BASIN-GUIDED REINFORCEMENT LEARNING FOR MULTI-TASK VEHICLE ROUTING PROBLEMS

Anonymous authors
Paper under double-blind review

ABSTRACT

Multitask neural constructive solvers for vehicle routing are efficient but often lag behind classical solvers in solution quality. Combining neural construction with classical refinement can balance solution quality and efficiency. We propose a basin-guided reinforcement learning framework in which a neural solver generates intermediate solutions that are converted into feasible solutions and refined through bounded local moves. The framework builds on a simple insight: distinct candidates can reach the same refined outcome, so construction-level diversity does not necessarily translate into diversity after post-processing. This many-to-one mapping induces post-processing basins that guide learning. Alongside refined solution costs, training uses two complementary signals: a Refined-Edge Prior (REP) learns customer adjacencies from refined solutions to guide construction, while Refined-Outcome Entropy (ROE) encourages rare refined outcomes at both solution and route levels. During inference, candidates are generated by the neural network and post-processed in parallel. Optional route recombination further exploits diversity across refined candidates, forming the full pipeline. Across 16 VRP variants, the full pipeline reduces macro-average gaps to PyVRP with a 60-second per-instance budget by 69.2–81.7% relative to RouteFinder, PoMtVRS, CCL, MoSES, and RL-RFCS with the same post-processing stages, at mean end-to-end latencies of 0.11 and 0.56 seconds per instance for $n = 50$ and 100, respectively. After fine-tuning on randomly generated 200-customer instances, it achieves an average gap of -0.34% on CVRP200 to the same PyVRP reference at approximately 2.6 seconds per instance with a larger inference budget.

1 INTRODUCTION

Vehicle Routing Problems (VRPs) are a fundamental class of Combinatorial Optimization Problems (COPs) with significant practical importance (Vidal et al., 2020). Since finding optimal solutions to many VRPs is NP-hard (Lenstra & Kan, 1981), classical heuristic solvers (Wouda et al., 2024; Furnon & Perron, 2024) and Neural Combinatorial Optimization (Bello* et al., 2017) (NCO) solvers have emerged as two important solution paradigms. Classical solvers typically obtain high-quality solutions, whereas neural constructive solvers can generate solutions efficiently through neural-network inference. How to combine their complementary strengths to achieve a better trade-off between solution quality and computational efficiency remains an important research question.

PyVRP (Wouda et al., 2024), one of the strongest classical solvers for multiple VRP variants, uses iterated local search (ILS) to escape local optima through repeated perturbation and local-search moves.[1] Although each local-search pass is fast, successive perturbation-and-move cycles within a search trajectory are inherently sequential. Consequently, achieving high-quality solutions can require substantial overall search time. Multitask neural constructive solvers (Zhou et al., 2024; Berto et al., 2025; Yu et al., 2026) directly construct solutions without subsequent refinement moves, enabling efficient inference. However, the absence of post-construction refinement leaves construction errors uncorrected, contributing to a non-negligible solution-quality gap relative to PyVRP. Neural iterative solvers repeatedly apply a learned policy to improve an incumbent solution (Wu et al., 2022; Ma et al., 2023), whereas hybrid solvers combine neural construction with a learned refiner, as

[1] PyVRP refers to v0.13.4, the version used in all experiments; see `https://doi.org/10.5281/zenodo.20308425`.

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.

exemplified by CaR (Bi et al., 2026). In these cited methods, refinement introduces additional neural computation across improvement steps or candidates. Consequently, their inference costs grow with the refinement depth or candidate-pool size, making it expensive to refine numerous candidates extensively. In summary, the challenge is to satisfy two requirements simultaneously: efficiently generating promising candidates and refining them at scale with low computational cost.

To directly combine the complementary strengths of neural solvers and post-processing refinement, we propose a basin-guided framework for VRPs. We refine candidates generated by a neural constructive solver using a bounded number of local moves. With a fixed move budget and deterministic tie-breaking, distinct candidates can converge to the same refined outcome, thereby inducing empirical equivalence classes that we term post-processing basins. Our training design addresses two aspects: extracting structural information from refinement and promoting refined-outcome diversity. First, beyond solution costs, refined solutions provide structural information through their customer adjacencies. To exploit this information, we introduce a Refined-Edge Prior (REP) that learns customer adjacencies from refined endpoints through an auxiliary edge-prediction loss. This auxiliary loss is combined with the reinforcement learning objective to train the constructor. Second, POMO (Kwon et al., 2020) promotes diversity by enforcing different starting points, but this construction-level diversity does not account for whether distinct candidates converge to the same post-processing outcome. We therefore introduce Refined-Outcome Entropy (ROE), which assigns diversity credit to rare post-processing outcomes at the solution and route levels. During inference, the neural constructor generates multiple candidate solutions, which are refined in parallel using bounded local moves; together, these steps constitute our core pipeline. Optionally, our full pipeline further recombines routes (Subramanian et al., 2013) extracted from the refined candidates by solving a restricted set-partitioning problem with a specialized search-based solver.

Across 16 VRP variants, our framework achieves lower macro-average gaps than five neural baselines equipped with the same post-processing stages. Relative to PyVRP with a 60-second per-instance budget, our full pipeline achieves macro-average gaps of 0.059% and 0.087% for $n = 50$ and 100, respectively, at mean end-to-end latencies of 0.110 and 0.560 seconds per instance. After fine-tuning on randomly generated 200-customer instances, it achieves a gap of -0.340% on CVRP200 at 2.635 seconds per instance with a larger inference budget, matching the solution quality of HGS-CVRP (Vidal, 2022) with a 60-second budget.

To summarize, we make the following contributions: 1) We formulate deterministic post-processing as a many-to-one mapping that induces empirical basins over neural solutions, and present a post-processing basin-guided framework that unifies neural construction and bounded local refinement for VRPs; 2) We propose Refined-Outcome Entropy (ROE), which assigns diversity credit based on outcome frequencies after refinement. Unlike action-level entropy, ROE directly promotes complementary solutions and vehicle routes that remain diverse after post-processing; 3) We propose a Refined-Edge Prior (REP) that learns customer adjacencies from refined candidates through an auxiliary edge-prediction loss and uses its predicted edge scores to guide decoding. REP is trained jointly with the policy-gradient objective without requiring optimal labels; and 4) Extensive experiments across 16 VRP variants and multiple problem scales demonstrate improved solution quality, refined-outcome diversity, and quality-efficiency trade-offs over representative neural baselines and classical solvers.

2 RELATED WORK

Classical foundations of our pipeline. Our solution pipeline builds on three ideas with a long history in classical VRP algorithms. First, the refinement stage employs standard intra- and inter-route neighborhoods, including 2-opt, Relocate, Swap, Or-opt-2/3, and 2-opt*, which form the basis of many local-search-based VRP solvers (Wouda et al., 2024; Vidal et al., 2013). Second, our permutation-based representation and dynamic-programming splitting follow the classical route-first, cluster-second paradigm (Beasley, 1983; Prins, 2004). Third, recombining routes collected from candidate solutions through a restricted set-partitioning problem follows a long-standing route-pool post-optimization strategy in hybrid metaheuristics (Rochat & Taillard, 1995; Subramanian et al., 2013). Unlike classical pipelines, our framework incorporates bounded refinement into learning, while route recombination remains an inference-only extension.

Multitask neural construction. Recent multitask neural VRP solvers improve constructor-side modeling through unified architectures (Berto et al., 2025), expert specialization and cross-task knowledge sharing (Zhou et al., 2024; Yu et al., 2026), preference optimization (Meng et al., 2026), dynamic constraint-aware context modeling (Gui et al., 2026), and reuse of specialized basis policies (Pan et al., 2025). These advances primarily concern how the neural constructor represents and transfers knowledge across VRP variants. Our focus is complementary: we study how outcomes of deterministic downstream optimization can be converted into learning signals for the constructor.

Learning from solver feedback and solution diversity. Recent neural routing methods have increasingly coupled solution construction with downstream optimization. For example, CaR and LCP combine a neural constructor with learned refinement policies for feasibility restoration or solution improvement (Bi et al., 2026; Kim et al., 2021). In contrast, our framework treats deterministic post-processing, rather than a learned refiner, as the source of training feedback for the constructor. Closer to our setting, GaS and RL-RFCS train their constructors using costs returned by deterministic splitting procedures, but do not exploit structural or diversity signals in the resulting solutions (Wang et al., 2025; 2026). Structural information from solver solutions has also been distilled into supervised edge scores or candidate sets to guide subsequent search (Xin et al., 2021b); in contrast, our edge targets are derived online from the model's own refined candidates and require no optimal labels. Meanwhile, candidate diversity has been promoted through multi-start rollouts, entropy-regularized seed generation, complementary decoders, and implicitly specialized construction strategies within a single decoder (Kwon et al., 2020; Kim et al., 2021; Xin et al., 2021a; Hottung et al., 2025). These mechanisms operate before refinement, whereas our method assigns diversity credit to outcomes that remain distinct after deterministic post-processing.

3 PRELIMINARIES

Vehicle Routing Problems and RL. To validate the generality of the proposed framework, we consider a unified multi-task setting including 16 VRP variants (Berto et al., 2025). The VRPs in this study are defined on a complete graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$. The set $\mathcal{V}$ contains a depot node v_0 and n customer nodes $\{v_1, v_2, \cdots, v_n\}$. The i-th node is located in the 2D Euclidean plane, with coordinates (x_i, y_i). The travel cost from node v_i to node v_j is their Euclidean distance, denoted by c_{ij}. The Capacitated Vehicle Routing Problem (CVRP) is the base variant in this multi-task setting. In that case, each customer node has a demand q_i, and each vehicle has a capacity Q. Each route starts and ends at the depot and sequentially serves customers whose total demand does not exceed Q. Each customer must be visited exactly once. The objective is to minimize the total travel cost of all routes. Beyond capacity, each variant is characterized by four binary attributes: (1) open routes (O), (2) backhauls (B), (3) distance limits (L), and (4) time windows (TW). Enabling or disabling these attributes yields $2^4 = 16$ VRP variants, such as OVRP, VRPB, VRPL, and VRPTW. Hereafter, x denotes the full instance, including all problem attributes. We follow RouteFinder's attribute definitions; Appendix A.1 summarizes the constraints and the 16 variants. Neural constructive solvers typically use an encoder-decoder policy to generate solutions autoregressively and are trained with REINFORCE (Williams, 1992) to minimize expected solution cost. In our framework, the policy generates customer permutations, which are converted into feasible solutions and locally refined; training uses the resulting negative refined costs as rewards.

Local Search and Basins of Attraction. Starting from an initial solution, ILS repeatedly applies perturbation, local search, and an acceptance criterion (Lourenço et al., 2019). Perturbation modifies the current solution without necessarily reducing its cost, for example, by randomly cutting and reconnecting several route edges. This part of the ILS framework is used to escape the current local optimum. Local search means applying a set of local moves to the current solution until the solution cannot be further improved. Many solutions can be mapped to the same local optimum, forming a basin of attraction: $\mathcal{B}(s^*) = \{s \in S : \mathrm{LS}(s) = s^*\}$, where S is the original solution set and LS is a deterministic local search mapping. An acceptance criterion determines whether the resulting solution replaces the incumbent; a simple criterion accepts only solutions with lower costs. Perturbation and the acceptance rule work together to govern the diversity of the search process, while local search is used to optimize current solutions. Within each ILS trajectory, perturbation, local search, and acceptance are executed sequentially. In contrast, our framework generates and refines multiple candidates in parallel and promotes candidate diversity through RL training.

4 BASIN-GUIDED REINFORCEMENT LEARNING

As illustrated in Fig. 1, our framework trains a neural constructor using three signals from deterministic post-processing: negative refined costs as RL rewards, refined edges as weak supervision, and refined-outcome frequencies to promote diversity.

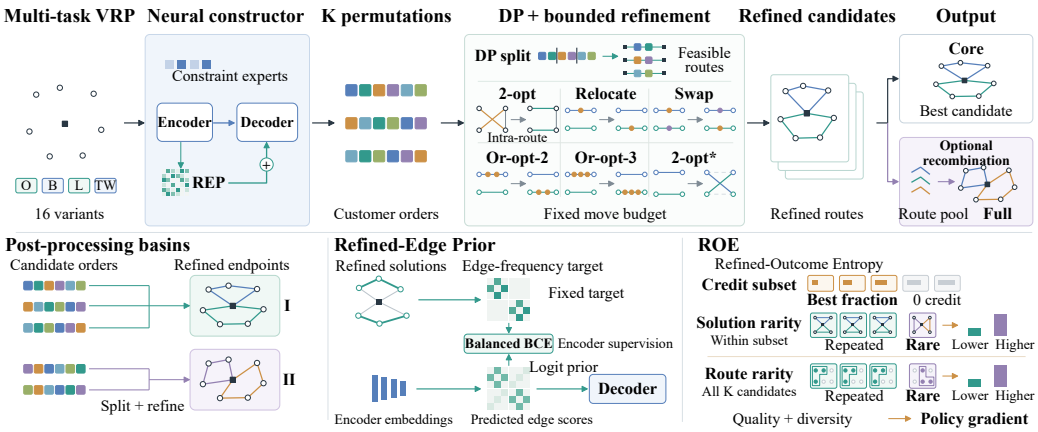

Figure 1: Basin-guided reinforcement learning. Top: the solution pipeline. Bottom: post-processing basins and the REP and ROE learning components.

4.1 THE OVERALL LEARNING FRAMEWORK

Permutation policy. For an instance x with customer set $V_x = \{1, \ldots, n\}$ and depot 0, the policy samples a customer permutation:

$$\tau = (\tau_1, \ldots, \tau_n), \qquad \pi_\theta(\tau \mid x) = \prod_{t=1}^{n} \pi_\theta(\tau_t \mid \tau_{<t}, x). \tag{1}$$

Each customer appears exactly once. Following RL-RFCS (Wang et al., 2026), dynamic programming optimally partitions this order into feasible routes, yielding $s = \mathrm{DP}(\tau)$ (Appendix A.2).

Local refinement. A deterministic refiner produces $\hat{s} = P_M(s)$ through at most M best-improving feasible cross-route moves from Relocate, Swap, Or-opt-2/3, and 2-opt*, with deterministic tie-breaking. For non-TW variants, capped intra-route 2-opt descent is also applied initially and after each accepted move. The result remains feasible with $C(\hat{s}) \le C(s)$. With fixed M and a fixed 2-opt cap, refinement costs $O(n^2)$; operator and schedule details are in Appendix A.3.

Post-processing basins. Let κ remove vehicle-order and valid route-reversal symmetries (Appendix A.1). For a fixed instance and budget, define the endpoint $z_\tau = \kappa(P_M(\mathrm{DP}(\tau)))$ and its basin $\mathcal{B}(z) = \{\tau : z_\tau = z\}$. Because refinement is bounded, the endpoint may still admit improving moves and need not be a local optimum. Distinct permutations in the same basin share the same refined cost but do not contribute distinct refined outcomes.

Inference pipelines. The core pipeline samples K_{test} permutations, applies DP and M_{test} refinement steps to each, and returns the best candidate. The full pipeline further exploits route complementarity across refined candidates. It pools their independently feasible routes, recomputes physical costs, and retains the lowest-cost route for each distinct customer set. The resulting restricted set-partitioning problem (Subramanian et al., 2013) selects routes of minimum total cost that cover each customer exactly once. Time-bounded search starts from the best refined candidate and cannot worsen its cost (Appendix A.6). Recombination is used only at inference and contributes no training rewards.

Learning from refinement. For K sampled permutations, training uses negative refined costs and their mean as the reward and baseline:

$$R(\tau_k) = -C(P_M(\mathrm{DP}(\tau_k))); \qquad b = \frac{1}{K}\sum_{k=1}^{K} R(\tau_k). \tag{2}$$

Beyond scalar rewards, refined solutions provide structural information through customer adjacencies, while their frequencies reveal outcome redundancy that costs alone do not capture. REP and ROE exploit these complementary signals to guide construction and promote refined-outcome diversity, respectively.

4.2 Refined Edge Prior

REP learns refined customer adjacencies to supervise node representations and guide construction. Our constructor combines a RouteFinder-style Transformer encoder (Berto et al., 2025), an RL-RFCS-style LSTM decoding context (Wang et al., 2026), and constraint-activated residual experts inspired by MVMoE (Zhou et al., 2024); architecture details are in Appendix A.4. Let $H = \mathrm{Enc}_\theta(x) = [h_0, \ldots, h_n]$, with $h_i \in \mathbb{R}^d$, and let $\ell^{\mathrm{base}}_{t,j}$ be the decoder's base logit for customer j given H and prefix $\tau_{<t}$.

Refined-edge targets. For each instance, the policy samples K permutations, which are mapped to refined solutions $\{\hat{s}_x^{(k)}\}_{k=1}^K$. For distinct customers $i, j \in V_x$, the binary adjacency entry $A^{(k)}_{x,ij}$ indicates whether j immediately follows i on the same route in $\hat{s}_x^{(k)}$; depot edges and connections between routes are excluded. For reversal-symmetric variants (CVRP and VRPL), both edge directions are included; other variants retain directed edges. Averaging over the K candidates gives the empirical refined-edge frequencies:

$$\bar{A}_x = \frac{1}{K}\sum_{k=1}^{K} A_x^{(k)}. \tag{3}$$

Predictions guide sampling before refinement; targets are formed afterward for training.

Representation-level supervision. For distinct customers $i, j \in V_x$, REP predicts

$$e_{x,ij} = \frac{(W_Q^E h_{x,i})^\top (W_K^E h_{x,j})}{\sqrt{d_E}}, \tag{4}$$

where $W_Q^E, W_K^E \in \mathbb{R}^{d_E \times d}$ are learnable projections. Because customer-to-customer edges are sparse, we use class-balanced binary cross-entropy over a batch $\mathcal{X}$:

$$\mathcal{L}_{\mathrm{REP}}(\mathcal{X}) = \mathrm{BCE}_{\mathrm{bal}}\left(\{\sigma(e_{x,ij})\}, \{\bar{A}_{x,ij}\}\right), \tag{5}$$

where the entries range over $x \in \mathcal{X}$ and distinct customers $i, j \in V_x$, and σ is the sigmoid function. Edge and non-edge terms are balanced across the entire batch (Appendix A.4). Targets are held constant; the loss updates the projections and shared encoder.

Decision prior. At each non-initial step, the current customer is $c_t = \tau_{t-1}$ and REP augments the construction logit:

$$\ell_{t,j} = \ell^{\mathrm{base}}_{t,j} + \alpha_E e_{x,c_t j}. \tag{6}$$

The learnable scale α_E is initialized to zero and is positive in all examined trained models. Logit clipping, visited-customer masking, and softmax yield the next-customer probabilities.

4.3 Entropy-Based Refined-Outcome Diversification

Refined-Outcome Entropy (ROE) encourages distinct refined outcomes at the solution and route levels. Solution-level credit promotes distinct endpoints for candidate selection, while route-level credit promotes complementary customer groupings for optional route recombination. For credit assignment, our implementation uses a subset $\mathcal{G}_x$ containing the top γ fraction of candidates ranked by refined reward, including ties at the cutoff ($\gamma = 0.25$ by default; Appendix A.5).

Solution-level entropy. Let $p_x^{\text{sol}}(z)$ be the fraction of candidates in $\mathcal{G}_x$ with canonical endpoint z. Each candidate in $\mathcal{G}_x$ with canonical endpoint $z_x^{(k)}$ receives its empirical surprisal:

$$u_{x,k}^{\text{sol}} = -\log p_x^{\text{sol}}(z_x^{(k)}). \tag{7}$$

The mean credit within $\mathcal{G}_x$ equals the Shannon entropy of p_x^{sol}, favoring rare endpoints within this subset.

Route-level entropy. Let $\mathcal{R}(\hat{s})$ represent each route by its unordered customer set, matching route-pool deduplication. The frequency $p_x^{\text{route}}(r)$ is the number of occurrences of customer set r divided by the total route occurrences across all K refined candidates. The route-level credit is

$$u_{x,k}^{\text{route}} = \sum_{r \in \mathcal{R}(\hat{s}_x^{(k)})} \frac{|r|}{n} \left[-\log p_x^{\text{route}}(r) \right]. \tag{8}$$

Here, $|r|$ counts customers in route r. This coverage-weighted surprisal favors uncommon customer sets; its candidate average generally differs from the Shannon entropy of p_x^{route}. Frequencies use all K candidates; diversity advantages are assigned only within $\mathcal{G}_x$.

RL objective. The quality advantage is $A_{x,k}^{Q} = R(\tau_x^{(k)}) - b_x$. For $g \in \{\text{sol}, \text{route}\}$, we center diversity credits within $\mathcal{G}_x$ and assign zero outside it to obtain $A_{x,k}^{g}$. Candidates in $\mathcal{G}_x$ with above-average credit receive positive diversity advantages, and those with below-average credit receive negative advantages. Each term is rescaled to match the per-instance root-mean-square (RMS) magnitude of A_x^{Q}, yielding $\widetilde{A}_{x,k}^{g}$; terms with no variation remain zero (Appendix A.5). The combined advantage and training loss for a batch $\mathcal{X}$ of B instances are

$$A_{x,k} = A_{x,k}^{Q} + \lambda_{\text{sol}} \widetilde{A}_{x,k}^{\text{sol}} + \lambda_{\text{route}} \widetilde{A}_{x,k}^{\text{route}}, \tag{9}$$

$$\mathcal{L}(\mathcal{X}) = -\frac{1}{BK} \sum_{x \in \mathcal{X}} \sum_{k=1}^{K} A_{x,k} \log \pi_\theta(\tau_x^{(k)} \mid x) + \lambda_E \mathcal{L}_{\text{REP}}(\mathcal{X}). \tag{10}$$

Here, λ_{sol}, λ_{route}, and λ_E weight the two diversity terms and REP supervision, respectively. Refined solutions, frequencies, candidate subsets, and advantages are stop-gradient quantities; ROE trains the constructor through REINFORCE.

5 EXPERIMENTS

We evaluate solution quality and efficiency, generalization to out-of-distribution (OOD) data and larger instances, and the contributions of individual components.

Datasets and Baselines. We evaluate on uniform datasets from RouteFinder (Berto et al., 2025), with 1,000 instances per variant and size, and three Omni-VRP OOD distributions at $n = 200$ (Gaussian Mixture $(2, 5)$, Rotation, and Explosion) (Zhou et al., 2023). We assess larger-scale solution quality on the 128 official Scale-Net (Liu et al., 2026) CVRP1000 instances. OR-Tools (Furnon & Perron, 2024) and PyVRP (Wouda et al., 2024) serve as general-purpose classical baselines, while HGS-CVRP (Vidal, 2022) is also used for CVRP-only comparisons. The 16-variant baselines are RouteFinder (Berto et al., 2025), PoMtVRS (Meng et al., 2026), CCL (Gui et al., 2026), MoSES (Pan et al., 2025), and RL-RFCS (Wang et al., 2026) with released checkpoints. We evaluate CaDA-based (Li et al., 2025) PoMtVRS/MoSES and ReLD-based (Huang et al., 2025) CCL. CVRP quality–latency comparisons additionally include the learned-refinement baseline CaR (Bi et al., 2026).

Experiment Details. We train separate $n = 50$ and $n = 100$ models from scratch for 100 epochs, sampling the 16 variants uniformly by batch using RouteFinder's data generator. Local refinement uses $M = 5$ for the first 90 epochs and $M = 12$ for the final 10; the $n = 200$ model is finetuned for 10 epochs from the $n = 100$ epoch-90 checkpoint. At inference, A8Sk denotes eight geometric augmentations with k sampled candidates per augmentation, so $K_{\text{test}} = 8 \times k$; the standard settings are A8S50 for $n = 50$ and A8S100 for $n = 100$. Experiments run in this work use an RTX 5090 GPU and Intel Xeon Platinum 8470Q CPUs; OOD baseline results are quoted from Omni-VRP. Dataset selection and training, inference, and baseline configurations are detailed in Appendix B.

Table 1: Solution quality and latency across 16 VRP variants. †: original inference pipeline; bold: best neural solution quality.

Method	Stages	$n = 50$		$n = 100$	
		Avg. Obj. (Gap)	Latency (s)	Avg. Obj. (Gap)	Latency (s)
OR-Tools@60s	–	11.350 (0.188%)	60.00	17.969 (1.487%)	60.00
PyVRP@60s	–	11.328 (0.000%)	60.00	17.696 (0.000%)	60.00
RouteFinder†	–	11.557 (2.099%)	0.117	18.236 (3.187%)	0.232
RouteFinder	M20	11.371 (0.392%)	0.150	17.854 (0.971%)	0.281
RouteFinder	DFS	11.358 (0.271%)	0.152	17.775 (0.477%)	0.300
PoMtVRS (CaDA)†	–	11.482 (1.441%)	0.123	18.076 (2.273%)	0.228
PoMtVRS (CaDA)	M20	11.364 (0.334%)	0.153	17.819 (0.743%)	0.275
PoMtVRS (CaDA)	DFS	11.357 (0.257%)	0.156	17.765 (0.391%)	0.292
CCL (ReLD)†	–	11.447 (1.129%)	0.225	18.032 (2.045%)	0.386
CCL (ReLD)	M20	11.357 (0.265%)	0.256	17.807 (0.685%)	0.434
CCL (ReLD)	DFS	11.349 (0.192%)	0.259	17.753 (0.337%)	0.451
MoSES (CaDA)†	–	11.496 (1.549%)	0.279	18.141 (2.692%)	0.460
MoSES (CaDA)	M20	11.366 (0.346%)	0.310	17.838 (0.886%)	0.508
MoSES (CaDA)	DFS	11.357 (0.266%)	0.312	17.772 (0.465%)	0.523
RL-RFCS†	DP	11.470 (1.273%)	0.059	18.115 (2.459%)	0.116
RL-RFCS	M20	11.369 (0.355%)	0.088	17.823 (0.735%)	0.164
RL-RFCS	DFS	11.361 (0.281%)	0.091	17.759 (0.332%)	0.199
Ours (Core)	M20	11.341 (0.117%)	0.105	17.776 (0.480%)	0.190
Ours (Full)	DFS	**11.335 (0.059%)**	0.110	**17.714 (0.087%)**	0.560

Evaluation Metrics. Unless otherwise specified, we report mean objectives (Avg. Obj.) and percentage gaps computed from each variant's mean objective relative to PyVRP@60s, both macro-averaged across the 16 variants. Latency is the mean end-to-end wall-clock time per instance, including all pipeline stages, with batch size 1 on a single GPU and CPU. Neural methods report measured end-to-end wall-clock latency. Classical-solver points are plotted at their per-instance search-time budgets; their actual wall-clock times are slightly longer.

5.1 Solution Quality and Efficiency

Table 1 compares quality and latency across 16 variants against OR-Tools, PyVRP, RouteFinder, PoMtVRS, CCL, MoSES, and RL-RFCS; Appendix C.1 provides per-variant results. M20 and DFS denote cumulative pipelines ending after local refinement with a 20-move budget and subsequent DFS route recombination, respectively. For the M20 and DFS comparisons, all neural methods use the same DP splitting and subsequent post-processing, with identical candidate counts and post-processing settings. DFS route recombination in Table 1 and Fig. 2 uses pools of up to 3,200 unique routes. In Fig. 2, TR denotes the CaR refinement budget.

Table 1 and Fig. 2 demonstrate our framework's solution quality and quality–latency tradeoffs. Across the 16 variants, our method achieves lower macro-average gaps than all five neural baselines at both M20 and DFS for $n = 50$ and $n = 100$. The full pipeline reaches gaps of 0.059% and 0.087% to PyVRP@60s at mean latencies of 0.110 and 0.560 seconds per instance, respectively, although its $n = 100$ latency is higher than that of the neural baselines with DFS. Fig. 2 shows further quality gains from larger sampling and refinement budgets and route recombination. With A8S800, M50, and DFS, our method achieves a gap of -0.056% on CVRP100 at 0.754 seconds per instance; after fine-tuning on randomly generated 200-customer instances, the corresponding configuration reaches -0.340% on CVRP200 at 2.635 seconds. The latter is comparable in solution quality to HGS-CVRP with a 60-second search budget (a gap of -0.331%), while requiring substantially less inference time under the reported hardware settings. On CVRP100, our method achieves lower gaps and latencies than the evaluated CaR configurations with refinement; unrefined CaR remains the fastest plotted configuration.

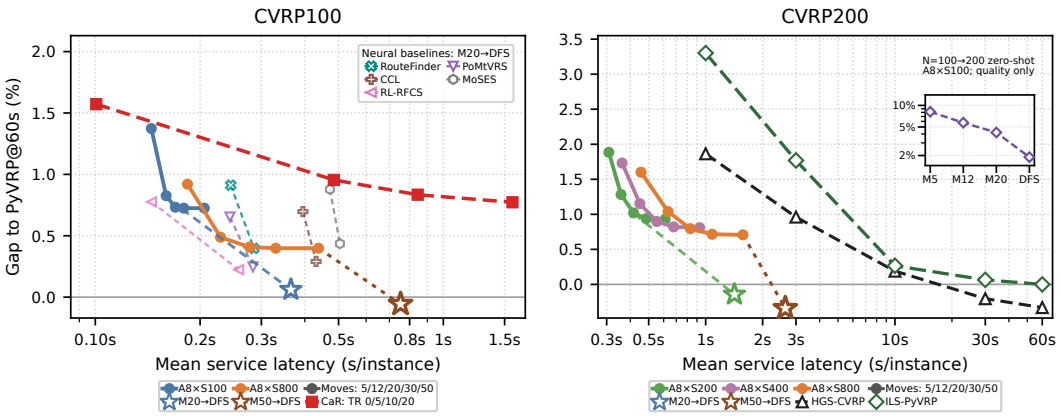

Figure 2: Quality–latency tradeoffs on CVRP100 (left) and CVRP200 (right). Solid curves vary the move budget; stars indicate route recombination. The inset shows zero-shot transfer from $n = 100$ to $n = 200$.

These results suggest that a multitask learning-based framework can reach the solution quality of a specialized, hand-designed solver, with a small numerical advantage over HGS-CVRP under the evaluated CVRP200 setting. Beyond this quality gain, the broader significance lies in the flexibility of the quality–efficiency tradeoff. The candidate count, local-refinement budget, and optional route recombination provide complementary ways to allocate computation at inference time. This allows the same trained constructor to support both low-latency operation and more computationally intensive configurations that pursue higher solution quality.

5.2 Evaluation on OOD Data and Larger Instances

Table 2 compares our method and Scale-Net on CVRP1000 with matched trajectory counts and post-processing. For the CVRP1000 comparison, both methods are adapted to the target scale. Our model is fine-tuned from its CVRP200 checkpoint using 84,000 online-generated CVRP1000 instances, whereas Scale-Net-1k uses the checkpoint released by the Scale-Net authors at epoch 5500, following 500 epochs of fine-tuning on six 1,000-node VRP variants. We use 16 trajectories following Scale-Net's original setting and expand to 800 with A8S100. Our method matches Scale-Net's quality, with slightly lower objectives after M20 and DFS at both budgets. Nevertheless, a substantial gap to PyVRP@60s remains for both methods, even at the larger budget. Scale-Net reports gaps relative to OR-Tools (Liu et al., 2026), highlighting the importance of interpreting results in the context of the reference solver and evaluation budget. Table 3 evaluates our method with A8S200 on the same CVRP200 OOD instances used by Omni-VRP, using the same CVRP200 checkpoint as the main curves in Fig. 2 (right). All gaps in Table 3 are recomputed from the displayed two-decimal mean objectives relative to HGS. Core uses M20; Full adds DFS with a pool of up to 3,200 unique routes. Both configurations of our method achieve lower gaps than Omni-VRP across all three distributions, and route recombination consistently improves solution quality. Our full pipeline also achieves lower gaps than LKH3 (Helsgaun, 2017) on Rotation and Explosion, while

Table 2: Objectives of our method and Scale-Net on 128 CVRP1000 instances.

PyVRP@60s reference: 27.234				
#Traj.	Method	Original	M20	DFS
16	Scale-Net-1k	31.833	29.352	29.351
16	Ours	–	**29.320**	**29.320**
800	Scale-Net-1k	31.204	28.994	28.937
800	Ours	–	**28.863**	**28.859**

Table 3: CVRP200 OOD objectives (HGS gaps). Baselines from Omni-VRP (Zhou et al., 2023).

Method	GM $(2, 5)$	Rotation	Explosion
HGS	**18.89 (0.00%)**	**19.36 (0.00%)**	**19.45 (0.00%)**
LKH3	19.09 (1.06%)	19.55 (0.98%)	19.66 (1.08%)
Omni-VRP	19.39 (2.65%)	19.91 (2.84%)	20.07 (3.19%)
Ours (Core)	19.35 (2.44%)	19.75 (2.01%)	19.81 (1.85%)
Ours (Full)	19.13 (1.27%)	19.53 (0.88%)	19.59 (0.72%)

Table 4: Ablations across 16 VRP variants at $n = 50$. Diversity counts are averaged over instances after M20 with A8S50. Full includes all training components; + AE adds action entropy to w/o Div.; – denotes unreported values. Bold marks the lowest gap within each inference budget.

	A8S50					A8Greedy1	
Metric	w/o Div.	+ AE	w/o RS	w/o REP	Full	w/o REP	Full
M12 Gap	0.237%	0.280%	**0.153%**	0.181%	0.170%	0.911%	**0.845%**
M20 Gap	0.230%	0.194%	0.126%	0.124%	**0.117%**	0.850%	**0.791%**
DFS Gap	0.173%	0.123%	0.074%	0.061%	**0.059%**	0.769%	**0.720%**
Unique endpoints	72.86	149.59	179.52	220.29	220.07	–	–
Unique routes	114.73	217.57	251.77	337.12	330.59	–	–

HGS remains best on all three distributions. These results suggest that our pipeline remains effective under the evaluated distribution shifts.

5.3 Ablations.

Refined-outcome diversity. We evaluate removing both proposed diversity terms (w/o Div.) and retaining solution-level diversity while removing route-level diversity (w/o RS). Table 4 shows that the full diversity objective reduces the final gap while increasing the numbers of distinct refined endpoints and routes. However, w/o RS achieves a lower gap at M12, suggesting a quality–diversity tradeoff under limited refinement. The full objective performs better after M20 and DFS, indicating that the benefit of additional route diversity depends on the subsequent optimization budget. We also replace the proposed diversity terms with conventional action entropy (+ AE). Although action entropy improves diversity counts and the M20 and DFS gaps over w/o Div., solution-level ROE alone further improves both diversity counts and all three reported gaps.

Refined-edge prior. Removing REP disables both its supervision and decision prior, with all other settings unchanged. Table 4 shows lower macro-average gaps with REP at all reported stages under both inference budgets in these runs. The gains are larger under A8Greedy1, which uses one greedy candidate per augmentation, and diminish with further refinement and route recombination. Under A8S50, REP improves quality despite slightly fewer distinct endpoints and routes, supporting its complementary role in guiding construction.

Additional experiments. Appendix C provides per-variant results, a comparison with CoEKS (Yu et al., 2026) on atomic variants, and sensitivity to the candidate fraction used for diversity credit. We also evaluate zero-shot transfer from six training tasks to ten unseen constraint combinations (Appendix C.4) and performance on 22 CVRPLIB X-set instances (Appendix C.5).

6 Conclusion, Limitation, and Future Work

In this study, we propose a basin-guided reinforcement learning framework for vehicle routing problems that combines neural construction with bounded local refinement. Since distinct customer permutations can lead to the same refined outcome, we encourage diversity among post-processing outcomes through ROE. REP complements this objective by using refined customer adjacencies as weak supervision for the constructor. Experiments across 16 VRP variants show improved solution quality over neural baselines equipped with the same post-processing stages, while optional route recombination provides further quality gains at additional computational cost.

A limitation of our framework is its reliance on routing-specific components, including dynamic-programming splitting and local-search neighborhoods. Although the framework covers 16 VRP variants, extending it to structurally different combinatorial optimization problems would require adapting the post-processing operators and structural supervision targets. Future work will investigate how to incorporate domain knowledge more systematically and generalize learning from post-processing feedback across broader problem classes.

AI USE STATEMENT

Writing assistance. The authors developed the manuscript structure and wrote the initial draft. Generative AI tools assisted with polishing the language and condensing the text to improve clarity and information density.

Retrieval and discovery. After independently designing the proposed method, the authors used AI tools to help search for related work that they might have overlooked, including potentially similar approaches.

Research feedback. The authors asked AI tools to identify potential methodological weaknesses, reviewed the responses, and decided which suggestions, if any, to adopt.

Technical execution. AI tools assisted with implementing GPU-based local-move operators and organizing experimental data. The authors independently checked the correctness of the implementation and the organized data. The authors take responsibility for the final manuscript, claims, code, and reported results.

REPRODUCIBILITY STATEMENT

All comparisons use publicly available datasets widely adopted in the routing community. Neural baselines use publicly available code and checkpoints from the repositories listed in Table 10 (Appendix B). Appendix B details the training procedure, hyperparameters, inference and post-processing settings, evaluation metrics, and timing protocol, while Appendix A.4 describes the detailed network architecture. Tables 11 and 12 report per-variant results at $n = 50$ and $n = 100$, respectively. The submitted supplementary materials include per-instance evaluation results for all 16 variants. Our code and checkpoints will be publicly released upon acceptance.

REFERENCES

JE Beasley. Route first—cluster second methods for vehicle routing. *Omega*, 11(4):403–408, 1983. ISSN 0305-0483. doi: https://doi.org/10.1016/0305-0483(83)90033-6. URL https://www.sciencedirect.com/science/article/pii/0305048383900336.

Irwan Bello*, Hieu Pham*, Quoc V. Le, Mohammad Norouzi, and Samy Bengio. Neural combinatorial optimization with reinforcement learning, 2017. URL https://openreview.net/forum?id=rJY3vK9eg.

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. *Transactions on Machine Learning Research*, 2025. ISSN 2835-8856. URL https://openreview.net/forum?id=QzGLoaOPiY.

Jieyi Bi, Zhiguang Cao, Jianan Zhou, Wen Song, Yaoxin Wu, Jie Zhang, Yining Ma, and Cathy Wu. Towards efficient constraint handling in neural solvers for routing problems. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=raDFGuQxvD.

Vincent Furnon and Laurent Perron. OR-Tools routing library, 2024. URL https://developers.google.com/optimization/routing/.

Shuangchun Gui, Suyu Liu, Xuehe Wang, and Zhiguang Cao. Chain-of-context learning: Dynamic constraint understanding for multi-task VRPs. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=AhE6aSlz5g.

Keld Helsgaun. An extension of the Lin-Kernighan-Helsgaun TSP solver for constrained traveling salesman and vehicle routing problems. Technical report, Roskilde University, 2017. URL https://webhotel4.ruc.dk/~keld/research/LKH-3/LKH-3_REPORT.pdf.

André Hottung, Mridul Mahajan, and Kevin Tierney. Polynet: Learning diverse solution strategies for neural combinatorial optimization. In Y. Yue, A. Garg, N. Peng, F. Sha, and R. Yu

(eds.), *International Conference on Learning Representations*, volume 2025, pp. 80417–80435, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/file/c7b9810b2e27bccdc8d605a3a54c8cd6-Paper-Conference.pdf.

Ziwei Huang, Jianan Zhou, Zhiguang Cao, and Yixin Xu. Rethinking light decoder-based solvers for vehicle routing problems. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=4pRwkYpa2u.

Q. Huangfu and J. A. J. Hall. Parallelizing the dual revised simplex method. *Mathematical Programming Computation*, 10(1):119–142, 2018. doi: 10.1007/s12532-017-0130-5. URL https://doi.org/10.1007/s12532-017-0130-5.

Minsu Kim, Jinkyoo Park, and Joungho Kim. Learning collaborative policies to solve NP-hard routing problems. In A. Beygelzimer, Y. Dauphin, P. Liang, and J. Wortman Vaughan (eds.), *Advances in Neural Information Processing Systems*, 2021. URL https://openreview.net/forum?id=pDgN3l3EAF.

Yeong-Dae Kwon, Jinho Choo, Byoungjip Kim, Iljoo Yoon, Youngjune Gwon, and Seungjai Min. Pomo: Policy optimization with multiple optima for reinforcement learning. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin (eds.), *Advances in Neural Information Processing Systems*, volume 33, pp. 21188–21198. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper_files/paper/2020/file/f231f2107df69eab0a3862d50018a9b2-Paper.pdf.

J. K. Lenstra and A. H. G. Rinnooy Kan. Complexity of vehicle routing and scheduling problems. *Networks*, 11(2):221–227, 1981. doi: https://doi.org/10.1002/net.3230110211. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/net.3230110211.

Han Li, Fei Liu, Zhi Zheng, Yu Zhang, and Zhenkun Wang. CaDA: Cross-problem routing solver with constraint-aware dual-attention. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 35438–35456. PMLR, 2025. URL https://proceedings.mlr.press/v267/li25bi.html.

Suyu Liu, Zhiguang Cao, Nan Yin, and Yew-Soon Ong. Scale-Net: A hierarchical U-Net framework for cross-scale generalization in multi-task vehicle routing. *Proceedings of the AAAI Conference on Artificial Intelligence*, 40(17):14295–14303, 2026. doi: 10.1609/aaai.v40i17.38444. URL https://ojs.aaai.org/index.php/AAAI/article/view/38444.

Helena Ramalhinho Lourenço, Olivier C. Martin, and Thomas Stützle. Iterated local search: Framework and applications. In *Handbook of Metaheuristics*, pp. 129–168. Springer International Publishing, Cham, 2019. ISBN 978-3-319-91086-4. doi: 10.1007/978-3-319-91086-4_5. URL https://doi.org/10.1007/978-3-319-91086-4_5.

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*, volume 36, 2023.

Dian Meng, Yaoxin Wu, Yaqing Hou, and Zhiguang Cao. PomtVRS: Preference-optimized multi-task vehicle routing solver with preference gating. In *Forty-third International Conference on Machine Learning*, 2026. URL https://openreview.net/forum?id=vJsJPNRFIJ.

Yuxin Pan, Zhiguang Cao, Chengyang GU, Liu Liu, Peilin Zhao, Yize Chen, and Fangzhen Lin. Multi-task vehicle routing solver via mixture of specialized experts under state-decomposable MDP. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025. URL https://openreview.net/forum?id=ezSyZM6Lp7.

Christian Prins. A simple and effective evolutionary algorithm for the vehicle routing problem. *Computers & Operations Research*, 31(12):1985–2002, 2004. ISSN 0305-0548. doi: https://doi.org/10.1016/S0305-0548(03)00158-8. URL https://www.sciencedirect.com/science/article/pii/S0305054803001588.

Yves Rochat and Éric D. Taillard. Probabilistic diversification and intensification in local search for vehicle routing. *Journal of Heuristics*, 1(1):147–167, September 1995. ISSN 1572-9397. doi: 10.1007/BF02430370.

Anand Subramanian, Eduardo Uchoa, and Luiz Satoru Ochi. A hybrid algorithm for a class of vehicle routing problems. *Computers & Operations Research*, 40(10):2519–2531, 2013. ISSN 0305-0548. doi: https://doi.org/10.1016/j.cor.2013.01.013. URL https://www.sciencedirect.com/science/article/pii/S030505481300021X.

Eduardo Uchoa, Diego Pecin, Artur Pessoa, Marcus Poggi, Thibaut Vidal, and Anand Subramanian. New benchmark instances for the capacitated vehicle routing problem. *European Journal of Operational Research*, 257(3):845–858, 2017. ISSN 0377-2217. doi: https://doi.org/10.1016/j.ejor.2016.08.012. URL https://www.sciencedirect.com/science/article/pii/S0377221716306270.

Thibaut Vidal. Hybrid genetic search for the cvrp: Open-source implementation and swap* neighborhood. *Computers & Operations Research*, 140:105643, 2022. ISSN 0305-0548. doi: https://doi.org/10.1016/j.cor.2021.105643. URL https://www.sciencedirect.com/science/article/pii/S030505482100349X.

Thibaut Vidal, Teodor Gabriel Crainic, Michel Gendreau, and Christian Prins. A hybrid genetic algorithm with adaptive diversity management for a large class of vehicle routing problems with time-windows. *Computers & Operations Research*, 40(1):475–489, 2013. ISSN 0305-0548. doi: https://doi.org/10.1016/j.cor.2012.07.018. URL https://www.sciencedirect.com/science/article/pii/S0305054812001645.

Thibaut Vidal, Gilbert Laporte, and Piotr Matl. A concise guide to existing and emerging vehicle routing problem variants. *European Journal of Operational Research*, 286(2):401–416, 2020. ISSN 0377-2217. doi: https://doi.org/10.1016/j.ejor.2019.10.010. URL https://www.sciencedirect.com/science/article/pii/S0377221719308422.

Wen Wang, Xiangchen Wu, Liang Wang, Hao Hu, Xianping Tao, and Linghao Zhang. Solving the min-max multiple traveling salesmen problem via learning-based path generation and optimal splitting. In *ECAI*, pp. 4889–4896, 2025. URL https://doi.org/10.3233/FAIA251399.

Wen Wang, Xiangchen Wu, Liang Wang, Hao Hu, and Xianping Tao. A unified knowledge embedded reinforcement learning-based framework for generalized capacitated vehicle routing problems, 2026. URL https://arxiv.org/abs/2605.14416.

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

Niels A. Wouda, Leon Lan, and Wouter Kool. PyVRP: a high-performance VRP solver package. *INFORMS Journal on Computing*, 36(4):943–955, 2024. doi: 10.1287/ijoc.2023.0055. URL https://doi.org/10.1287/ijoc.2023.0055.

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, 2022.

Liang Xin, Wen Song, Zhiguang Cao, and Jie Zhang. Multi-decoder attention model with embedding glimpse for solving vehicle routing problems. *Proceedings of the AAAI Conference on Artificial Intelligence*, 35(13):12042–12049, May 2021a. doi: 10.1609/aaai.v35i13.17430. URL https://doi.org/10.1609/aaai.v35i13.17430.

Liang Xin, Wen Song, Zhiguang Cao, and Jie Zhang. Neurolkh: combining deep learning model with lin-kernighan-helsgaun heuristic for solving the traveling salesman problem. In *Proceedings of the 35th International Conference on Neural Information Processing Systems*, NIPS '21, Red Hook, NY, USA, 2021b. Curran Associates Inc. ISBN 9781713845393.

Zikang Yu, Jinbiao Chen, and Jiahai Wang. Combination-of-experts with knowledge sharing for cross-task vehicle routing problems. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=lHBs9mbgwp.

Jianan Zhou, Yaoxin Wu, Wen Song, Zhiguang Cao, and Jie Zhang. Towards omni-generalizable neural methods for vehicle routing problems. In *Proceedings of the 40th International Conference on Machine Learning*, ICML23. JMLR.org, 2023.

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

APPENDIX CONTENTS

A Additional Method Details **15**

A.1 VRP Definitions and Task Set 15

A.2 Dynamic-Programming Route Splitting 16

A.3 Bounded Local Refinement 17

A.4 Neural Constructor and Refined-Edge Prior 18

A.5 Refined-Outcome Entropy 19

A.6 Route-Pool Recombination 19

B Reproducibility Details **21**

B.1 Datasets and Evaluation Scope 21

B.2 Training and Fine-Tuning 21

B.3 Inference Configurations 22

B.4 Baseline Configurations 22

B.5 Evaluation Metrics and Timing Protocol 23

B.6 Ablation Configurations 23

C Additional Experimental Results **24**

C.1 Per-Variant Results on the 16-Variant Benchmark 24

C.2 Comparison with CoEKS on Atomic VRP Variants 28

C.3 Sensitivity to the Diversity-Credit Candidate Fraction 28

C.4 Zero-Shot Generalization to Unseen Constraint Combinations 29

C.5 Results on a CVRPLIB X-Set Subset 30

A Additional Method Details

A.1 VRP Definitions and Task Set

We follow the VRP attribute definitions of RouteFinder (Berto et al., 2025), restricting attention to the 16 single-depot variants formed by combining O, B, L, and TW with capacity constraints. Mixed-backhaul and multi-depot extensions are outside this task set. We summarize the conventions relevant to our experiments below.

Capacity and backhauls (B). All variants require each customer to be served exactly once and minimize total Euclidean travel distance. Vehicles have capacity Q. Without B, customers receive deliveries from the depot. With B, customers are either delivery (linehaul) or pickup (backhaul) customers; all deliveries on a route precede its pickups. The total delivery demand and total pickup demand on each route must each be at most Q.

Open routes (O). Every route starts at the depot. Closed routes also return to the depot, whereas open routes end at their last customer. The return leg is therefore omitted from both the objective and the distance-limit calculation for open routes.

Distance limits (L). Each route's travel distance must not exceed the instance-specific limit L. RouteFinder also refers to this attribute as a duration limit; here, it constrains travel distance, excluding waiting and service times. These times are accounted for separately when checking TW feasibility.

Time windows (TW). Customer i has a service window $[e_i, l_i]$ and service duration d_i^{srv}. Service starts no earlier than arrival or e_i, and no later than l_i; early arrivals wait. Travel time equals Euclidean distance under unit speed. For closed routes, service and travel must also allow return within the depot time horizon.

Attribute combinations. All enabled constraints apply simultaneously. Table 5 lists the resulting tasks; capacity remains active even when C is omitted from a variant's name. We use RouteFinder's instance generator as described in Appendix B.

Table 5: The 16 VRP variants evaluated in this work. Each row pairs a closed-route task with its open-route counterpart. All tasks include capacity constraints.

Additional attributes	Closed routes	Open routes (O)
None	CVRP	OVRP
B	VRPB	OVRPB
L	VRPL	OVRPL
TW	VRPTW	OVRPTW
B, L	VRPBL	OVRPBL
B, TW	VRPBTW	OVRPBTW
L, TW	VRPLTW	OVRPLTW
B, L, TW	VRPBLTW	OVRPBLTW

Canonical endpoint representation. Represent a refined solution as a collection of nonempty customer sequences $\hat{s} = \{\rho_1, \ldots, \rho_m\}$, with depot visits implicit and empty routes discarded. For reversal-symmetric variants (CVRP and VRPL), each route takes its lexicographically smaller orientation:

$$\bar{\rho}_q = \min_{\text{lex}}\{\rho_q, \text{rev}(\rho_q)\}. \tag{11}$$

For all other variants, $\bar{\rho}_q = \rho_q$. To remove vehicle-order symmetry, we then sort the routes lexicographically:

$$\kappa(\hat{s}) = \text{sort}_{\text{lex}}(\bar{\rho}_1, \ldots, \bar{\rho}_m). \tag{12}$$

Table 6: Symmetries removed by the canonical solution representation κ. Vehicle-order symmetry is valid for all variants because vehicles are homogeneous. Route-reversal symmetry is valid only for closed variants without backhauls or time windows.

Variant	Vehicle-order symmetry	Route-reversal symmetry
CVRP	✓	✓
OVRP	✓	–
VRPB	✓	–
VRPL	✓	✓
VRPTW	✓	–
OVRPB	✓	–
OVRPL	✓	–
OVRPTW	✓	–
VRPBL	✓	–
VRPBTW	✓	–
VRPLTW	✓	–
OVRPBL	✓	–
OVRPBTW	✓	–
OVRPLTW	✓	–
VRPBLTW	✓	–
OVRPBLTW	✓	–

Route boundaries are retained, and two refined solutions share an endpoint identifier exactly when their canonical representations coincide. This operation changes only the identifier, leaving the underlying solutions unchanged and preserving distinctions beyond the symmetries in Table 6.

For example, the CVRP solutions $\{(1,4,3),(2,5)\}$ and $\{(5,2),(3,4,1)\}$ both yield the identifier $((1,4,3),(2,5))$. For variants without route-reversal symmetry, these route collections retain distinct identifiers.

A.2 Dynamic-Programming Route Splitting

Following RL-RFCS (Wang et al., 2026), DP partitions a customer permutation $\tau = (\tau_1, \ldots, \tau_n)$ into contiguous feasible routes without changing customer order.

Segment costs and feasibility. For $1 \le i \le j \le n$, let $r_{i,j} = (\tau_i, \ldots, \tau_j)$ be the customer sequence of one route. Its travel cost is

$$D_{i,j} = c_{0,\tau_i} + \sum_{k=i}^{j-1} c_{\tau_k,\tau_{k+1}} + (1-o)c_{\tau_j,0}, \tag{13}$$

where $o = 1$ for open routes and $o = 0$ otherwise. Define $w_{i,j} = D_{i,j}$ if this route satisfies all active constraints in Appendix A.1, and $w_{i,j} = +\infty$ otherwise. Capacity checks use separate delivery and pickup totals, and B requires deliveries to precede pickups. When L is active, $D_{i,j} \le L$ is required. For TW, initialize the departure time at the depot to zero and scan customers in order. If a is the preceding node and t its departure time, service at customer v begins at $b_v = \max\{t + c_{a,v}, e_v\}$; require $b_v \le l_v$ and update $t = b_v + d_v^{\mathrm{srv}}$. Closed routes additionally require $t + c_{\tau_j,0}$ not to exceed the depot horizon. Waiting and service affect feasibility but not the distance objective.

Recurrence and reconstruction. Let $F(j)$ be the minimum cost of serving the first j customers. With $F(0) = 0$, compute

$$F(j) = \min_{0 \le i < j} \{F(i) + w_{i+1,j}\}, \qquad j = 1, \ldots, n. \tag{14}$$

Store a minimizing predecessor for each state and backtrack from n to recover route boundaries. Ties are resolved deterministically. The resulting solution $s = \mathrm{DP}(\tau)$ has cost $F(n)$.

A.3 BOUNDED LOCAL REFINEMENT

Algorithm 1 summarizes the post-processing of a customer permutation. DP first constructs a feasible solution as described in Appendix A.2; the remaining steps define P_M. The same refinement procedure is used during training and inference.

Algorithm 1 DP splitting and bounded local refinement

Require: Instance x, customer permutation τ, move budget M
Ensure: Refined feasible solution $\hat{s}$

$s \leftarrow \mathrm{DP}(\tau)$
$\mathcal{O} \leftarrow \{\text{Relocate-1}, \text{Swap-1-1}, \text{Or-opt-2}, \text{Or-opt-3}, \text{2-opt*}\}$
if x has no time-window constraint **then**
    $s \leftarrow \textsc{TwoOptClosure}(x, s)$
end if
for $t = 1, \ldots, M$ **do**
    $\mathcal{C} \leftarrow \{m \in \bigcup_{o \in \mathcal{O}} \mathcal{N}_o(s) : m(s) \text{ is feasible}, \ \Delta(m; s) < 0\}$
    if $\mathcal{C} = \emptyset$ **then**
        break
    end if
    $m^* \leftarrow \textsc{BestMove}(\mathcal{C}, s)$
    $s \leftarrow m^*(s)$
    if x has no time-window constraint **then**
        $s \leftarrow \textsc{TwoOptClosure}(x, s)$
    end if
end for
return $\hat{s} \leftarrow s$

Here, $\mathcal{N}_o(s)$ denotes the moves available under operator o, and $\Delta(m; s) = C(m(s)) - C(s)$ is the change in total travel cost. BESTMOVE selects the candidate with the smallest cost change, using deterministic tie-breaking.

Neighborhoods and budget. Each round considers the same five cross-route operators. Relocate-1 transfers one customer to an insertion position in another route; Swap-1-1 exchanges one customer from each of two routes; Or-opt-2/3 transfers a consecutive chain of two or three customers, considering both orientations; and 2-opt* exchanges two route suffixes without reversing them. An insertion position may precede any customer or follow the last customer of a route. M5, M12, and M20 allow at most 5, 12, and 20 accepted cross-route moves, respectively.

Intra-route refinement. TWOOPTCLOSURE repeatedly applies the best feasible improving reversal of a contiguous subsequence within one route. It changes customer order without transferring customers between vehicles. For non-TW variants, this descent is applied before the first cross-route round and after each accepted cross-route move. TW variants omit it but retain all five cross-route operators. Intra-route moves do not count toward M. Each descent is bounded by a fixed cap of 4,096 accepted moves.

Feasibility checks. Only moves that preserve all active constraints in Appendix A.1 are considered. Route prefix and suffix summaries are constructed in $O(n)$ time per cross-route round. Each candidate move is evaluated through a constant number of summary concatenations, enabling $O(1)$ cost and feasibility checks, including time windows, without rescanning entire routes.

Computational cost. Each cross-route round requires $O(n)$ summary construction and $O(n^2)$ candidate evaluation, hence $O(n^2)$ time. With fixed M and a fixed intra-route descent cap, the overall refinement cost is $O(n^2)$ in the number of customers.

A.4 Neural Constructor and Refined-Edge Prior

Our constructor combines a shared Transformer encoder following RouteFinder (Berto et al., 2025), constraint-specific residual experts inspired by MVMoE (Zhou et al., 2024), and an LSTM-enhanced pointer decoder following RL-RFCS (Wang et al., 2026). It generates a customer permutation, which is partitioned into feasible routes by the DP procedure in Appendix A.2. Table 7 summarizes the architectural settings.

Table 7: Architecture used by the main model.

Component	Setting
Node embedding dimension d	128
Encoder layers / attention heads	6 / 8
Gated-MLP hidden dimension	512
Constraint experts per layer	4
Constraint-expert bottleneck	32
Decoder history / hidden dimension	LSTM / 128
Context-adapter bottleneck	64
REP projection dimension d_E	32
Logit clipping constant	10
Encoder / decoder parameters	1.776M / 0.289M
Total trainable parameters	2.065M

A.4.1 Encoder

Customer inputs contain coordinates, capacity-normalized signed demand, time-window bounds, and service duration, $\mathbf{x}_i = [x_i, y_i, q_i, e_i, l_i, d_i^{\mathrm{srv}}]$. Positive and negative demands represent linehaul and backhaul customers, respectively. The depot input contains its coordinates, the normalized route-length limit, and four binary indicators $\mathbf{f} = (f_O, f_B, f_L, f_{TW})$. Separate affine maps project the six-dimensional customer inputs and seven-dimensional depot input into $d = 128$ dimensions. Task indicators also activate the corresponding residual experts.

The encoder has six Transformer blocks, each with eight-head self-attention, pre-RMS normalization, and a SiLU-gated feed-forward network of hidden dimension 512. Each block additionally contains four residual experts with bottleneck dimension 32. Writing the node embeddings as columns of H, the block computes

$$U^{(\ell)} = H^{(\ell-1)} + \mathrm{MHA}^{(\ell)}\left(\mathrm{RMSNorm}_{\mathrm{attn}}^{(\ell)}(H^{(\ell-1)})\right), \tag{15}$$

$$\bar{U}^{(\ell)} = \mathrm{RMSNorm}_{\mathrm{ffn}}^{(\ell)}(U^{(\ell)}), \tag{16}$$

$$H^{(\ell)} = U^{(\ell)} + \mathrm{FFN}^{(\ell)}(\bar{U}^{(\ell)}) + \sum_{r\in\{O,B,L,TW\}} f_r E_r^{(\ell)}(\bar{U}^{(\ell)}), \tag{17}$$

where $E_r(U) = W_{r,\mathrm{up}}\,\mathrm{SiLU}(W_{r,\mathrm{down}}U)$. Experts are activated directly by the constraint indicators and added without a learned router. The final representations are $H = H^{(6)} = [\mathbf{h}_0, \ldots, \mathbf{h}_n]$, with graph embedding $\mathbf{g} = (n+1)^{-1}\sum_{i=0}^{n}\mathbf{h}_i$.

A.4.2 Decoder

The encoder outputs and their static projections are shared across sampled trajectories. At step t, the current node is $c_1 = 0$ or $c_t = \tau_{t-1}$ for $t > 1$. The decoder context is

$$\mathbf{z}_t = \mathrm{LN}_1(W_s[\mathbf{g}\|\mathbf{h}_0\|\mathbf{h}_{c_t}] + W_v\mathbf{v}_t + \mathbf{b}_s)\,. \tag{18}$$

In the main protocol, the vehicle context is $\mathbf{v}_t = [m_{\min}/n, m_{\min}/n, 0, R_t/m_{\min}]$. Here, $m_{\min}$ is the capacity-based vehicle-count lower bound, obtained by rounding up the larger of the total normalized delivery and pickup workloads, with a minimum value of one. The remaining workload R_t is the larger of the total normalized delivery and pickup workloads among unvisited customers. This context does not impose a vehicle-count constraint on DP.

A 128-dimensional LSTM processes $\mathbf{z}_t$ to summarize the construction history. Its output $\mathbf{o}_t$ is layer-normalized and passed through a residual SiLU adapter with bottleneck dimension 64:

$$\tilde{\mathbf{o}}_t = \bar{\mathbf{o}}_t + W_{A,2}\,\mathrm{SiLU}(W_{A,1}\bar{\mathbf{o}}_t + \mathbf{b}_{A,1}) + \mathbf{b}_{A,2}, \qquad \bar{\mathbf{o}}_t = \mathrm{LN}_2(\mathbf{o}_t). \tag{19}$$

The adapted context supplies the query for eight-head attention over the encoded nodes, with a visited-customer mask. The projected attention output $\mathbf{a}_t$ gives the base pointer score

$$\ell_{t,j}^{\mathrm{base}} = \frac{\mathbf{a}_t^\top (W_K^\ell \mathbf{h}_j + \mathbf{b}_K^\ell)}{\sqrt{d}}. \tag{20}$$

REP augments this score at non-initial steps as in Equation 6. The resulting logits are clipped by $10\tanh(\ell_{t,j})$ and normalized over unvisited customers. The depot is not an action; routing constraints are handled by DP after the complete permutation is generated.

A.4.3 REP Head and Supervision

REP uses a shared pair of 32-dimensional projections of the encoder output, as defined in Equation 4. Its edge scores are computed once per instance and reused across trajectories. For completeness, the batch-balanced loss in Section 4.2 is given below. Let $Y_b = \bar{A}_{x_b}$ be the refined-edge target, using the same symmetry convention as the main text, and let $M_{ij} = \mathbb{I}[i, j \in \{1, \dots, n\},\ i \neq j]$.

$$\begin{aligned}\mathcal{L}_{\mathrm{REP}} = \frac{1}{2}\Bigg[&\frac{\sum_{b,i,j} M_{ij} Y_{b,ij}\ \mathrm{softplus}(-e_{b,ij})}{\sum_{b,i,j} M_{ij} Y_{b,ij}} \\ &+ \frac{\sum_{b,i,j} M_{ij}(1 - Y_{b,ij})\ \mathrm{softplus}(e_{b,ij})}{\sum_{b,i,j} M_{ij}(1 - Y_{b,ij})}\Bigg],\end{aligned} \tag{21}$$

A term with zero total target weight is defined as zero. Targets are treated as constants. REP supervision updates the shared encoder and edge projections, while the policy-gradient objective also trains the decoder and the decision-prior scale α_E.

A.5 Refined-Outcome Entropy

In our training configuration ($K = 32$, $\gamma = 0.25$), the quality gate retains the eight candidates with the highest refined rewards, together with any additional candidates tied at the cutoff. For each diversity term, we subtract its mean credit over the gated candidates and set the advantages outside the gate to zero. The solution-level implementation uses negative log occurrence counts, which differ from the negative log frequencies in Section 4.3 only by a constant and therefore give the same centered advantages.

Both diversity terms are independently scaled to the RMS of the original quality advantage A_x^Q before combination. For $g \in \{\mathrm{sol}, \mathrm{route}\}$, we use

$$\widetilde{A}_x^g = \begin{cases} A_x^g \dfrac{\mathrm{RMS}(A_x^Q)}{\mathrm{RMS}(A_x^g)}, & \mathrm{RMS}(A_x^g) > \epsilon, \\ \mathbf{0}, & \mathrm{RMS}(A_x^g) \leq \epsilon, \end{cases} \tag{22}$$

where $\epsilon = 10^{-8}$ and $\mathrm{RMS}(a) = \sqrt{K^{-1}\sum_{k=1}^{K} a_k^2}$ includes all K candidates, including the zero entries outside the gate.

This threshold suppresses negligible diversity signals rather than amplifying them. If all refined rewards are equal, the quality advantage is zero and both scaled diversity terms vanish. Equal endpoint frequencies within the gate also yield zero solution-level advantage, including when all endpoints are distinct; route-level credit can still distinguish candidates through their customer groupings.

A.6 Route-Pool Recombination

The full inference pipeline recombines feasible routes from refined candidates without changing their internal customer order. Following the route-pool perspective of hybrid VRP methods (Subramanian et al., 2013), we formulate this step as restricted set partitioning and solve it using a time-bounded branch-and-bound depth-first search (DFS).

Route pool. Let $V_c = \{1, \ldots, n\}$ and let $\mathcal{R}$ be the pool of feasible routes. Each route r has customer set S_r and travel cost c_r. Complete refined candidates are stably sorted by ascending total solution cost, and their routes are scanned in this order, starting with the best candidate. The deduplication key is the unordered customer set S_r, independent of visit order or route orientation. Among routes with the same customer set, only the cheapest is retained. This preserves the best recombination available before any pool-size limit is applied, since all active routing constraints are route-local. Once the pool reaches its limit, no new customer sets are admitted, but all remaining candidates are still scanned to replace retained routes with cheaper routes serving the same customer sets. The pool admits all routes of the best complete candidate first; subsequent replacements preserve their customer sets and cannot increase their costs. Thus the pool retains a feasible cover no more expensive than that candidate. The search selects routes by solving

$$\begin{aligned} \min_{\mathbf{x}} \quad & \sum_{r \in \mathcal{R}} c_r x_r, \\ \text{s.t.} \quad & \sum_{r : i \in S_r} x_r = 1, \quad i \in V_c, \\ & x_r \in \{0, 1\}, \qquad r \in \mathcal{R}. \end{aligned} \tag{23}$$

Thus every customer is covered exactly once by independently feasible routes. The best complete refined candidate is retained as the initial incumbent, so the returned solution cannot be worse than that candidate.

Lower bound. We obtain nonnegative customer prices by solving the dual of the set-cover relaxation once per instance:

$$\begin{aligned} \max_{\boldsymbol{\pi}} \quad & \sum_{i \in V_c} \pi_i, \\ \text{s.t.} \quad & \sum_{i \in S_r} \pi_i \le c_r, \quad r \in \mathcal{R}, \\ & \pi_i \ge 0, \qquad i \in V_c. \end{aligned} \tag{24}$$

We solve this LP using the C++ interface of HiGHS v1.14.0 (Huangfu & Hall, 2018). For covered customers C and accumulated cost g, the remaining cost is bounded below by $L(C) = \sum_{i \in V_c \setminus C} \pi_i$. The state is pruned when $g + L(C) \ge UB$, where UB is the incumbent cost.

Search procedure. DFS starts from $(C, g) = (\emptyset, 0)$. For each uncovered customer i, define the compatible route set $\mathcal{R}_i(C) = \{r \in \mathcal{R} : i \in S_r,\ S_r \cap C = \emptyset\}$. We branch on $i^* \in \arg\min_{i \in V_c \setminus C} |\mathcal{R}_i(C)|$; if $\mathcal{R}_{i^*}(C) = \emptyset$, the state is infeasible. Routes in $\mathcal{R}_{i^*}(C)$ are explored in increasing reduced cost $\bar{c}_r = c_r - \sum_{i \in S_r} \pi_i \ge 0$. Selecting r gives the child state $(C \cup S_r, g + c_r)$. When $C = V_c$, the selected routes form a complete solution and replace the incumbent if their cost is lower.

The child-state lower bound satisfies

$$g + c_r + L(C \cup S_r) = g + L(C) + \bar{c}_r. \tag{25}$$

Thus, when this bound reaches UB, the current route and all subsequent routes in the ordered list are pruned. We also discard a state (C, g) if a previously reached state (C, g') satisfies $g' \le g$, since both states admit the same remaining routes.

Search returns the best complete incumbent when the time budget is reached. If all branches have been explored or pruned, the returned solution is optimal within the supplied route pool.

B Reproducibility Details

B.1 Datasets and Evaluation Scope

The uniform benchmark uses 1,000 RouteFinder test instances per variant and size. For each Omni-VRP OOD distribution (Gaussian Mixture $(2, 5)$, Rotation, and Explosion), we evaluate the first 1,000 of the 2,000 CVRP200 instances; the quoted HGS, LKH3, and Omni-VRP results correspond to this same subset. The CVRP1000 comparison uses the 128 official Scale-Net instances. Training instances are generated online using RouteFinder's generator.

B.2 Training and Fine-Tuning

We train separate $n = 50$ and $n = 100$ models. Each batch samples one of the 16 variants uniformly and contains 128 instances, with 32 candidate permutations per instance; each epoch comprises 2,500 updates. All stages use Adam, gradient clipping at 1.0, REP weight $\lambda_E = 0.1$, and quality-gate fraction $\gamma = 0.25$. Following RL-RFCS (Wang et al., 2026), we use a sampling temperature of 1.0 for all training and fine-tuning runs and 1.6 for sampled inference at test time. The starting customer of each candidate is selected randomly, without enforcing distinct starting customers across candidates. Table 8 lists the stage-dependent settings; architecture details are given in Appendix A.4. We use fixed hyperparameter settings for our model without hyperparameter tuning.

Table 8: Training schedule. Stages 1 and 2 apply separately to the $n = 50$ and $n = 100$ models.

Setting	Stage 1	Stage 2	Fine-tuning
Problem size n	50 / 100	50 / 100	200
Epochs	1–90	91–100	91–100
Initialization	From scratch	Stage 1	$n = 100$, epoch 90
Training distribution	RouteFinder	RouteFinder	RouteFinder
Variant sampling	Uniform over 16	Uniform over 16	Uniform over 16
Instances per batch B	128	128	128
Candidates per instance K	32	32	32
Updates per epoch U	2,500	2,500	2,500
Refinement steps M	5	12	12
Optimizer	Adam	Adam	Adam
Learning rate	3×10^{-4}	3×10^{-5}	3×10^{-5}
Gradient clipping	1.0	1.0	1.0
REP weight λ_E	0.1	0.1	0.1
Solution-diversity weight λ_{sol}	1.0	5.0	5.0
Route-diversity weight λ_{route}	0	2.0	2.0
Quality-gate fraction γ	0.25	0.25	0.25

The $n = 200$ model is fine-tuned on the same 16-variant task set from the $n = 100$ epoch-90 checkpoint. For each of our models, evaluation uses the final checkpoint at the end of its training or fine-tuning run.

CVRP1000 fine-tuning. We initialize from the final $n = 200$ main-model checkpoint (E99) and fine-tune exclusively on CVRP with 1,000 customers and vehicle capacity 500. Instances have uniformly distributed coordinates and integer demands from 1 to 9 and are generated online with training seed 3333. Each batch contains 8 instances with 32 sampled trajectories per instance, without geometric augmentation (A1). Each epoch comprises 800 instances and 100 updates; 105 epochs (E0–E104) yield 10,500 updates, 84,000 generated instances, and 2,688,000 trajectories.

Training uses M12 refinement, diversity weights $\lambda_{\text{sol}} = 5$ and $\lambda_{\text{route}} = 2$, and a quality-gate fraction of $\gamma = 0.25$ for both terms. REP uses mean edge-frequency targets over all refined endpoints, with $\lambda_E = 0.1$ and its decision prior enabled. We reinitialize Adam without inheriting the $n = 200$ optimizer state and use a fixed learning rate of 3×10^{-5} without decay. All encoder, decoder, and REP parameters are updated in FP32, with encoder and decoder gradients clipped separately at 1.0. The training sampling temperature is 1.0, and the run takes approximately 10 hours and 14 minutes.

B.3 Inference Configurations

Table 9 summarizes the inference budgets used in the reported experiments. A8Sk uses eight geometric augmentations with k sampled candidates each; A8Greedy1 uses one greedy candidate per augmentation. Mk allows at most k accepted cross-route moves, and DFS follows local refinement.

Table 9: Inference budgets for our method. DFS is applied after the stated local-refinement stage for full-pipeline results.

Evaluation	Candidate budget	Move budget
Main benchmark, $n = 50$	A8S50	M20
Main benchmark, $n = 100$	A8S100	M20
CVRP100 tradeoff	A8S100 / A8S800	M5/12/20/30/50
CVRP200 tradeoff	A8S200 / A8S400 / A8S800	M5/12/20/30/50
Zero-shot $100 \rightarrow 200$	A8S100	M5/12/20
CVRP200 OOD	A8S200	M20
CVRP1000	16 / 800 trajectories	M20
Ablations, $n = 50$	A8S50 / A8Greedy1	M12 / M20

The CVRP200 OOD evaluation uses the same fine-tuned checkpoint as the main CVRP200 tradeoff curves; the inset evaluates zero-shot transfer from $n = 100$. Unless otherwise specified, all DFS experiments use a pool of up to 3,200 unique routes, with uniqueness defined by the served customer set. All experiments that use DFS impose a time limit of 10 seconds per instance. With bounded route pools, most searches terminate before reaching this limit. Route pools are built from complete refined candidates in stable ascending cost order. After the pool limit is reached, scanning continues to update retained customer sets with cheaper routes, as detailed in Appendix A.6.

B.4 Baseline Configurations

Neural baselines. The main benchmark uses publicly released checkpoints of RouteFinder, PoMtVRS (CaDA), CCL (ReLD), MoSES (CaDA), and RL-RFCS. For M20 and DFS, all methods use identical candidate counts, DP splitting, and subsequent post-processing settings, as described in Section 5.1. CoEKS (Yu et al., 2026) uses its publicly released $n = 50$ model without retraining; Appendix C reports its five-variant atomic comparison. Scale-Net-1k uses the authors' epoch-5500 checkpoint, obtained after 500 fine-tuning epochs on six 1,000-node VRP variants. CaR's evaluated refinement budgets are shown in Fig. 2; the OOD baseline results are taken from Omni-VRP.

Table 10 lists the official repositories for the reported methods. Omni-VRP supplies the OOD data and quoted reference results.

Table 10: Official repositories for the reported methods and OOD evaluation.

Method	Official repository
RouteFinder	`https://github.com/ai4co/routefinder`
PoMtVRS (CaDA)	`https://github.com/Regina921/PoMtVRS`
CCL (ReLD)	`https://github.com/gshuangchun/CCL-MTLVRP`
MoSES (CaDA)	`https://github.com/panyxy/moses_vrp`
RL-RFCS	`https://github.com/wenwenla/ijcai26-cvrp-solver`
CoEKS	`https://github.com/yuzikang0/CoEKS`
CaR	`https://github.com/jieyibi/CaR-constraint`
Scale-Net	`https://github.com/lsyysl9711/Scale-Net`
Omni-VRP	`https://github.com/RoyalSkye/Omni-VRP`

Classical solvers. We run HGS-CVRP v2.0.0 and PyVRP 0.13.4 with their default search parameters and per-instance time limits of $T \in \{1, 3, 10, 30, 60\}$ seconds. We additionally evaluate OR-Tools 9.15.6755 using the official RouteFinder adapter, Guided Local Search, and a per-instance time limit of 60 seconds. HGS-CVRP uses seed $1234 + i$ for instance i, whereas PyVRP uses seed 0 and uses time limit T; no random seed is explicitly specified for OR-Tools. For HGS-CVRP and

PyVRP, distances, demands, and vehicle capacities are multiplied by 1,000 and rounded to integers, while OR-Tools uses a scaling factor of 10^5. We recompute all reported objectives from the original coordinates. Instances are evaluated in parallel using at most 25 independent solver processes, with each process restricted to a single computational thread.

B.5 Evaluation Metrics and Timing Protocol

For variant v, let $\bar{C}_v$ and $\bar{C}_v^{\text{ref}}$ denote the mean objectives of the evaluated method and PyVRP@60s on the same test set. We compute $\text{Gap}_v = 100(\bar{C}_v - \bar{C}_v^{\text{ref}})/\bar{C}_v^{\text{ref}}$ and report the unweighted means of $\bar{C}_v$ and Gap_v across the 16 variants. The OOD table instead uses the corresponding HGS reference. Per-variant benchmark results appear in Appendix C.1.

Experiments run on Linux with an RTX 5090 GPU (32 GB) and Intel Xeon Platinum 8470Q CPUs; the container is allocated 25 CPU cores and 90 GB of memory. Neural latency measurements use one GPU, one CPU core, and batch size 1, and cover the complete input-to-output pipeline. We use the same timing procedure across all experiments conducted in this work. For each variant, we measure mean latency over 128 sequentially solved instances and report the unweighted average of these means across all 16 variants. Classical-solver points use their search-time budgets; their actual wall-clock times are slightly longer.

B.6 Ablation Configurations

All ablations use $n = 50$. The w/o Div. configuration removes both ROE terms; w/o RS removes only route-level diversity. The w/o REP configuration removes both REP supervision and its decision prior, with all other settings unchanged. The + AE configuration replaces ROE with action entropy while retaining REP.

Action-entropy objective and coefficient. At each decoding state $s_{b,k,t}$, we compute the categorical entropy of the action distribution and average over the batch and the $K = 32$ sampled trajectories. We then average over the actual number of decoding steps T (`entropy_steps`):

$$\bar{H} = \frac{1}{T}\sum_{t=1}^{T}\left[\frac{1}{BK}\sum_{b=1}^{B}\sum_{k=1}^{K} H(\pi_\theta(\cdot \mid s_{b,k,t}))\right], \qquad \mathcal{L}_{\text{AE}} = -\beta_{\text{AE}}\bar{H}. \tag{26}$$

This term is added to the training loss with ROE removed and REP retained. We screened $\beta_{\text{AE}} \in \{0.001, 0.003, 0.010\}$ through epoch E4 with shared initialization, $\lambda_E = 0.1$, and M5 refinement. We selected $\beta_{\text{AE}} = 0.003$ as a compromise between refined cost and action entropy based on these early training statistics, without using test-set performance for selection.

C Additional Experimental Results

C.1 Per-Variant Results on the 16-Variant Benchmark

Given the widespread use of the RouteFinder test protocol in the neural routing community, Tables 11 and 12 expand the aggregate results in Table 1 into per-variant mean objectives and gaps, providing a convenient reference for comparisons with other methods.

Table 11: Per-variant results at $n = 50$. Each entry is the mean objective (Gap to PyVRP@60s) over 1,000 instances. † denotes the original inference pipeline; M20 and DFS follow Table 1. All neural methods use A8S50.

Method	Stage	CVRP	OVRP	VRPB	OVRPB
PyVRP@60s	–	10.365 (0.000%)	6.507 (0.000%)	9.687 (0.000%)	6.898 (0.000%)
RouteFinder†	–	10.504 (1.337%)	6.684 (2.720%)	9.978 (3.006%)	7.071 (2.511%)
RouteFinder	M20	10.407 (0.398%)	6.541 (0.529%)	9.743 (0.580%)	6.924 (0.374%)
RouteFinder	DFS	10.391 (0.244%)	6.526 (0.294%)	9.726 (0.400%)	6.915 (0.237%)
PoMtVRS (CaDA)†	–	10.459 (0.905%)	6.640 (2.043%)	9.899 (2.191%)	7.021 (1.783%)
PoMtVRS (CaDA)	M20	10.401 (0.343%)	6.534 (0.422%)	9.735 (0.502%)	6.920 (0.317%)
PoMtVRS (CaDA)	DFS	10.390 (0.234%)	6.525 (0.273%)	9.724 (0.387%)	6.914 (0.233%)
CCL (ReLD)†	–	10.463 (0.945%)	6.611 (1.599%)	9.874 (1.933%)	6.992 (1.364%)
CCL (ReLD)	M20	10.398 (0.311%)	6.529 (0.347%)	9.725 (0.397%)	6.914 (0.230%)
CCL (ReLD)	DFS	10.385 (0.190%)	6.520 (0.204%)	9.714 (0.283%)	6.909 (0.159%)
MoSES (CaDA)†	–	10.462 (0.934%)	6.630 (1.887%)	9.904 (2.245%)	7.034 (1.967%)
MoSES (CaDA)	M20	10.402 (0.352%)	6.534 (0.419%)	9.740 (0.550%)	6.923 (0.360%)
MoSES (CaDA)	DFS	10.392 (0.259%)	6.525 (0.283%)	9.729 (0.433%)	6.916 (0.259%)
RL-RFCS†	DP	10.483 (1.137%)	6.602 (1.465%)	9.870 (1.892%)	6.984 (1.250%)
RL-RFCS	M20	10.404 (0.370%)	6.530 (0.354%)	9.736 (0.512%)	6.917 (0.266%)
RL-RFCS	DFS	10.391 (0.244%)	6.521 (0.226%)	9.725 (0.390%)	6.912 (0.196%)
Ours (Core)	M20	10.385 (0.192%)	6.515 (0.126%)	9.704 (0.176%)	6.903 (0.063%)
Ours (Full)	DFS	10.370 (0.048%)	6.508 (0.025%)	9.694 (0.072%)	6.900 (0.019%)

Method	Stage	VRPL	OVRPL	VRPTW	OVRPTW
PyVRP@60s	–	10.581 (0.000%)	6.507 (0.000%)	16.030 (0.000%)	10.510 (0.000%)
RouteFinder†	–	10.749 (1.587%)	6.686 (2.753%)	16.365 (2.092%)	10.652 (1.352%)
RouteFinder	M20	10.638 (0.536%)	6.541 (0.525%)	16.085 (0.342%)	10.526 (0.155%)
RouteFinder	DFS	10.621 (0.381%)	6.527 (0.303%)	16.075 (0.281%)	10.521 (0.108%)
PoMtVRS (CaDA)†	–	10.698 (1.106%)	6.639 (2.036%)	16.236 (1.286%)	10.593 (0.796%)
PoMtVRS (CaDA)	M20	10.626 (0.427%)	6.534 (0.420%)	16.075 (0.283%)	10.525 (0.147%)
PoMtVRS (CaDA)	DFS	10.615 (0.322%)	6.524 (0.268%)	16.069 (0.243%)	10.522 (0.116%)
CCL (ReLD)†	–	10.697 (1.101%)	6.610 (1.591%)	16.177 (0.917%)	10.565 (0.522%)
CCL (ReLD)	M20	10.624 (0.403%)	6.529 (0.338%)	16.065 (0.219%)	10.520 (0.100%)
CCL (ReLD)	DFS	10.611 (0.281%)	6.521 (0.211%)	16.059 (0.185%)	10.518 (0.076%)
MoSES (CaDA)†	–	10.703 (1.158%)	6.629 (1.882%)	16.262 (1.446%)	10.611 (0.962%)
MoSES (CaDA)	M20	10.630 (0.468%)	6.534 (0.413%)	16.077 (0.293%)	10.523 (0.123%)
MoSES (CaDA)	DFS	10.619 (0.361%)	6.525 (0.284%)	16.069 (0.248%)	10.519 (0.088%)
RL-RFCS†	DP	10.749 (1.593%)	6.601 (1.448%)	16.229 (1.246%)	10.571 (0.580%)
RL-RFCS	M20	10.640 (0.557%)	6.531 (0.368%)	16.086 (0.354%)	10.524 (0.132%)
RL-RFCS	DFS	10.626 (0.427%)	6.521 (0.222%)	16.080 (0.312%)	10.521 (0.109%)
Ours (Core)	M20	10.611 (0.283%)	6.515 (0.119%)	16.041 (0.072%)	10.513 (0.034%)
Ours (Full)	DFS	10.595 (0.133%)	6.509 (0.027%)	16.038 (0.052%)	10.513 (0.030%)

Continued on next page

Table 11 continued

Method	Stage	VRPBL	OVRPBL	VRPBTW	OVRPBTW
PyVRP@60s	–	10.183 (0.000%)	6.899 (0.000%)	18.291 (0.000%)	11.668 (0.000%)
RouteFinder†	–	10.578 (3.883%)	7.075 (2.543%)	18.600 (1.689%)	11.805 (1.176%)
RouteFinder	M20	10.265 (0.810%)	6.926 (0.392%)	18.345 (0.293%)	11.684 (0.141%)
RouteFinder	DFS	10.247 (0.630%)	6.917 (0.255%)	18.331 (0.218%)	11.680 (0.101%)
PoMtVRS (CaDA)†	–	10.494 (3.056%)	7.022 (1.781%)	18.478 (1.021%)	11.744 (0.653%)
PoMtVRS (CaDA)	M20	10.257 (0.732%)	6.921 (0.318%)	18.339 (0.261%)	11.684 (0.135%)
PoMtVRS (CaDA)	DFS	10.246 (0.620%)	6.916 (0.237%)	18.332 (0.222%)	11.681 (0.109%)
CCL (ReLD)†	–	10.440 (2.525%)	6.992 (1.348%)	18.418 (0.691%)	11.718 (0.429%)
CCL (ReLD)	M20	10.245 (0.608%)	6.915 (0.228%)	18.323 (0.174%)	11.678 (0.090%)
CCL (ReLD)	DFS	10.231 (0.474%)	6.910 (0.157%)	18.317 (0.144%)	11.677 (0.075%)
MoSES (CaDA)†	–	10.516 (3.277%)	7.036 (1.984%)	18.495 (1.112%)	11.760 (0.789%)
MoSES (CaDA)	M20	10.260 (0.754%)	6.924 (0.354%)	18.336 (0.245%)	11.682 (0.123%)
MoSES (CaDA)	DFS	10.247 (0.632%)	6.917 (0.258%)	18.328 (0.202%)	11.679 (0.094%)
RL-RFCS†	DP	10.472 (2.845%)	6.987 (1.278%)	18.467 (0.959%)	11.723 (0.468%)
RL-RFCS	M20	10.265 (0.809%)	6.918 (0.278%)	18.343 (0.282%)	11.682 (0.117%)
RL-RFCS	DFS	10.254 (0.696%)	6.913 (0.207%)	18.336 (0.246%)	11.680 (0.102%)
Ours (Core)	M20	10.214 (0.310%)	6.904 (0.069%)	18.305 (0.073%)	11.674 (0.050%)
Ours (Full)	DFS	10.202 (0.187%)	6.901 (0.022%)	18.303 (0.064%)	11.673 (0.044%)

Method	Stage	VRPLTW	OVRPLTW	VRPBLTW	OVRPBLTW
PyVRP@60s	–	16.353 (0.000%)	10.510 (0.000%)	18.586 (0.000%)	11.668 (0.000%)
RouteFinder†	–	16.762 (2.500%)	10.652 (1.359%)	18.941 (1.909%)	11.804 (1.166%)
RouteFinder	M20	16.431 (0.474%)	10.526 (0.155%)	18.664 (0.419%)	11.684 (0.140%)
RouteFinder	DFS	16.413 (0.366%)	10.521 (0.107%)	18.644 (0.310%)	11.680 (0.100%)
PoMtVRS (CaDA)†	–	16.634 (1.716%)	10.593 (0.797%)	18.814 (1.227%)	11.744 (0.654%)
PoMtVRS (CaDA)	M20	16.420 (0.411%)	10.525 (0.146%)	18.650 (0.343%)	11.684 (0.140%)
PoMtVRS (CaDA)	DFS	16.409 (0.341%)	10.522 (0.114%)	18.639 (0.285%)	11.681 (0.114%)
CCL (ReLD)†	–	16.554 (1.226%)	10.564 (0.517%)	18.759 (0.929%)	11.718 (0.427%)
CCL (ReLD)	M20	16.409 (0.342%)	10.519 (0.092%)	18.636 (0.265%)	11.678 (0.089%)
CCL (ReLD)	DFS	16.399 (0.279%)	10.517 (0.071%)	18.626 (0.211%)	11.677 (0.074%)
MoSES (CaDA)†	–	16.669 (1.930%)	10.612 (0.974%)	18.854 (1.441%)	11.761 (0.797%)
MoSES (CaDA)	M20	16.426 (0.448%)	10.523 (0.128%)	18.657 (0.382%)	11.683 (0.127%)
MoSES (CaDA)	DFS	16.413 (0.364%)	10.520 (0.095%)	18.642 (0.298%)	11.680 (0.100%)
RL-RFCS†	DP	16.644 (1.779%)	10.571 (0.586%)	18.843 (1.379%)	11.723 (0.468%)
RL-RFCS	M20	16.448 (0.581%)	10.524 (0.136%)	18.669 (0.445%)	11.682 (0.121%)
RL-RFCS	DFS	16.438 (0.521%)	10.521 (0.110%)	18.659 (0.389%)	11.680 (0.102%)
Ours (Core)	M20	16.375 (0.132%)	10.513 (0.034%)	18.604 (0.095%)	11.673 (0.047%)
Ours (Full)	DFS	16.366 (0.079%)	10.513 (0.030%)	18.599 (0.068%)	11.673 (0.044%)

Table 12: Per-variant results at $n = 100$. Each entry is the mean objective (Gap to PyVRP@60s) over 1,000 instances. † denotes the original inference pipeline; M20 and DFS follow Table 1. All neural methods use A8S100.

Method	Stage	CVRP	OVRP	VRPB	OVRPB
PyVRP@60s	–	15.560 (0.000%)	9.727 (0.000%)	14.381 (0.000%)	10.342 (0.000%)
RouteFinder†	–	15.856 (1.902%)	10.121 (4.051%)	14.942 (3.903%)	10.771 (4.150%)
RouteFinder	M20	15.702 (0.912%)	9.864 (1.411%)	14.586 (1.425%)	10.468 (1.222%)
RouteFinder	DFS	15.622 (0.398%)	9.784 (0.581%)	14.494 (0.788%)	10.399 (0.560%)
PoMtVRS (CaDA)†	–	15.768 (1.337%)	10.034 (3.151%)	14.806 (2.959%)	10.657 (3.053%)
PoMtVRS (CaDA)	M20	15.662 (0.653%)	9.835 (1.108%)	14.532 (1.055%)	10.427 (0.831%)
PoMtVRS (CaDA)	DFS	15.598 (0.246%)	9.774 (0.478%)	14.458 (0.540%)	10.381 (0.386%)
CCL (ReLD)†	–	15.789 (1.468%)	10.010 (2.905%)	14.800 (2.914%)	10.621 (2.707%)
CCL (ReLD)	M20	15.669 (0.697%)	9.830 (1.054%)	14.534 (1.065%)	10.423 (0.787%)
CCL (ReLD)	DFS	15.605 (0.291%)	9.767 (0.413%)	14.463 (0.576%)	10.375 (0.328%)
MoSES (CaDA)†	–	15.834 (1.762%)	10.082 (3.648%)	14.902 (3.625%)	10.726 (3.714%)
MoSES (CaDA)	M20	15.697 (0.878%)	9.858 (1.346%)	14.580 (1.385%)	10.458 (1.122%)
MoSES (CaDA)	DFS	15.628 (0.436%)	9.786 (0.609%)	14.496 (0.805%)	10.399 (0.553%)
RL-RFCS†	DP	15.864 (1.953%)	10.016 (2.974%)	14.854 (3.289%)	10.659 (3.074%)
RL-RFCS	M20	15.681 (0.776%)	9.807 (0.824%)	14.535 (1.072%)	10.417 (0.726%)
RL-RFCS	DFS	15.595 (0.224%)	9.748 (0.214%)	14.445 (0.448%)	10.365 (0.223%)
Ours (Core)	M20	15.672 (0.719%)	9.786 (0.606%)	14.474 (0.652%)	10.388 (0.445%)
Ours (Full)	DFS	15.570 (0.061%)	9.729 (0.016%)	14.390 (0.068%)	10.342 (0.005%)

Method	Stage	VRPL	OVRPL	VRPTW	OVRPTW
PyVRP@60s	–	15.717 (0.000%)	9.727 (0.000%)	25.415 (0.000%)	16.924 (0.000%)
RouteFinder†	–	16.050 (2.117%)	10.121 (4.052%)	26.234 (3.223%)	17.327 (2.380%)
RouteFinder	M20	15.885 (1.066%)	9.864 (1.417%)	25.615 (0.785%)	16.991 (0.397%)
RouteFinder	DFS	15.802 (0.541%)	9.785 (0.604%)	25.526 (0.434%)	16.954 (0.174%)
PoMtVRS (CaDA)†	–	15.951 (1.490%)	10.033 (3.146%)	25.976 (2.207%)	17.175 (1.482%)
PoMtVRS (CaDA)	M20	15.837 (0.763%)	9.834 (1.105%)	25.573 (0.621%)	16.982 (0.343%)
PoMtVRS (CaDA)	DFS	15.769 (0.332%)	9.772 (0.470%)	25.522 (0.419%)	16.960 (0.211%)
CCL (ReLD)†	–	15.981 (1.678%)	10.025 (3.064%)	25.882 (1.838%)	17.115 (1.131%)
CCL (ReLD)	M20	15.850 (0.844%)	9.831 (1.074%)	25.549 (0.526%)	16.965 (0.243%)
CCL (ReLD)	DFS	15.778 (0.388%)	9.767 (0.411%)	25.495 (0.313%)	16.948 (0.140%)
MoSES (CaDA)†	–	16.025 (1.962%)	10.081 (3.646%)	26.032 (2.428%)	17.220 (1.747%)
MoSES (CaDA)	M20	15.880 (1.039%)	9.858 (1.354%)	25.576 (0.634%)	16.977 (0.313%)
MoSES (CaDA)	DFS	15.805 (0.562%)	9.785 (0.604%)	25.507 (0.362%)	16.950 (0.150%)
RL-RFCS†	DP	16.089 (2.364%)	10.017 (2.984%)	26.030 (2.419%)	17.147 (1.318%)
RL-RFCS	M20	15.878 (1.027%)	9.807 (0.830%)	25.592 (0.696%)	16.974 (0.294%)
RL-RFCS	DFS	15.778 (0.390%)	9.748 (0.217%)	25.531 (0.455%)	16.952 (0.167%)
Ours (Core)	M20	15.865 (0.939%)	9.786 (0.607%)	25.508 (0.366%)	16.943 (0.109%)
Ours (Full)	DFS	15.746 (0.184%)	9.729 (0.024%)	25.441 (0.101%)	16.930 (0.035%)

Continued on next page

Table 12 continued

Method	Stage	VRPBL	OVRPBL	VRPBTW	OVRPBTW
PyVRP@60s	–	14.776 (0.000%)	10.342 (0.000%)	29.458 (0.000%)	19.154 (0.000%)
RouteFinder†	–	15.529 (5.095%)	10.779 (4.221%)	30.243 (2.663%)	19.550 (2.069%)
RouteFinder	M20	15.060 (1.920%)	10.470 (1.235%)	29.646 (0.638%)	19.225 (0.374%)
RouteFinder	DFS	14.940 (1.110%)	10.401 (0.570%)	29.555 (0.330%)	19.186 (0.170%)
PoMtVRS (CaDA)†	–	15.351 (3.892%)	10.659 (3.063%)	29.981 (1.774%)	19.394 (1.254%)
PoMtVRS (CaDA)	M20	14.988 (1.434%)	10.428 (0.831%)	29.608 (0.507%)	19.214 (0.317%)
PoMtVRS (CaDA)	DFS	14.896 (0.811%)	10.382 (0.388%)	29.553 (0.320%)	19.190 (0.191%)
CCL (ReLD)†	–	15.325 (3.719%)	10.626 (2.743%)	29.890 (1.464%)	19.331 (0.928%)
CCL (ReLD)	M20	14.979 (1.374%)	10.423 (0.780%)	29.576 (0.399%)	19.195 (0.214%)
CCL (ReLD)	DFS	14.892 (0.784%)	10.376 (0.329%)	29.523 (0.220%)	19.176 (0.118%)
MoSES (CaDA)†	–	15.478 (4.751%)	10.724 (3.687%)	30.053 (2.018%)	19.441 (1.502%)
MoSES (CaDA)	M20	15.041 (1.792%)	10.460 (1.139%)	29.613 (0.527%)	19.211 (0.300%)
MoSES (CaDA)	DFS	14.940 (1.113%)	10.399 (0.547%)	29.545 (0.295%)	19.181 (0.143%)
RL-RFCS†	DP	15.449 (4.555%)	10.663 (3.099%)	30.039 (1.971%)	19.371 (1.136%)
RL-RFCS	M20	15.006 (1.555%)	10.417 (0.722%)	29.609 (0.512%)	19.200 (0.240%)
RL-RFCS	DFS	14.894 (0.800%)	10.366 (0.228%)	29.547 (0.301%)	19.179 (0.134%)
Ours (Core)	M20	14.933 (1.065%)	10.389 (0.457%)	29.535 (0.260%)	19.174 (0.105%)
Ours (Full)	DFS	14.823 (0.318%)	10.343 (0.003%)	29.484 (0.086%)	19.162 (0.045%)

Method	Stage	VRPLTW	OVRPLTW	VRPBLTW	OVRPBLTW
PyVRP@60s	–	25.740 (0.000%)	16.924 (0.000%)	29.793 (0.000%)	19.154 (0.000%)
RouteFinder†	–	26.691 (3.695%)	17.329 (2.392%)	30.688 (3.005%)	19.552 (2.079%)
RouteFinder	M20	26.020 (1.088%)	16.990 (0.390%)	30.057 (0.887%)	19.225 (0.370%)
RouteFinder	DFS	25.893 (0.594%)	16.953 (0.173%)	29.925 (0.442%)	19.186 (0.165%)
PoMtVRS (CaDA)†	–	26.433 (2.691%)	17.174 (1.478%)	30.429 (2.134%)	19.395 (1.256%)
PoMtVRS (CaDA)	M20	25.977 (0.919%)	16.982 (0.344%)	30.016 (0.749%)	19.214 (0.311%)
PoMtVRS (CaDA)	DFS	25.896 (0.605%)	16.960 (0.210%)	29.931 (0.464%)	19.190 (0.188%)
CCL (ReLD)†	–	26.339 (2.326%)	17.112 (1.111%)	30.328 (1.795%)	19.333 (0.933%)
CCL (ReLD)	M20	25.950 (0.814%)	16.964 (0.238%)	29.979 (0.626%)	19.196 (0.218%)
CCL (ReLD)	DFS	25.861 (0.468%)	16.947 (0.138%)	29.898 (0.354%)	19.177 (0.122%)
MoSES (CaDA)†	–	26.490 (2.911%)	17.221 (1.754%)	30.511 (2.411%)	19.441 (1.498%)
MoSES (CaDA)	M20	25.987 (0.957%)	16.978 (0.320%)	30.025 (0.780%)	19.210 (0.294%)
MoSES (CaDA)	DFS	25.878 (0.535%)	16.951 (0.156%)	29.919 (0.423%)	19.181 (0.144%)
RL-RFCS†	DP	26.564 (3.200%)	17.147 (1.315%)	30.558 (2.567%)	19.370 (1.130%)
RL-RFCS	M20	26.025 (1.106%)	16.973 (0.291%)	30.047 (0.852%)	19.200 (0.240%)
RL-RFCS	DFS	25.923 (0.710%)	16.953 (0.172%)	29.939 (0.491%)	19.179 (0.130%)
Ours (Core)	M20	25.913 (0.672%)	16.941 (0.102%)	29.936 (0.481%)	19.173 (0.101%)
Ours (Full)	DFS	25.794 (0.208%)	16.930 (0.034%)	29.843 (0.168%)	19.162 (0.042%)

Some open-route variants with and without distance limits have nearly identical results, including PyVRP references; rounding can obscure these small differences. Our core pipeline leads the neural baselines on all but CVRP100 and VRPL100, where PoMtVRS has lower objectives, while our full pipeline achieves the lowest neural objectives across all 32 size–variant combinations. Route recombination yields larger macro-average gains at $n = 100$; small gains on some $n = 50$ variants are not visible at the displayed precision.

C.2 Comparison with CoEKS on Atomic VRP Variants

Table 13: Solution quality on the five atomic VRP variants at $n = 50$. Neural solvers use A8S50, matching prior work (eight augmentations with 50 candidates each). DFS route recombination uses a pool of up to 3,200 unique routes. † denotes the original inference pipeline of each baseline. Bold indicates the best solution quality among learning-based methods.

Method	CVRP	OVRP	VRPB	VRPL	VRPTW
PyVRP@60s	10.365 (0.00)	6.507 (0.00)	9.687 (0.00)	10.581 (0.00)	16.030 (0.00)
CoEKS†	10.464 (0.95)	6.648 (2.16)	9.930 (2.51)	10.712 (1.24)	16.360 (2.06)
CoEKS+M20	10.396 (0.30)	6.531 (0.38)	9.731 (0.46)	10.623 (0.40)	16.076 (0.29)
CoEKS+DFS	10.383 (0.17)	6.520 (0.20)	9.715 (0.29)	10.609 (0.26)	16.066 (0.23)
Ours+M20	10.385 (0.19)	6.515 (0.13)	9.704 (0.18)	10.611 (0.28)	16.041 (0.07)
Ours+DFS	**10.370 (0.05)**	**6.508 (0.03)**	**9.694 (0.07)**	**10.595 (0.13)**	**16.038 (0.05)**

C.3 Sensitivity to the Diversity-Credit Candidate Fraction

We vary the quality-gate fraction $\gamma \in \{0.25, 0.50, 1.00\}$ used for ROE credit assignment. At $\gamma = 1$, all candidates participate, removing quality-based filtering while retaining both diversity terms and their centering and scaling. Table 14 reports the resulting objectives and refined-outcome diversity.

Table 14: Sensitivity to the quality-gate fraction γ at $n = 50$ with A8S50. Diversity is measured after M20; the unique-endpoint ratio is the number of distinct endpoints divided by the 400 candidates. Bold marks the lowest objective or highest diversity in each column.

	Objective ↓			Diversity after M20 ↑		
γ	M12	M20	DFS	Unique endpoints	Endpoint ratio	Unique routes
25%	**11.347212**	**11.340532**	11.334769	220.07	55.02%	330.59
50%	11.351545	11.340967	11.334858	232.93	58.23%	357.02
100%	11.353181	11.340909	**11.334568**	**239.20**	**59.80%**	**377.36**

Increasing γ increases both the number of distinct endpoints and the number of distinct routes in these runs. The default $\gamma = 0.25$ gives the lowest M12 and M20 objectives, whereas $\gamma = 1$ gives the lowest DFS objective. The objective ranges across the three settings are 0.005969 at M12, 0.000435 at M20, and 0.000290 after DFS. Thus, these results show small numerical differences in objectives, with no single fraction performing best at all stages; they do not establish a consistent solution-quality benefit from quality-based filtering. We retain $\gamma = 0.25$ as the default implementation setting and treat it as a credit-assignment choice within ROE.

C.4 Zero-Shot Generalization to Unseen Constraint Combinations

Training and evaluation split. We train an $n = 50$ model on six representative tasks: CVRP, OVRP, VRPB, VRPL, VRPTW, and OVRPTW (Six-task). This set contains the five atomic variants and one combined variant, OVRPTW. The remaining ten variants form the unseen evaluation set: OVRPB, OVRPL, VRPBL, VRPBTW, VRPLTW, OVRPBL, OVRPBTW, OVRPLTW, VRPBLTW, and OVRPBLTW. All constituent constraints occur during training; the zero-shot setting concerns their unseen combinations. We evaluate the same final checkpoint on all 16 variants without further training and compare it with the model trained on all 16 tasks (Full-16). Seen-6 and Unseen-10 refer to the Six-task training split; Full-16 is trained on both groups.

Protocol. Both models use the same random initialization, architecture, training seed (3333), total training budget of 250,000 updates, M5-to-M12 training schedule, and REP and ROE weights. Each update samples a task uniformly from the respective training set. Evaluation uses the same 1,000 instances per variant, A8S50 (400 candidates), sampling temperature 1.6, and inference seed 19971023. Each refinement budget starts independently from the same sampled permutations after DP splitting. DFS pools routes from all M20 candidates, with a limit of 3,200 unique routes and a 10-second search budget per instance. For each variant, we compute the gap from its mean objective relative to the corresponding PyVRP@60s mean, then average these gaps equally over variants within each evaluation group.

Table 15: Six-task training and zero-shot evaluation at $n = 50$ with A8S50. Entries are macro-average gaps to PyVRP@60s; lower is better. Bold marks the lower numerical gap within each evaluation group and stage.

Evaluation set	Model	M12 Gap	M20 Gap	M20+DFS Gap
Seen-6	Six-task	**0.1758%**	**0.1361%**	**0.0590%**
	Full-16	0.1869%	0.1471%	0.0601%
Unseen-10	Six-task	0.9212%	0.2735%	0.1076%
	Full-16	**0.1601%**	**0.0992%**	**0.0586%**

Results. On Seen-6, Six-task has slightly lower numerical gaps at all three stages, but the difference after DFS is only 0.0011 percentage points (Table 15). On Unseen-10, its gap decreases from 0.9212% at M12 to 0.2735% at M20 and 0.1076% after DFS. The difference from Full-16 correspondingly narrows from 0.7611 to 0.1743 and 0.0490 percentage points, but Full-16 retains lower gaps throughout. These results support zero-shot transfer of the complete construction-and-post-processing pipeline to unseen constraint combinations, with further refinement and route recombination reducing the performance difference. They do not establish lossless generalization by the neural constructor or equivalence to training on all 16 tasks.

C.5 Results on a CVRPLIB X-Set Subset

Table 16 reports results on 22 instances from the CVRPLIB X set (Uchoa et al., 2017), from X-n101-k25 to X-n200-k36. N100 and N200 denote the same $n = 100$ trained and $n = 200$ fine-tuned models used in the main experiments, respectively. Both use A8S12800 followed by M20 refinement and DFS route recombination with a pool of up to 2,500 unique routes per instance. For this benchmark, reported objectives follow the CVRPLIB/TSPLIB `EUC_2D` convention: Euclidean distances computed from the original coordinates are rounded to the nearest integer for each edge before summation. DP splitting and GPU local refinement both use the precomputed integer distance matrix divided by 1,000, denoted by $\widetilde{d}_{ij} = d_{ij}^{\mathrm{EUC_2D}}/1000$, without recomputing distances from normalized coordinates. Before route recombination, each route cost is recomputed as $c_r = \mathrm{round}(1000 \sum_{(i,j)\in r} \widetilde{d}_{ij})$. HiGHS and DFS use these restored integer route costs for the LP bound and route selection, including pruning, without accessing coordinates. The scaling keeps GPU numerical magnitudes suitable while preserving the benchmark objective throughout the pipeline. We also recompute the cost of each official reference solution using the integer distance matrix and require exact agreement with the optimum recorded in its `.sol` file; evaluation aborts on any mismatch. Times cover the complete inference pipeline; differences in runtime arise primarily from DFS.

Table 16: Results on 22 CVRPLIB X-set instances. M20 and DFS columns report objectives, with DFS gaps to the optimum in parentheses; times are in seconds. Underlined DFS objectives attain the known optimum. The final row reports average gaps for the objective columns and mean times.

		N100			N200		
Instance	Opt.	M20	DFS (Gap)	Time	M20	DFS (Gap)	Time
X-n101-k25	27,591	28,432	27,591 (0.000%)	12.035	29,229	27,609 (0.065%)	8.039
X-n106-k14	26,362	26,632	26,425 (0.239%)	18.769	26,609	26,531 (0.641%)	18.583
X-n110-k13	14,971	14,986	14,971 (0.000%)	7.125	15,030	14,971 (0.000%)	7.645
X-n115-k10	12,747	12,760	12,747 (0.000%)	7.760	12,753	12,747 (0.000%)	8.156
X-n120-k6	13,332	13,340	13,332 (0.000%)	7.973	13,367	13,332 (0.000%)	8.527
X-n125-k30	55,539	58,518	58,169 (4.735%)	19.519	58,929	56,752 (2.184%)	19.806
X-n129-k18	28,940	29,322	29,322 (1.320%)	18.823	29,485	29,400 (1.589%)	18.792
X-n134-k13	10,916	11,106	10,944 (0.257%)	9.435	11,131	10,931 (0.137%)	9.194
X-n139-k10	13,590	13,697	13,590 (0.000%)	8.749	13,692	13,614 (0.177%)	9.297
X-n143-k7	15,700	15,903	15,742 (0.268%)	9.246	15,934	15,741 (0.261%)	9.490
X-n148-k46	43,448	47,426	46,324 (6.619%)	20.941	49,454	45,408 (4.511%)	20.980
X-n153-k22	21,220	22,608	21,860 (3.016%)	21.460	23,606	23,262 (9.623%)	22.103
X-n157-k13	16,876	17,079	16,936 (0.356%)	10.822	16,993	16,901 (0.148%)	10.324
X-n162-k11	14,138	14,336	14,180 (0.297%)	11.569	14,289	14,173 (0.248%)	11.541
X-n167-k10	20,557	20,950	20,625 (0.331%)	11.196	20,687	20,592 (0.170%)	10.788
X-n172-k51	45,607	50,322	47,106 (3.287%)	25.002	51,863	47,365 (3.855%)	24.604
X-n176-k26	47,812	50,973	49,645 (3.834%)	24.624	51,137	49,098 (2.690%)	24.646
X-n181-k23	25,569	25,962	25,631 (0.242%)	19.919	25,880	25,622 (0.207%)	22.734
X-n186-k15	24,145	24,828	24,375 (0.953%)	14.704	24,388	24,211 (0.273%)	13.776
X-n190-k8	16,980	17,309	17,302 (1.896%)	16.121	17,188	17,113 (0.783%)	14.537
X-n195-k51	44,225	50,029	46,333 (4.767%)	26.077	53,017	50,114 (13.316%)	26.813
X-n200-k36	58,578	61,982	60,543 (3.355%)	26.745	62,904	61,188 (4.456%)	26.747
Macro-average	–	3.511%	1.626%	15.846	4.483%	2.061%	15.778

Route recombination reduces the macro-average gap from 3.511% to 1.626% for N100 and from 4.483% to 2.061% for N200. N100 reaches the optimum on five instances, while N200 does so on three. Although N200 obtains lower DFS objectives on 11 instances, compared with eight for N100 and three ties, N100 has the lower macro-average gap. Performance varies across instances: for example, N200 improves the DFS gap on X-n125-k30 from 4.735% to 2.184%, but has a larger gap on X-n195-k51 (13.316% versus 4.767%).