimality gap. Therefore, this paper adopts an architecture with a unified encoder and separate decoders for each module.

F.4 Additional results for the pattern of CaR's refinement

During inference, we follow [9] to augment each instance 8×, generating eight initial solutions for refinement. Figure 8 shows four representative refinement trajectories on a TSPTW instance to illustrate the diversity in initial solutions and their subsequent refinement.

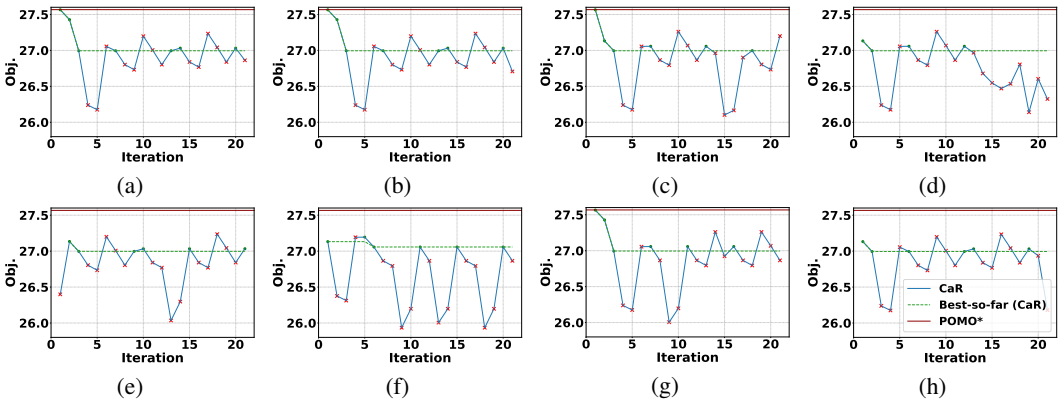

Figure 8: CaR's refinement on one TSPTW instance. Green dots and red crosses denote the feasible and infeasible solutions, respectively. Green dashed lines mark the best-so-far feasible objective; the red line indicates the best objective with data augmentation. CaR's refinement process is shown over time, with $t = 0$ representing the initially constructed solution.

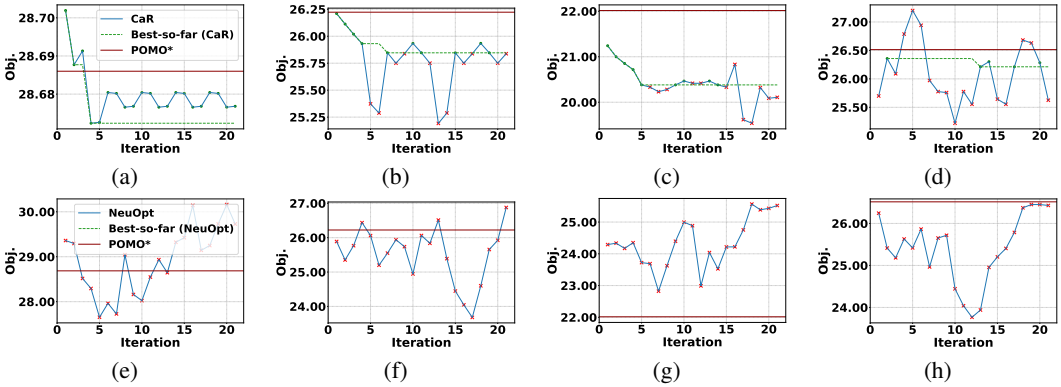

Figure 9: Refinement efficiency of CaR (*top*) and NeuOpt (*bottom*) on typical TSPTW-50 instances.

Moreover, to better understand how CaR refines solutions for feasibility and optimality, we also visualize the objective trajectories on other randomly generated TSPTW-50 instances in Figure 9. CaR dynamically navigates both feasible and infeasible regions, adjusting its focus based on the current solution state. For example, in the instance of Figure 9 (d), CaR begins with a near-feasible solution ($t = 0$), achieves feasibility in one step ($t = 1$), explores infeasible regions ($t = 2$–12), and ultimately improves optimality ($t = 13$). In contrast, in the instance of Figure 9 (c), the construction already yields a feasible, high-quality solution, and refinement continues to enhance optimality efficiently. These patterns illustrate CaR's ability to balance feasibility and optimality by exploring both feasible and infeasible spaces. By comparison, NeuOpt fails to escape infeasible regions within limited steps, highlighting CaR's advantage in constraint-aware refinement.

F.5 Effects of the joint training framework.

We evaluate our joint training framework against simple construction-improvement combinations. As shown in Table 7, CaR consistently outperforms these baselines. Although the construction quality of pretrained PIP and CaR-PIP is similar (results not shown), their refinement performance differs markedly, indicating that CaR's stronger initializations that are easier to refine and light refinement together drive rapid performance gains.

F.6 Effects of $\mathcal{L}_{\text{DIV}}$ and $\mathcal{L}_{\text{SL}}$.

We further ablate the diversity loss and the supervised loss for the construction module in Eq. (4) and Eq. (5), respectively. Figure 10 shows that diverse initial solutions aid refinement, and supervision

Table 7: Effects of joint training on TSPTW-50 with fixed runtime.

Joint train	Construction	Improvement	Gap↓	Infsb%↓
×	Random	LKH-3	0.011%	60.66%
×	Random	Pretrained NeuOpt*	/	100.00%
×	Random	Trainable NeuOpt*	/	100.00%
×	Pretrained PIP	LKH-3	**0.003%**	0.20%
×	Pretrained PIP	Pretrained NeuOpt*	0.134%	0.79%
×	Pretrained PIP	Trainable NeuOpt*	0.172%	2.59%
✓ (CaR)	Trainable PIP	Trainable NeuOpt*	0.005%	**0.00%**

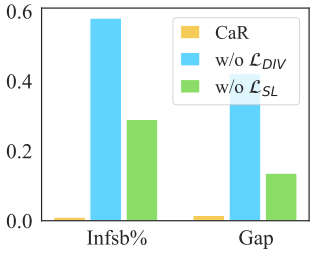

Figure 10: $\mathcal{L}_{\text{DIV}}$ and $\mathcal{L}_{\text{SL}}$.

with improved solutions strengthens cross-paradigm collaboration. Additional CVRP results in Appendix F.2 show that $\mathcal{L}_{\text{DIV}}$ remains effective with multi-starting.

F.7 Sensitivity to random seeds and statistical significance

Sensitivity to random seeds. To assess the robustness of CaR, we evaluate its performance under five different random seeds, i.e., 2023, 2024, 2025, 2026 and 2027, during inference. As shown in Table 8, the standard deviations are small, indicating that CaR's performance is stable with respect to random initialization.

Statistical significance. Figure 11 illustrates the distribution of optimality gaps for NeuOpt, CaR-POMO, and CaR-PIP, corresponding to the results on TSPTW-50 in Table 1. We additionally conduct significance tests, revealing that all pairwise differences are statistically significant ($p < 0.001$). This suggest that CaR delivers a significant performance improvement over the representative baselines.

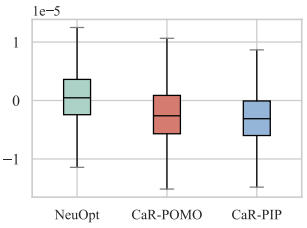

Figure 11: Boxplot of diverse methods on TSPTW-50.

Table 8: Standard deviation of the results in Table 1 under five different random seeds.

Problem	Method	n=50 Gap ↓	n=50 Infsb% ↓	n=100 Gap ↓	n=100 Infsb% ↓
TSPTW	CaR-POMO ($T_{\text{R}} = 20$)	0.014% ± 0.000%	0.00% ± 0.01%	0.404% ± 0.001%	2.30% ± 0.03%
	CaR-PIP ($T_{\text{R}} = 20$)	0.005% ± 0.000%	0.00% ± 0.01%	0.144% ± 0.001%	0.02% ± 0.01%
CVRPBLTW	CaR (k-opt) ($T_{\text{R}} = 20$)	2.147% ± 0.029%	0.00% ± 0.00%	-1.730% ± 0.004%	0.00% ± 0.00%
	CaR (R&R) ($T_{\text{R}} = 20$)	0.465% ± 0.007%	0.00% ± 0.00%	-2.449% ± 0.010%	0.00% ± 0.00%