DEPOT-CLOSED MULTI-COMPONENT CONSTRUCTION FOR NEURAL VEHICLE ROUTING

Anonymous authors
Paper under double-blind review

ABSTRACT

Most neural constructive solvers for the vehicle routing problem (VRP) use route-by-route construction, extending one route until completion before starting the next. This commits route membership early and hinders global coordination across routes. We propose multi-component construction, which maintains many route components simultaneously and merges them in an arbitrary order. This removes the depot-return cue that route-by-route construction obtains from the remaining capacity; to compensate, we introduce an interpretation in which every component is treated as an implicitly depot-closed route. Under this depot-closed interpretation, every intermediate state of standard CVRP construction is a complete feasible solution, and the exact cost reduction of a merge is the Clarke–Wright saving. The neural policy combines this CW-saving signal with the evolving component state to learn *what to connect* and *when to connect*. A policy trained only on CVRP100 outperforms the reported results of representative neural solvers on CVRP100–500 with greedy inference and, reused for ruin-and-reconstruct, performs strongly at all evaluated sizes up to CVRP1000. In a zero-shot Constraint Tightness evaluation with capacities from $C = 10$ to 500, it outperforms the reported neural solvers at every capacity. Controlled analyses show that robustness persists without CW grounding and point to learned route-closing behavior as a plausible contributor to the tight-regime degradation of learned route-by-route solvers.

1 INTRODUCTION

Neural combinatorial optimization (NCO) solves combinatorial optimization problems with neural networks (Bello et al., 2017; Bengio et al., 2021; Cappart et al., 2021; Wu et al., 2026), and routing problems such as the traveling salesman problem (TSP) and the capacitated vehicle routing problem (CVRP) are among its most representative applications. Neural constructive solvers, which generate solutions directly and sequentially, have been particularly successful because they combine fast inference with high solution quality (Vinyals et al., 2015; Nazari et al., 2018; Kool et al., 2019; Kwon et al., 2020; Xin et al., 2021; Kim et al., 2022; Drakulic et al., 2023; Luo et al., 2023; Jiang et al., 2026; Fang et al., 2024; Huang et al., 2025).

Most constructive CVRP solvers extend a single active route: they choose the next customer from the current position, return to the depot when the capacity requires it, and then start the next route. In this *route-by-route construction*, the remaining capacity is not only feasibility information but also a cue of how far the current route has grown; because it is synchronized with construction progress, the policy can use it to judge the progress of the route and the timing of the depot return. At the same time, growing one route first means committing its customers one after another before the formation of the other routes can be observed, which limits the global coordination of multiple routes even in standard CVRP.

We therefore revisit route-by-route construction itself. In *multi-component construction*, each customer initially forms a singleton component, and each step connects the endpoints of two different components by one edge. Many route components are maintained simultaneously and grown in an arbitrary order. This makes the commitment order of assignment decisions spanning multiple routes flexible and lets the route partition form while the entire component configuration is observed. This freedom comes at a price: since components grow asynchronously, the unique route progression is

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

lost, and it is no longer obvious how a partial component should relate to the depot as a CVRP route. In our preliminary experiments, a multi-component policy without depot-related grounding formed petal-like route geometry from the coordinates to some extent but closed routes to the depot poorly, suggesting that customer-to-customer geometry alone is insufficient for depot closure.

We address this by interpreting each component $P = (v_1, \ldots, v_m)$ not as an open path but as the *implicitly depot-closed route* $0 \to v_1 \to \cdots \to v_m \to 0$, which we call the *depot-closed interpretation*. It is closely related to the route-merging construction of the classical Clarke–Wright (CW) savings algorithm (Clarke & Wright, 1964), which starts from depot-closed singleton routes and repeatedly merges them. Under this interpretation, every component is a route with a routing cost at that moment, and every intermediate state of standard CVRP construction is a complete feasible solution. Moreover, the exact objective improvement of merging endpoints i and j is the CW saving $s_{ij} = d(0, i) + d(0, j) - d(i, j)$, which our neural policy uses together with the current component state to evaluate candidate merges. Since the order in which edges are committed is not unique even for the same final routes, the policy must also decide when each merge should be executed. We first learn merge connectivity consistent with reference solutions by supervised learning and then improve the entire merge sequence by reinforcement learning under the terminal routing objective, extending learning from *what to connect* to *when to connect* (Figure 1).

We evaluate solution quality on standard CVRP, size generalization from CVRP100 to CVRP200–1000, and zero-shot robustness across capacities on the Constraint Tightness benchmark (Luo et al., 2025b). The final policy outperforms the reported results of representative neural solvers on CVRP100–500 with greedy inference and, reused as a reconstruction operator, performs strongly at all sizes up to CVRP1000, where its greedy inference falls short of LEHD. A single $C = 50$ model outperforms the reported neural solvers at every capacity from $C = 10$ to 500; controls in Section 4.5 point to learned route closing as a plausible contributor to the tight-regime failures of learned route-by-route solvers. Our main contributions are as follows.

1. We propose multi-component construction for NCO, which operates on multiple route components simultaneously in place of route-by-route construction and makes the commitment order of route-assignment decisions spanning multiple routes flexible.
2. To compensate for the depot-return cue lost by moving to multiple components, we introduce the depot-closed interpretation and feed the CW saving, the exact cost reduction of a merge in standard CVRP, into the neural action score as a grounding signal. The depot-closed interpretation keeps the routing cost of every component evaluable, while the CW-saving bias ties neural action selection to the routing objective.
3. We demonstrate strong size and capacity generalization from a single CVRP100 policy, reuse it for reconstruction, and motivate learned route closing as a plausible contributor to tight-regime failures.

2 Related Work

Neural constructive solvers for routing. Since Pointer Networks, NCO for routing has widely adopted autoregressive construction, which selects the next customer given the current partial solution (Vinyals et al., 2015; Bello et al., 2017; Nazari et al., 2018; Kool et al., 2019; Kwon et al., 2020; Xin et al., 2021). For CVRP, demands and the remaining capacity enter the decoder context or node features, and solutions are generated route by route. Subsequent work has improved solution quality and generalization through symmetry, state representation, model architecture, and multi-task learning (Kim et al., 2022; Drakulic et al., 2023; Luo et al., 2023; Jiang et al., 2026; Fang et al., 2024; Huang et al., 2025; Gao et al., 2024; Xiao et al., 2025b; Zhou et al., 2023; Berto et al., 2025); multi-task approaches include POMO-MTL (Liu et al., 2024) and MVMoE (Zhou et al., 2024). Other studies revisit construction itself. JAMPR starts and extends multiple routes simultaneously and selects route–customer pairs as joint actions (Falkner & Schmidt-Thieme, 2020), and L2C-Insert inserts an unvisited customer at any valid position of the current partial solution (Luo et al., 2025a). The TSP decoder of DIFUSCO also greedily joins multiple path fragments (Sun & Yang, 2023), but its edge scores come from a static heatmap computed before decoding and are not recomputed from an evolving component state. In contrast, our policy uses the current route-component state as its state and selects each merge as a sequential action.

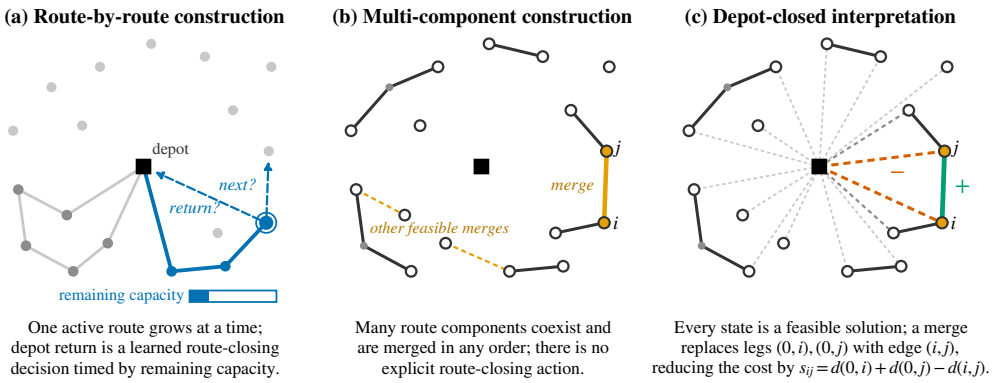


Figure 1: Route-by-route versus multi-component construction. (a) Typical learned route-by-route construction grows one active route at a time; remaining capacity provides both feasibility information and a route-progress cue for the learned depot-return decision. (b) Multi-component construction maintains multiple route components simultaneously and selects merges between endpoints of different components; route closure is not an explicit policy action. (c) Under the depot-closed interpretation, every component is treated as an implicitly closed route, so every intermediate state is a feasible CVRP solution and merging endpoints i and j reduces the total cost by exactly the Clarke–Wright saving $s_{ij} = d(0,i) + d(0,j) - d(i,j)$. Here 0 denotes the depot.

Reconstruction and inference-time refinement. Ruin-and-reconstruct reapplies a trained constructive solver to subproblems or partial solutions obtained by ruining part of a solution and is widely used for large-scale generalization and inference-time refinement (Luo et al., 2023; Jiang et al., 2026; Li et al., 2025; Luo et al., 2025a). Local rewriting, learned local search, test-time adaptation, and subproblem selection have also been used for refinement in neural routing (Chen & Tian, 2019; Ma et al., 2021; Hottung et al., 2022; Li et al., 2021; Ma et al., 2023). In our method, construction from singletons and reconstruction from ruined route fragments share the same depot-closed component representation, so the same policy serves both (Section 3.6).

Clarke–Wright savings algorithm. CW (Clarke & Wright, 1964) initializes every customer i as a singleton route $0 \to i \to 0$ and greedily merges routes whose endpoints i, j have a large saving s_{ij} as long as feasibility constraints such as capacity are satisfied; many studies extend the saving criterion or the merging rule (Altınel & Öncan, 2005; Stanojević et al., 2013). We call the classical algorithm itself *Classical CW* when it serves as a baseline. Its standard *parallel* variant, used unless otherwise noted, repeatedly performs feasible merges in decreasing order of saving until no merge is possible. Its *sequential* variant extends one active route from both ends in decreasing order of feasible saving and closes it to start the next route only when no feasible extension remains. Although this extension rule differs from the one-ended appending of autoregressive neural solvers, it belongs to the same route-by-route scheme in that only one active route grows at a time; we use it as a control in Section 4.5. Learning-based enhancements also exist: Legrand et al. (2021) use neural edge-selection probabilities to adjust the priority order within the saving list. Rather than learning the ranking rule of the CW heuristic, we incorporate the CW saving as an exact action-local objective signal into a neural merge policy operating on a dynamic multi-component state (Sections 3.3 and 4.4).

Generalization under size, distribution, and constraint shifts. Generalization of neural routing solvers has mainly been studied for size extrapolation and shifts in the node-coordinate distribution (Kim et al., 2022; Drakulic et al., 2023; Zhou et al., 2023; Luo et al., 2023; Fang et al., 2024; Berto et al., 2025; Xiao et al., 2025a). The Constraint Tightness study (Luo et al., 2025b) shows that merely changing the CVRP capacity substantially changes the solution structure and the required solving strategy, and that models trained with a fixed capacity can overfit to that value; its problem-similarity analysis reports that CVRP100 with $C = 10$ is close to OVRP, whereas with $C = 500$ it approaches TSP. A capacity shift is thus a structural shift in route count, route geometry, and effective construction strategy, and we use this benchmark in addition to size extrapolation.

3 DEPOT-CLOSED MULTI-COMPONENT CONSTRUCTION

3.1 MULTI-COMPONENT DECISION PROCESS

Let 0 denote the depot, $V = \{1, \ldots, N\}$ the customers with locations $\mathbf{x}_i \in \mathbb{R}^2$ and demands q_i, and Q the vehicle capacity. Standard CVRP minimizes total length over capacity-feasible depot-to-depot routes, without fixing the vehicle count (Dantzig & Ramser, 1959; Toth & Vigo, 2014). We represent state t by route components $\mathcal{P}_t = \{P_1, \ldots, P_{K_t}\}$, initially one singleton per customer; each component is a path whose two endpoints alone may participate in the next merge. An action a_t joins endpoints i, j of different components P_a, P_b. With $D(P) = \sum_{i \in P} q_i$, the merge is feasible if and only if $D(P_a) + D(P_b) \leq Q$, and $\pi_\theta(a \mid \mathcal{P}_t)$ selects among feasible endpoint pairs. Each merge reduces K_t by one. Construction ends when no feasible component pair remains, so route partition and route count emerge jointly from the merge sequence.

This process loses two cues of route-by-route construction: the relation between each partial path and the depot is not explicit, and there is no unique order that determines which feasible merge to commit now. We first address the former.

3.2 GIVING ROUTE SEMANTICS TO EACH COMPONENT

Route-by-route construction couples remaining capacity to the progress of one active route; with asynchronous components, this single-route lifecycle disappears. To compensate for this loss, we introduce the depot-closed interpretation: each component $P = (v_1, \ldots, v_m)$ is treated as the implicitly depot-closed route $0 \to v_1 \to \cdots \to v_m \to 0$. With the Euclidean distance $d(\cdot, \cdot)$ and the internal path length $\mathrm{len}(P)$, its cost is

$$\mathrm{cost}(P) = d(0, v_1) + \mathrm{len}(P) + d(v_m, 0), \tag{1}$$

and the cost of a state is $\mathrm{cost}(\mathcal{P}_t) = \sum_{P \in \mathcal{P}_t} \mathrm{cost}(P)$. An important consequence is that, in standard CVRP, every intermediate state is already a complete feasible solution: the initial singleton state contains every customer exactly once, and merging two capacity-feasible components preserves feasibility once each component is closed at the depot. Construction therefore proceeds as a sequence of transformations $\mathcal{P}_t \to \mathcal{P}_{t+1}$ between feasible solutions, and every partial path has a meaning in terms of the routing objective at every step.

3.3 GROUNDING EACH MERGE IN THE ROUTING OBJECTIVE

Merging the endpoints i, j of depot-closed components removes the depot legs $i \to 0$ and $0 \to j$ and adds the edge $i \to j$. The objective improvement of a single merge is therefore

$$\mathrm{cost}(\mathcal{P}_t) - \mathrm{cost}(\mathcal{P}_{t+1}) = d(0, i) + d(0, j) - d(i, j) = s_{ij}, \tag{2}$$

which is exactly the Clarke–Wright saving. We provide s_{ij} to the policy explicitly because the multi-component representation alone offers no direct point of contact, from the learning perspective, between the depot-closed interpretation and action selection. In supervised learning in particular, the teacher only indicates expert-compatible merges, and the distance change associated with depot closure is never observed; without a depot-aware signal, even a merge rule that imitates customer-to-customer geometry while ignoring depot legs can satisfy the teacher. Since s_{ij} is exactly the cost difference of a merge under the depot-closed interpretation, it is a natural signal for grounding this interpretation in the policy.

Since $\mathrm{cost}(\mathcal{P}_0) = 2\sum_i d(0, i)$, the final cost decomposes as $\mathrm{cost}(\mathcal{P}_T) = \mathrm{cost}(\mathcal{P}_0) - \sum_{t=0}^{T-1} s_{a_t}$, where s_{a_t} is the saving of merge a_t. The saving thus measures the objective improvement of each merge at that moment, but always selecting the largest current saving is not sufficient: a merge fixes the edges inside the component and changes which endpoint pairs remain selectable, so the policy must also consider which feasible merges will remain in the future. The action score therefore combines two kinds of information. The CW saving exactly gives the immediate improvement of a candidate merge, whereas a learned score conditioned on the current component state supplies the longer-horizon information about future options that the saving lacks, that is, *when to connect*.

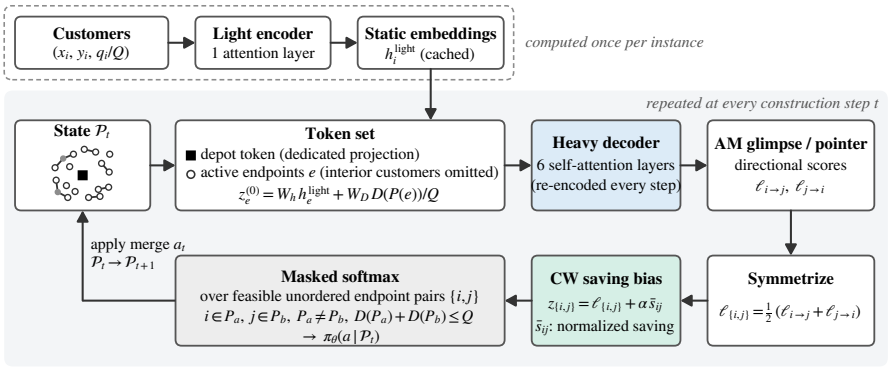


Figure 2: Policy architecture. Static customer features are encoded once by a light encoder. At each construction step, the depot token and active endpoints, together with their component loads, are re-encoded by the heavy decoder. An AM-based glimpse/pointer produces directional scores, which are symmetrized into unordered pair scores, augmented by the normalized CW-saving bias, and normalized by a masked softmax over capacity-feasible endpoint pairs. Applying the selected merge produces the next component state. For clarity, the graph-context term used in the full implementation is omitted; the endpoint-token equation is the conceptual form of Section 3.4.

3.4 Global neural reasoning over unordered merges

To capture longer-horizon effects, the learned score conditions on the full current state. We adapt LEHD's Transformer-based light-encoder/heavy-decoder architecture (Luo et al., 2023; Vaswani et al., 2017); its heavy decoder re-encodes the current token set at every construction step (Figure 2).

Static customer information and dynamic component information. The light encoder computes h_i^{light} once from the static feature $(x_i, y_i, q_i/Q)$. The component load $D(P)/Q$ changes with every merge and is therefore handled by the heavy decoder, which at each step receives only the depot and the active endpoints; internal customers are excluded because they can no longer participate in merges. Conceptually, endpoint e is represented as

$$z_e^{(0)} = W_h h_e^{\text{light}} + W_D\, D(P(e))/Q, \tag{3}$$

where $P(e)$ is the component containing e. Endpoints share one projection, whereas the depot is a special token with its own projection. The depot is not a merge candidate, but self-attention lets every endpoint refer to both the depot and the other components.

Scoring an unordered endpoint pair. With endpoint i as query and j as destination, the Attention Model (AM) glimpse/pointer (Kool et al., 2019) gives a directional score $\ell_{i\to j}$. Because merge $\{i, j\}$ is unordered, we symmetrize the two directional scores before the softmax, $\ell_{\{i,j\}} = (\ell_{i\to j} + \ell_{j\to i})/2$. Since CW saving is pair-specific, its normalized value enters the pair score as a logit bias rather than a node feature:

$$z_{\{i,j\}} = \ell_{\{i,j\}} + \alpha\, \bar{s}_{ij}, \tag{4}$$

where $\bar{s}_{ij}$ is the normalized CW saving and α its strength. A single softmax over all feasible unordered endpoint pairs yields $\pi_\theta(a \mid \mathcal{P}_t)$. The architecture thus separates static customer information in the light encoder, dynamic component state in the heavy decoder, pair compatibility in the pointer, and the objective change in the pair logit.

3.5 Learning what to connect and when to connect

Freeing both what and when to connect from the beginning makes learning difficult because correct connectivity and good merge ordering must be explored simultaneously. We therefore first learn what to connect stably by supervised learning (SL), gradually transfer merge-order freedom to the policy, and finally optimize the entire merge sequence against the terminal routing cost by reinforcement learning (RL) in S3. In parallel, the training states move from intermediate states of ordinary

construction (S0–S1) to repair states obtained by partially ruining completed solutions (S2–S3). The curriculum thus has two axes (Table 6 in Appendix A): decision freedom expands from what to when, and the state distribution expands from construction to reconstruction.

In the SL stages S0–S2, all currently feasible, unselected merge actions corresponding to customer–customer edges in the reference solution are treated as correct. Denoting this set by $A^+(\mathcal{P})$, we impose no unique teacher order and maximize the probability mass assigned to the correct set, $\sum_{a\in A^+(\mathcal{P})} \pi_\theta(a \mid \mathcal{P})$. The stages differ in how the next state is generated. In *shuffle-guided* trajectories (S0), the state is updated by a merge chosen uniformly at random from $A^+(\mathcal{P})$, so the reference-solution edges are committed in a random order independent of the policy. In *policy-guided* trajectories (S1–S2), the action distribution of the policy is restricted to $A^+(\mathcal{P})$; the trajectory remains expert-compatible, but its ordering depends on the current policy. Moving from the random ordering of S0 to the policy-induced ordering of S1 also reduces the mismatch between artificially generated expert-compatible states and the states actually visited by the policy. In all stages, we additionally use *route-subset sampling*: a subset of routes is extracted from a source solution to form a training subproblem, on which the stage-specific initial state is generated. This reduces the training problem size and exposes the policy to different effective problem sizes (Section 4.3).

3.6 The same policy for construction and reconstruction

To improve solution quality beyond initial construction, we also use the same policy for iterative ruin-and-reconstruct (RRC), which refines an incumbent solution by repeatedly ruining part of it and reconstructing it from the remaining fragments. We use *polar ruin*, inspired by the polar-angle-based destruction strategy used for CVRP in L2C-Insert (Luo et al., 2025a): the polar angle viewed from the depot defines the neighborhood metric. In each iteration, a customer chosen uniformly at random serves as the ruin center, customers with polar angles close to that of the center are returned to singleton components, and the undestroyed route fragments are kept. The same operator generates the initial states of S2–S3. Regarding each remaining fragment as a depot-closed component, the set of fragments and singletons is a state of exactly the same type as $\mathcal{P}$ in Sections 3.1–3.2. Construction and reconstruction therefore differ only in their initial state; the feasible actions, heavy decoder, pair scorer, CW grounding, and policy π_θ are all shared, and no reconstruction-specific solver is needed.

4 Experiments

4.1 Experimental setup

All main models are trained on uniform Euclidean CVRP100 with capacity $C = 50$; following the Constraint Tightness benchmark, we write C for the capacity value in the experiments. We follow the protocol of LEHD (Luo et al., 2023): training uses 1,000,000 CVRP100 instances solved by Hybrid Genetic Search (HGS) (Vidal, 2022), and the test sets contain 10,000 CVRP100 instances and 128 instances each of CVRP200/500/1000 with capacities 50/80/100/250, with LKH3 (Helsgaun, 2017) solutions as references. S0–S3 are successive training stages of the same policy, and S3 is the final model; training settings and runtime results are given in Appendix A. The same policy is applied to all sizes without additional training. Unless otherwise noted, we report the mean per-instance gap $\frac{1}{M}\sum_{m=1}^{M} 100\,(L_m - L_m^*)/L_m^*$ over the M test instances; a negative gap means that the solution is shorter than the reference. Reported values are transcribed from the tables of the cited papers (Source column), and Mean denotes the unweighted average over the four sizes.

4.2 Standard size generalization

Greedy inference. Table 1 applies the policy trained on CVRP100 to CVRP100–1000 with greedy inference; Classical CW uses the parallel variant. As the curriculum proceeds, the mean gap improves monotonically ($5.642 \to 3.288 \to 3.129 \to 2.700$), and the final S3 outperforms the compared reported greedy results on CVRP100, 200, and 500. On CVRP1000, however, greedy inference falls short of LEHD, and the gap even worsens slightly from 7.164% at S1 to 7.495%/7.544% at S2/S3, leaving room for improvement under extreme scale shifts. Classical CW also merges depot-closed routes but selects each merge greedily by saving, and its gap is larger than that of S3 at every size: depot-closed route merging and the CW saving alone cannot explain the performance of S3,

Table 1: Greedy inference on standard CVRP (mean per-instance gap to LKH3, %).

Method	CVRP100	CVRP200	CVRP500	CVRP1000	Mean	Source
BQ	2.726	2.972	3.248	5.892	3.710	ReLD Table 3 / BQ original
LEHD	3.648	3.312	3.178	**4.912**	3.763	LEHD Table 1; ReLD Table 3
MnLP	3.509	3.206	2.928	6.195	3.960	MnLP Table 1
INViT	9.964	12.160	13.772	15.548	12.861	MnLP Table 1
DGL	6.441	8.368	9.356	16.727	10.223	MnLP Table 1
Classical CW	5.479	7.310	6.525	11.999	7.828	This work
S0	3.504	4.417	4.682	9.963	5.642	This work
S1	1.732	1.990	2.266	7.164	3.288	This work
S2	1.383	1.589	2.049	7.495	3.129	This work
S3	**1.019**	**0.828**	**1.407**	7.544	**2.700**	This work

Table 2: Enhanced inference on standard CVRP (mean per-instance gap to LKH3, %). Negative values indicate solutions shorter than the LKH3 references.

Method	Inference	CVRP100	CVRP200	CVRP500	CVRP1000	Mean	Source
MDAM	bs50	2.211	4.304	10.498	27.814	11.207	ReLD Table 3
POMO	aug×8	1.004	3.403	11.135	110.632	31.544	ReLD Table 3
ELG	aug×8	1.207	2.553	5.472	10.760	4.998	ReLD Table 3
ReLD	aug×8	0.960	1.654	2.975	6.757	3.087	ReLD Table 3
BQ	bs16	0.611	1.141	2.991	7.784	3.132	LEHD Table 1
BQ	bs16, later rerun	1.020	0.940	1.010	2.880	1.463	DRHG Table 3
LEHD	RRC100	0.272	0.217	0.546	2.370	0.851	LEHD Table 1
LEHD	RRC1000	−0.112	−0.383	−0.347	0.921	0.020	LEHD Table 1
MnLP	RRC1000	0.012	−0.255	−0.257	1.148	0.162	MnLP Table 1
DRHG	T=1000	−0.020	−0.160	1.310	7.660	2.198	DRHG Table 3
S0	RRC1000	0.597	0.400	0.658	3.125	1.195	This work
S1	RRC1000	0.018	0.144	0.620	1.957	0.685	This work
S2	RRC1000	−0.199	−0.605	−1.102	−0.014	−0.480	This work
S3	RRC1000	**−0.216**	**−0.722**	**−1.179**	**−0.272**	**−0.597**	This work

and the learned, state-dependent merge selection provides an additional gain. The greedy inference of S3 also runs at a speed comparable to LEHD (Appendix A.2).

Enhanced inference. Table 2 compares the reported settings that use additional inference-time computation (full results in Table 11). For S0–S3, the same policy serves as the reconstruction operator of RRC with polar ruin and 1,000 reconstruction iterations (RRC1000). Since the other methods use different procedures (augmentation, beam search, or RRC with other ruin operators), the table compares reported enhanced-inference settings rather than a common inference algorithm. RRC resolves the degradation on CVRP1000 that remained under greedy inference: already S2 outperforms the LKH3 references at all four sizes, and S3 outperforms LEHD RRC1000, the best among the compared reconstruction methods, at all four sizes. BQ bs16 appears twice because different sources report different rerun values.

4.3 Analysis of size generalization

Effect of route-subset sampling. Table 3 changes only the presence of route-subset sampling for S2 and S3 while keeping the rest of the training setup fixed. Route-subset sampling sacrifices about 0.1–0.2 pp of the in-distribution gap on CVRP100 but substantially improves extrapolation to CVRP200–1000; on CVRP500, for example, the gap of S3 decreases from 2.746% to 1.407%. This suggests that experiencing different effective problem sizes during training, not the architecture alone, is an important factor for size generalization.

Table 3: Effect of route-subset sampling (greedy, gap %).

Stage	Subset	100	200	500	1000	Mean
S2	No	1.217	1.833	3.615	9.952	4.154
S2	Yes	1.383	1.589	2.049	7.495	3.129
S3	No	0.889	1.159	2.746	8.057	3.213
S3	Yes	1.019	0.828	1.407	7.544	2.700

Table 4: CW-grounding bootstrap on CVRP100 (gap %).

Training	Inference	Gap
no CW grounding	no CW bias	5.822
no CW grounding	+CW bias	3.883
CW fine-tuning	+CW bias	**3.787**

Table 5: Zero-shot generalization across constraint tightness on CVRP100 (aggregate gap to HGS, %). All neural models are trained with $C = 50$. Baseline values are taken from the Constraint Tightness benchmark (Luo et al., 2025b); rows below the middle rule are ours. Mean is recomputed from the seven displayed values for all methods.

Model	C=10	C=50	C=100	C=200	C=300	C=400	C=500	Mean
AM	45.16	7.66	12.85	21.63	26.26	25.35	27.23	23.73
POMO	20.26	3.66	11.17	22.45	29.56	30.80	34.12	21.72
MDAM	7.47	5.38	12.47	21.13	24.31	22.55	23.91	16.75
BQ	18.02	3.25	3.48	4.53	6.53	6.54	8.16	7.22
LEHD	36.56	4.22	4.87	4.18	5.78	4.92	6.82	9.62
ELG	10.44	5.24	10.91	18.70	22.81	21.84	24.05	16.28
INViT	17.44	7.85	11.12	14.36	15.86	12.61	13.71	13.28
POMO-MTL	13.62	4.50	8.20	10.40	14.12	13.44	15.57	11.41
MVMoE	10.72	5.06	10.52	17.63	21.80	21.86	16.00	14.80
Classical CW (parallel)	4.31	5.95	9.92	11.31	10.78	9.03	8.19	8.50
Classical CW (sequential)	5.59	16.44	23.64	24.02	21.31	18.57	14.68	17.75
No-CW base	5.91	6.28	7.94	8.04	8.11	7.45	7.56	7.33
S0	4.81	4.00	5.80	6.01	5.90	4.75	4.86	5.16
S1	**4.17**	2.23	3.22	3.98	4.62	4.13	4.60	3.85
S2	5.49	1.87	2.75	3.17	3.78	3.50	4.05	3.52
S3	5.42	**1.51**	**2.13**	**2.55**	**3.04**	**2.92**	**3.40**	**3.00**

Target-size calibration of α. The saving coefficient α in Eq. (4) is a scalar hyperparameter rather than a learnable parameter, which allows the strength of the CW-saving signal to be calibrated without retraining the neural policy. In the main results, it is selected on validation data for each target size, since the optimal value differs across sizes. Fixing the CVRP100-selected $\alpha = 100$ for all sizes changes the mean gap of S3 only from 2.700% to 2.725% and is even slightly better on CVRP500 (Table 10 in Appendix B). Target-size-specific calibration is therefore not essential.

4.4 Disentangling neural reasoning and Clarke–Wright grounding

To separate the learned score from the analytic CW saving, besides the gap between Classical CW and S3 (Table 1), we decompose the CW-grounding bootstrap that forms S0 (Table 4): a base policy is trained by shuffle-guided SL with $\alpha = 0$, α is selected on validation data, and the policy is then fine-tuned on the CW-grounded logits. This diagnostic comparison, which does not use the final S0 checkpoint of Table 1, separates the effect of adding the CW signal post hoc from that of adapting the policy to the resulting action landscape. Adding the signal post hoc improves the gap from 5.822% to 3.883%, so making the objective change of each candidate merge explicit contributes substantially on its own; fine-tuning further improves it to 3.787%, showing that the learned score also adapts to the CW-grounded action space. Together with the large gap between Classical CW and S3, both the analytic local objective signal and the learned state-dependent merge scheduling contribute to solution quality. Section 4.5 shows, however, that basic constraint robustness appears even before CW grounding is introduced.

4.5 GENERALIZATION ACROSS CONSTRAINT TIGHTNESS

Following the Constraint Tightness benchmark (Luo et al., 2025b), we fix the problem size to CVRP100 and evaluate $C \in \{10, 50, 100, 200, 300, 400, 500\}$ zero-shot: all neural models are trained with $C = 50$, and α is not re-tuned per capacity (each model keeps the value selected at $C = 50$). Following the original protocol, we report the aggregate gap $100\,(\sum_m L_m - \sum_m L_m^*)/\sum_m L_m^*$ with HGS as the reference.

S3 outperforms the reported neural solvers at every capacity, with a mean gap of 3.00% versus 7.22% for BQ and 9.62% for LEHD (Table 5). The difference is largest in the tight regime: from $C = 50$ to $C = 10$, LEHD degrades from 4.22% to 36.56% and BQ from 3.25% to 18.02%, whereas S3 increases from 1.51% to 5.42%. The stage-wise improvement is not monotone at every capacity; the best value at $C = 10$ is 4.17% (S1).

To diagnose the tight-regime failure mode—not to establish robustness from one capacity alone—we focus on $C = 10$, where the separation is sharpest; Table 5 provides the full sweep. At $C = 10$, every reported learned route-by-route solver has a gap of at least 7.47%, whereas both non-learned CW controls and all our multi-component policies are at most 5.91%. No single attribute explains this split. CW saving does not: parallel CW reaches 4.31%, while the No-CW base uses no saving and reaches 5.91%. Route-by-route construction does not either: sequential CW grows one route at a time yet reaches 5.59%. Nor does learning alone, since the No-CW base and S0–S3 are learned. The sharpest contrast is LEHD versus the No-CW base: both use the light-encoder/heavy-decoder design, supervised training on HGS solutions, and no saving signal, yet they reach 36.56% and 5.91% at $C = 10$. Among the evaluated methods, the combination of learning and route-by-route construction coincides exactly with the poorly performing group.

This conjunction has a natural mechanistic interpretation: learned route closing. In learned route-by-route solvers, the depot-return decision is learned under the state progression of the training capacity, where remaining capacity can also cue route progress; a large capacity shift can therefore create a mismatch at test time. Our policy has no route-closing action, while sequential CW closes a route deterministically when no feasible extension remains. We therefore hypothesize that learned route closing contributes to the tight-regime degradation. The full sweep is qualitatively consistent: several learned route-by-route baselines deteriorate again for $C > 50$ as routes lengthen, although BQ and LEHD remain comparatively stable. The controls do not isolate this cause: the same poorly performing group is also the set of route-by-route methods without an analytic saving signal, and other capacity-dependent learned behaviors may contribute.

Separately, the CW controls show that construction scheme strongly affects solution quality. Under the same saving rule, sequential CW is 6.49–13.72 percentage points worse than parallel CW for $C = 50$–500 (mean 17.75% vs. 8.50%), but only 1.28 points worse at $C = 10$ (5.59% vs. 4.31%). Thus route-by-route construction alone does not explain the severe tight-regime degradation, while construction scheme still has a large effect when routes become long.

5 CONCLUSION

We proposed multi-component construction, which changes the basic unit of neural route construction from a single active route to multiple simultaneous route components. Interpreting every component as a depot-closed route makes every intermediate state of standard CVRP a complete feasible solution and makes the objective improvement of a merge equal to the Clarke–Wright saving; the neural policy combines this signal with the evolving component state to learn which components to connect and when. A single CVRP100 policy performed strongly under size and capacity shifts. Controls point to learned route closing as a plausible contributor to tight-regime failures, while separate CW controls show that construction scheme strongly affects solution quality.

Limitations. Greedy inference on CVRP1000 falls short of LEHD. Edges once fixed inside a component cannot be changed directly during ordinary construction, so major corrections require re-partitioning by ruin-and-reconstruct. We focused on additive total-distance CVRP with a variable number of routes; prescribed route counts and non-additive route-level objectives such as min-max routing are left for future work. Nevertheless, our results suggest that multi-component construction is a promising alternative paradigm for neural route construction.

AI USE STATEMENT

Generative AI tools, primarily ChatGPT and Claude, were used as research assistants throughout this work. Their use included technical discussion and brainstorming during the development of the method, literature search and reference verification, assistance with software implementation and debugging, discussion of experimental design and analysis of experimental results, and assistance in drafting and preparing the manuscript.

In particular, the tools were used to discuss alternative modeling and algorithmic choices, review and generate portions of experimental code and command-line procedures, help diagnose implementation and experimental issues, and assist in organizing and interpreting experimental results. They were also used to improve the presentation of the work, including structuring the manuscript, assisting with English translation and editing, and assisting with LaTeX preparation.

AI-generated suggestions and outputs were reviewed and validated by the authors as appropriate. Final decisions regarding the research questions, proposed method, experimental protocol, interpretation of results, and conclusions were made by the authors. The authors take full responsibility for the correctness and final content of this work, including content produced with the assistance of generative AI.

REPRODUCIBILITY STATEMENT

We provide the details required to reproduce the proposed method and experiments throughout the paper and appendix. The multi-component decision process, depot-closed representation, policy architecture and action scoring, training curriculum, and reconstruction procedure are specified in Sections 3.1–3.6. The datasets, evaluation protocol, reference solutions, evaluation metrics, and main experimental settings are described in Section 4.1 and the corresponding experimental subsections (Sections 4.2–4.5). Detailed model and training configurations, including architecture hyperparameters, stage-wise optimization settings, random seeds, inference settings, and the runtime measurement protocol, are provided in Appendix A.

REFERENCES

İ. Kuban Altınel and Temel Öncan. A new enhancement of the Clarke and Wright savings heuristic for the capacitated vehicle routing problem. *Journal of the Operational Research Society*, 56(8): 954–961, 2005.

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.

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.

Federico Berto, Chuanbo Hua, Nayeli Gast Zepeda, André Hottung, Niels A. Wouda, Leon Lan, Junyoung Park, Kevin Tierney, and Jinkyoo Park. RouteFinder: Towards foundation models for vehicle routing problems. *Transactions on Machine Learning Research*, 2025(9):1–19, 2025. ISSN 2835-8856.

Quentin Cappart, Didier Chételat, Elias B. Khalil, Andrea Lodi, Christopher Morris, and Petar Veličković. Combinatorial optimization and reasoning with graph neural networks. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI), Survey Track*, 2021.

Xinyun Chen and Yuandong Tian. Learning to perform local rewriting for combinatorial optimization. In *Advances in Neural Information Processing Systems*, 2019.

G. Clarke and J. W. Wright. Scheduling of vehicles from a central depot to a number of delivery points. *Operations Research*, 12(4):568–581, 1964.

G. B. Dantzig and J. H. Ramser. The truck dispatching problem. *Management Science*, 6(1):80–91, 1959.

Darko Drakulic, Sofia Michel, Florian Mai, Arnaud Sors, and Jean-Marc Andreoli. BQ-NCO: Bisimulation quotienting for efficient neural combinatorial optimization. In *Advances in Neural Information Processing Systems*, 2023.

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

Han Fang, Zhihao Song, Paul Weng, and Yutong Ban. INViT: A generalizable routing problem solver with invariant nested view transformer. In *International Conference on Machine Learning*, 2024.

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-Third International Joint Conference on Artificial Intelligence*, pp. 6914–6922, 2024. doi: 10.24963/ijcai.2024/764.

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

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

Ziwei Huang, Jianan Zhou, Zhiguang Cao, and Yixin Xu. Rethinking light decoder-based solvers for vehicle routing problems. In *International Conference on Learning Representations*, 2025.

Xia Jiang, Yaoxin Wu, Yew-Soon Ong, and Yingqian Zhang. Learning with foresight: Enhancing neural routing policy via multi-node lookahead prediction. In *Proceedings of the Thirty-Fifth International Joint Conference on Artificial Intelligence*, pp. 6166–6174, 2026. doi: 10.24963/ijcai.2026/686.

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

Wouter Kool, Herke van Hoof, and Max Welling. Attention, learn to solve routing problems! In *International Conference on Learning Representations*, 2019.

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*, 2020.

Clément Legrand, Luca Accorsi, Diego Cattaruzza, Laetitia Jourdan, Marie-Eléonore Kessaci, and Daniele Vigo. Using machine learning to enhance Clarke and Wright heuristic. In *ROADEF 2021*, 2021. Conference communication.

Ke Li, Fei Liu, Zhenkun Wang, and Qingfu Zhang. Destroy and repair using hyper-graphs for routing. *Proceedings of the AAAI Conference on Artificial Intelligence*, 39(17):18341–18349, 2025. doi: 10.1609/aaai.v39i17.34018.

Sirui Li, Zhongxia Yan, and Cathy Wu. Learning to delegate for large-scale vehicle routing. In *Advances in Neural Information Processing Systems*, 2021.

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*, pp. 1898–1908, 2024. doi: 10.1145/3637528.3672040.

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.

Fu Luo, Xi Lin, Mengyuan Zhong, Fei Liu, Zhenkun Wang, Jianyong Sun, and Qingfu Zhang. Learning to insert for constructive neural vehicle routing solver. In *Advances in Neural Information Processing Systems*, volume 38, 2025a. doi: 10.52202/085713-2918.

Fu Luo, Yaoxin Wu, Zhi Zheng, and Zhenkun Wang. Rethinking neural combinatorial optimization for vehicle routing problems with different constraint tightness degrees. In *Advances in Neural Information Processing Systems*, volume 38, 2025b. doi: 10.52202/085713-4240.

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*, 2021.

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*, 2023.

Mohammadreza Nazari, Afshin Oroojlooy, Lawrence Snyder, and Martin Takáč. Reinforcement learning for solving the vehicle routing problem. In *Advances in Neural Information Processing Systems*, 2018.

Milan Stanojević, Bogdana Stanojević, and Mirko Vujošević. Enhanced savings calculation and its applications for solving capacitated vehicle routing problem. *Applied Mathematics and Computation*, 219(20):10302–10312, 2013. doi: 10.1016/j.amc.2013.04.002.

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

Paolo Toth and Daniele Vigo (eds.). *Vehicle Routing: Problems, Methods, and Applications*. SIAM, 2nd edition, 2014.

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*, 2017.

Thibaut Vidal. Hybrid genetic search for the CVRP: Open-source implementation and SWAP* neighborhood. *Computers & Operations Research*, 140:105643, 2022.

Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. Pointer networks. In *Advances in Neural Information Processing Systems*, 2015.

Xuan Wu, Lijie Wen, Yubin Xiao, Chunguo Wu, Yuesong Wu, Chaoyu Yu, Douglas L. Maskell, You Zhou, and Di Wang. Neural combinatorial optimization algorithms for solving vehicle routing problems: A comprehensive survey with perspectives. *IEEE Transactions on Neural Networks and Learning Systems*, 2026. doi: 10.1109/TNNLS.2026.3713193.

Yubin Xiao, Di Wang, Xuan Wu, Yuesong Wu, Boyang Li, Wei Du, Liupu Wang, and You Zhou. Improving generalization of neural vehicle routing problem solvers through the lens of model architecture. *Neural Networks*, 187:107380, 2025a. doi: 10.1016/j.neunet.2025.107380.

Yubin Xiao, Yuesong Wu, Rui Cao, Di Wang, Zhiguang Cao, Xuan Wu, Peng Zhao, Yuanshu Li, You Zhou, and Yuan Jiang. DGL: Dynamic global-local information aggregation for scalable VRP generalization with self-improvement learning. In *Proceedings of the Thirty-Fourth International Joint Conference on Artificial Intelligence*, pp. 8669–8677, 2025b. doi: 10.24963/ijcai.2025/964.

Liang Xin, Wen Song, Zhiguang Cao, and Jie Zhang. Multi-decoder attention model with embedding glimpse for solving vehicle routing problems. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2021.

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*, 2023.

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 *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 61804–61824, 2024.

A REPRODUCIBILITY AND RUNTIME

A.1 MODEL AND TRAINING CONFIGURATION

Table 6: Training curriculum. All stages train the same policy, starting from the checkpoint of the preceding stage.

Stage	Trajectory; initial state	Objective	Role
S0	shuffle-guided; construction prefix	set-valued SL	Shuffling suppresses dependence on a particular merge ordering; focuses on valid merge selection (what to connect).
S1	policy-guided; construction prefix	set-valued SL	Learns what to connect on expert-compatible, policy-generated construction trajectories.
S2	policy-guided; ruined solution	set-valued SL	Learns what to connect on expert-compatible, policy-generated repair trajectories.
S3	free policy rollout; ruined solution	terminal route-cost RL	Removes the teacher constraint and jointly learns what and when to connect.

All stages use the same network architecture. The static customer feature $(x, y, q/Q)$ is mapped to an embedding dimension of 128 and encoded once by a one-layer light encoder. The component state, which changes during construction, is re-encoded by a six-layer heavy decoder. Multi-head attention uses 8 heads with a per-head Q/K/V dimension of 16 and a feed-forward hidden dimension of 512, and we use the pointer mechanism and graph context of the Attention Model (AM) (Kool et al., 2019). The model has 1.56M parameters (Table 7).

Table 7: Model configuration.

Item	Setting
Training problem	CVRP100, $C = 50$
Training / validation data	1,000,000 HGS-solved instances / 1,000 instances
Embedding dimension	128
Light encoder	1 attention layer
Heavy decoder	6 attention layers
Attention	8 heads, Q/K/V dimension 16 per head
Feed-forward hidden dimension	512
Pointer / context	AM pointer mechanism; graph context
Parameters	1,562,752
Saving normalization	instance-wise maximum
Random seed	22

Training follows the S0–S3 curriculum, and each stage inherits the validation-selected checkpoint of the preceding stage. S0–S2 use supervised learning with Adam, and S3 uses REINFORCE with a rollout baseline. S0–S2 use centered-logit L_2 regularization ($\lambda = 3 \times 10^{-4}$). The rollout baseline of S3 is initialized from the S2 policy and updated when the candidate policy improves significantly on a 1,000-instance challenge set. Table 8 lists the stage-specific settings.

For polar ruin, a customer is selected uniformly at random as the ruin center. The destroy size is sampled from $\{4, \ldots, \min(100, n)\}$, and the customers with the closest polar angles to the center are removed. At test time, greedy inference performs a single construction pass from singleton components. RRC1000 uses the same policy with 1,000 reconstruction iterations; for a problem of size n, its destroy-size range is $\{4, \ldots, n\}$, and the reported runs use seed 4567.

Table 8: Stage-specific training settings. "constr. prefix" denotes a (shuffled or policy-generated) construction prefix; the S3 batch of 512 is processed in micro-batches of 256.

Setting	S0	S1	S2	S3
Learning objective	set-valued SL	set-valued SL	set-valued SL	REINFORCE
State generation	shuffle-guided	policy-guided	policy-guided	free rollout
Initial state	constr. prefix	constr. prefix	polar ruin	polar ruin
Route-subset sampling	Yes	Yes	Yes	Yes
Optimizer	Adam	Adam	Adam	Adam
Learning rate	10^{-4}	10^{-4}	10^{-4}	10^{-5}
LR decay factor	0.9	0.9	0.9	1.0
Training batch size	512	512	512	512 (micro 256)
Gradient clipping	None	None	5	1
Training α	72	85	95	100
Centered-logit L_2	3×10^{-4}	3×10^{-4}	3×10^{-4}	–
Ruin metric / range	–	–	polar / 100	polar / 100

For standard size generalization, the inference-time saving coefficient is selected on validation data separately for each training stage and target size. For the final S3 model used in the headline results, the selected values are $\alpha = 100, 100, 105, 200$ for CVRP100/200/500/1000, respectively. The fixed-α ablation uses the CVRP100-selected value $\alpha = 100$ for all four sizes. In the Constraint Tightness evaluation, each model uses a single α selected at the training capacity $C = 50$ and keeps it fixed across all target capacities; no target-capacity-specific calibration is performed.

A.2 Inference runtime

Runtime is measured on a single NVIDIA RTX PRO 6000 Blackwell Max-Q Workstation Edition GPU. For direct same-environment comparison, we rerun LEHD greedy inference on the same test instances as S3. We report the wall-clock time measured by each evaluation driver over its inference region, after model and dataset loading and before result serialization, divided by the number of evaluated instances. For both S3 and the LEHD greedy rerun, the evaluation batch sizes are 1000 for CVRP100 and 128 for CVRP200/500/1000. All same-environment measurements use the same machine and software environment.

Table 9: Inference runtime on the same RTX PRO 6000 system (wall-clock seconds per instance).

Method	Inference	CVRP100	CVRP200	CVRP500	CVRP1000
LEHD (rerun)	greedy	0.0012	0.0116	0.0628	0.3845
S3	greedy	0.0024	0.0156	0.0611	0.3115
S3	RRC1000	1.176	8.354	37.783	204.096

Greedy inference is in the same runtime regime as LEHD: S3 is slower on CVRP100 and CVRP200, nearly identical on CVRP500, and faster on CVRP1000. RRC1000 is substantially more expensive because it performs 1,000 destroy–repair iterations; Table 9 therefore reports its absolute inference cost rather than treating the iteration count as a compute-matched budget.

We also attempted a same-environment rerun of LEHD RRC1000. Unlike the greedy rerun, which reproduces the published LEHD greedy quality closely (3.636% versus 3.648% on CVRP100), the RRC rerun did not reproduce the published RRC solution quality in our environment. We therefore do not use its timing as a valid LEHD RRC runtime measurement. For context only, the original LEHD study reports total CVRP RRC1000 runtimes of 2.8 h, 11.3 min, 2 h, and 8 h over the full CVRP100/200/500/1000 test sets (10,000 and 128 instances), corresponding to roughly 1.0, 5.3, 56, and 225 s per instance, respectively, on a single NVIDIA GeForce RTX 3090 GPU (Luo et al., 2023). These published values were obtained under different hardware and implementation conditions and are not used for strict runtime ranking against S3.

Figure 3 characterizes the compute–quality trade-off within S3 itself using the saved trajectory of the same polar-RRC runs. The x-axis is wall-clock time per instance, reconstructed from the measured cumulative RRC time at each iteration plus the fixed per-run overhead so that the RRC1000 endpoint coincides with Table 9. Because the saved trajectory contains the mean incumbent cost rather than the per-instance gaps, the y-axis uses the ratio-of-means gap to the LKH3 references rather than the mean per-instance gap used in Tables 1 and 2. The curves show diminishing returns overall: substantial improvements are obtained in the first 10–100 reconstruction iterations, although CVRP1000 continues to improve markedly up to 1,000 iterations. RRC1000 is the high-compute end of this anytime trajectory rather than a required operating point.

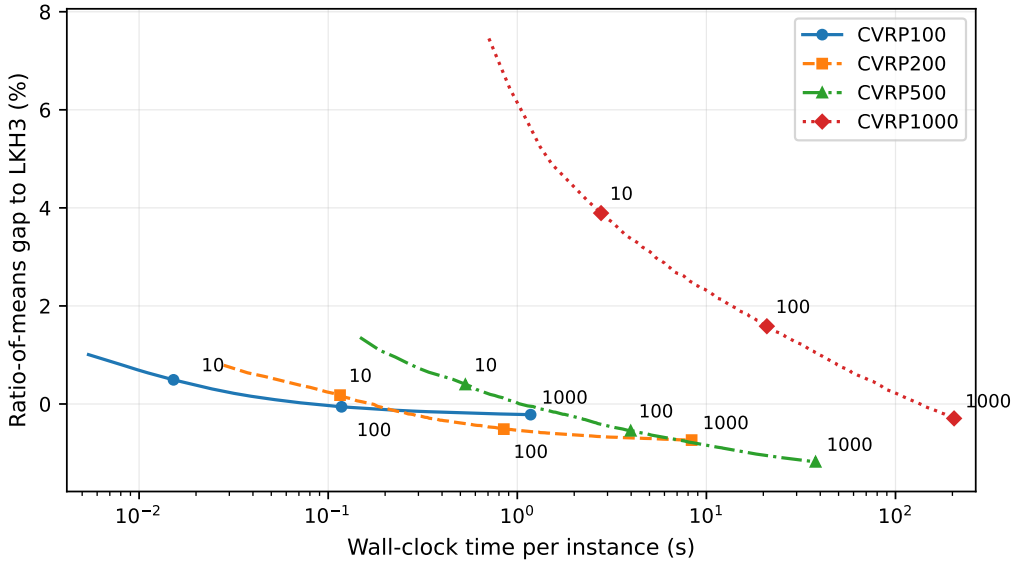


Figure 3: Anytime quality–runtime trade-off of S3 with polar RRC. The x-axis shows wall-clock time per instance on a logarithmic scale; markers are labeled by reconstruction iteration (10, 100, and 1,000). The y-axis is the ratio-of-means gap to the LKH3 references computed from the saved incumbent-cost trajectory. Lower is better.

Runtime environment: Python 3.12.7; PyTorch 2.11.0+cu130; CUDA 13.0; CPU x86_64 (44 PyTorch threads); GPU NVIDIA RTX PRO 6000 Blackwell Max-Q Workstation Edition.

B Additional Results

Table 10: Target-size calibration ablation for S3 (greedy, mean per-instance gap to LKH3, %). Fixed α uses the CVRP100-selected value $\alpha = 100$ for all sizes.

Inference	CVRP100	CVRP200	CVRP500	CVRP1000	Mean
size-calibrated α	1.019	0.828	1.407	7.544	2.700
fixed α	1.019	0.828	1.358	7.694	2.725

Table 11: Full enhanced-inference results on standard CVRP, including all reported RRC budgets of LEHD and MnLP (mean per-instance gap to LKH3, %).

Method	Inference	CVRP100	CVRP200	CVRP500	CVRP1000	Mean	Source
MDAM	bs50	2.211	4.304	10.498	27.814	11.207	ReLD Table 3
POMO	aug×8	1.004	3.403	11.135	110.632	31.544	ReLD Table 3
ELG	aug×8	1.207	2.553	5.472	10.760	4.998	ReLD Table 3
ReLD	aug×8	0.960	1.654	2.975	6.757	3.087	ReLD Table 3
BQ	bs16	0.611	1.141	2.991	7.784	3.132	LEHD Table 1
BQ	bs16, later rerun	1.020	0.940	1.010	2.880	1.463	DRHG Table 3
LEHD	RRC50	0.535	0.515	0.930	2.814	1.199	LEHD Table 1
LEHD	RRC100	0.272	0.217	0.546	2.370	0.851	LEHD Table 1
LEHD	RRC300	0.029	−0.146	0.045	1.582	0.378	LEHD Table 1
LEHD	RRC500	−0.044	−0.246	−0.107	1.270	0.218	LEHD Table 1
LEHD	RRC1000	−0.112	−0.383	−0.347	0.921	0.020	LEHD Table 1
MnLP	RRC100	0.471	0.363	0.627	2.859	1.080	MnLP Table 1
MnLP	RRC500	0.094	−0.136	0.007	1.624	0.397	MnLP Table 1
MnLP	RRC1000	0.012	−0.255	−0.257	1.148	0.162	MnLP Table 1
DRHG	T=1000	−0.020	−0.160	1.310	7.660	2.198	DRHG Table 3
S0	RRC1000	0.597	0.400	0.658	3.125	1.195	This work
S1	RRC1000	0.018	0.144	0.620	1.957	0.685	This work
S2	RRC1000	−0.199	−0.605	−1.102	−0.014	−0.480	This work
S3	RRC1000	**−0.216**	**−0.722**	**−1.179**	**−0.272**	**−0.597**	This work