RouteFinder
Towards Foundation Models for Vehicle Routing Problems

Federico Berto [*1] **Chuanbo Hua** [*1] **Nayeli Gast Zepeda** [*2] **André Hottung** [2] **Niels Wouda** [3] **Leon Lan** [4]
Kevin Tierney [2] **Jinkyoo Park** [1 5]

Abstract

This paper introduces `RouteFinder`, a framework for developing foundation models for Vehicle Routing Problems (VRPs). Our key idea is that a foundation model for VRPs should be able to model variants by treating each variant as a subset of a larger VRP problem, equipped with different attributes. We introduce a parallelized environment to handle any combination of attributes simultaneously in a batched manner and an efficient sampling procedure to train on a mix of problems at each optimization step, greatly improving convergence robustness. We also introduce novel Global Feature Embeddings that project instance-wise attributes efficiently onto the latent space and help the model understand different VRP variants. Finally, we introduce Efficient Adapter Layers, a simple-yet-effective technique to finetune pretrained `RouteFinder` models to solve novel variants with previously *unseen* attributes, outside of the original feature space. We validate our approach through extensive experiments on 24 VRP variants, demonstrating competitive results over recent multi-task learning models. We make our code openly available at https://github.com/ai4co/routefinder.

1. Introduction

Vehicle Routing Problems (VRPs) are an important class of Combinatorial Optimization (CO) problems that have received much attention in Operations Research (OR) and Computer Science. Since the VRP is an NP-hard problem, finding an optimal solution by exhaustively exploring the solution space is often computationally expensive, and impractical for large instances. Instead, heuristic methods that quickly generate good (but possibly suboptimal) solutions are commonly used to solve large-scale VRPs. The OR community has developed many VRP heuristics, including the well-known Lin-Kernighan-Helsgaun (LKH) heuristic (Helsgaun, 2017), Fast Iterated Local Optimization (FILO) (Accorsi and Vigo, 2021; 2024) and Hybrid Genetic Search (HGS) (Vidal, 2022; Wouda et al., 2024).

Recently, Neural Combinatorial Optimization (NCO) approaches have been developed to solve CO problems. By leveraging deep learning, these approaches seek to learn and generalize from data, potentially providing more flexible and scalable solutions (Kool et al., 2019; Hottung and Tierney, 2019; Kwon et al., 2020; Kim et al., 2022; Berto et al., 2024; Hottung et al., 2024). Similar to how the developments in Natural Language Processing (NLP) have resulted in Large Language Models (LLMs), research efforts in solving CO problems through machine learning are also going towards foundation models (Liu et al., 2024b; Ye et al., 2024a; Liu et al., 2024a; Zhou et al., 2024). A foundation model for VRPs would have important implications in terms of cost savings for companies and organizations while, most importantly, being *adaptable* to new requirements (constraints) outside of the training distribution.

In this work, we introduce `RouteFinder`, a foundation model framework for VRPs. Our contributions are:

- We introduce `RouteFinder`, a general framework to solve any combination of VRP variants via a unified environment handling any number of attributes.
- We propose Mixed Batch Training to sample diverse variants at the same optimization step to concurrently train multiple tasks to enhance performance.
- We introduce Global Attribute Embeddings to model shared instance features, enabling the model to understand and differentiate among various VRP variants.
- We present Efficient Adapter Layers to finetune pretrained `RouteFinder` models to adapt to novel VRP variants with any number of *new* attributes.

[*]Equal contributions. [‡]The authors of this paper are members of the AI4CO open research community. [1]KAIST [2]Bielefeld University [3]University of Groningen [4]VU Amsterdam [5]OMELET. Correspondence to: Federico Berto <fberto@kaist.ac.kr>.

Published at ICML 2024 Workshop on Foundation Models in the Wild.

- We validate `RouteFinder` in extensive experiments on 24 VRP variants demonstrating competitive performance against recent multi-task learning methods.

2. Related Works

Multi-task learning for VRPs In this work we develop a unified VRP solver that can readily be generalized to any number of VRP variants. This issue of generalization has garnered much attention recently. Wang and Yu (2023) introduces a multi-armed bandit method that solves several VRP variants with limited training budgets. Lin et al. (2024) proposes training a *backbone* model (i.e. deep layers) for VRPs that can then be adapted via low-dimensional layers such as linear projections to efficiently fine-tune different problems. Most related to us are the recent works of Liu et al. (2024a) and Zhou et al. (2024), which use attribute composition (Ruis et al., 2021) to achieve (zero-shot) generalization on several VRP variants. Liu et al. (2024a) builds on the Reinforcement-Learning-based POMO (Kwon et al., 2020), on top of which Zhou et al. (2024) employ a mixture-of-experts model to improve generalization. We include additional literature about more general VRPs in Appendix D.

3. Vehicle Routing Problems

The Capacitated VRP (CVRP) is formulated on a graph $G = (N, E)$, where $N = \{0, 1, \dots, n\}$ represents the set of nodes, with 0 denoting the depot and $N_c = \{1, \dots, n\}$ representing the customers. Each customer $i \in N_c$ has a demand q_i. The edges E connect pairs of nodes, and each edge $(i, j) \in E$ has a travel cost c_{ij} (e.g., distance or travel duration). A fleet of vehicles, each with a capacity Q, departs from the depot to serve each of the customers exactly once and returns, with the objective of minimizing the total travel cost.

Following (Vidal et al., 2014), we consider a collection of VRP *variants* that each consist of one or more *attributes*, resulting in a rich set of routing problems with practical relevance. Each of these variants offers a unique generalization task for `RouteFinder`. Table A.1 in the Appendix provides a list of all 24 VRP variants we consider in this paper. We divide the attributes we consider into *node attributes*, *global attributes*, and *edge attributes*. *Node attributes* are specific to the depot and customer nodes and local to specific nodes, for example (linehaul) demands, backhaul demands, time windows, and service durations. *Global attributes* represent structural aspects of the problem as a whole, e.g., open vs. closed routes, distance limits, and the type of backhaul. In this work, the only relevant *edge attributes* we consider is the travel cost of each edge, which represents a distance or travel duration depending on the problem definition. Fig. A.1 describes node and global attributes modeled in this work. We defer to Appendix A.1 additional details.

4. The `RouteFinder` Recipe

We leverage attribute composition from Liu et al. (2024a); Zhou et al. (2024) to solve multiple VRP variants, and we similarly employ autoregressive models trained with REINFORCE (Sutton et al., 1999) with the POMO (Kwon et al., 2020) baseline (see Appendix B). Attribute composition treats different variants of the VRP as combinations of fundamental attributes (see Appendix A.1), using a common network to learn their representations. We go one step further than previous works and consider different combinations of attributes *within* training batches (see Section 4.2). Fig. 4.1 provides an overview of `RouteFinder`'s architecture.

4.1. Unified VRP Environment

In previous works proposing multi-task learning across VRP variants, like MTPOMO (Liu et al., 2024a) and MVMoE (Zhou et al., 2024), the training scheme samples an instance variant (CVRP, VRPTW, etc.) out of the set of available variants during training. Every instance within that batch, therefore, is of the same problem category. We hypothesize that this leads to a bias in training: instead of learning a broad set of problems from the beginning, the model learns individual problems, potentially overlooking interrelations between them. We therefore propose to learn across problems from the beginning and include problem instances of various attributes within each training batch.

We define an environment capable of modeling all of the previously discussed VRP attributes simultaneously, essentially building an OVRPBLTW environment: an open route vehicle routing problem with linehauls, backhauls, distance limits, and time windows. The environment supports subsets of the OVRPBLTW defining other VRP variants, i.e., some attributes can be "turned off." For example, if an instance should not have time windows, the time windows are all set to $[0, \infty]$. In this way, all attributes characterizing a VRP variant can simply be turned "on" and "off", allowing us to model up to 16 different problem types with one single environment. This approach can be easily and flexibly extended (e.g., by including different location sampling mechanisms, backhaul classes, etc.), allowing for even more problem variants to be modeled with the same environment in the future.

4.2. Variant Sampling for Mixed Batch Training

Optimizing a neural solver for tackling multiple tasks requires careful consideration of its training scheme, which needs to be robust against different variant distributions.

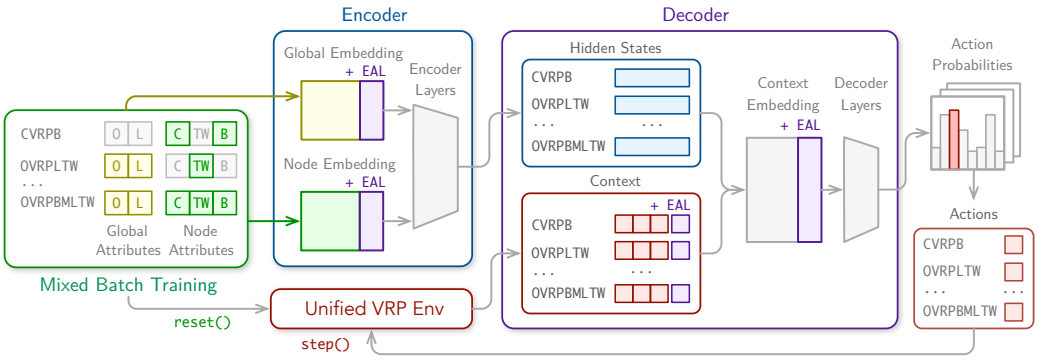

Figure 4.1: Overview of `RouteFinder`.

We introduce a flexible approach which we coin Mixed Batch Training (MBT) to efficiently reuse a single dataset to generate multiple problem variants, optimizing data storage and processing capabilities. We observe that the OVRPBLTW problem variant is the most general problem variant we study in this paper, and can be used to generate any of the other variants by selectively removing the (O), (B), (L), or (TW) attributes (note that for zero-shot generalization and few-shot learning, we will additionally sample with the mixed (M) backhaul attribute in Section 5.3). Denote by $\boldsymbol{X}$ a dataset of OVRPBLTW problem instances, and let V be the set of attributes, where each attribute $\nu \in V$ is associated with a sampling probability $\mathbf{p}_\nu$. For each instance $x \in \boldsymbol{X}$, we can write $x((\mathbf{1}_1)_{\nu \in V})$ to conveniently express using indicator functions $\mathbf{1}_1$ for each attribute $\nu \in V$ that the instance x is equipped with ν. The sampling procedure of MBT can be defined as follows:

$$\boldsymbol{X}_{\text{subsampled}} = \{x((\mathbf{1}_{\text{rand}(0,1)<\mathbf{p}_\nu})_{\nu \in V})\}_{x \in \boldsymbol{X}},$$

where $\text{rand}(0,1)$ draws an independent sample from $U[0,1]$. For example, to sample uniformly across all problem variants, we could set $\mathbf{p}_\nu = \frac{1}{2}$ for each $\nu \in V$. MBT is flexible and scalable, capable of adapting to any problem where different constraints or features might be selectively activated or deactivated. Fig. 4.2 provides an overview of MBT.

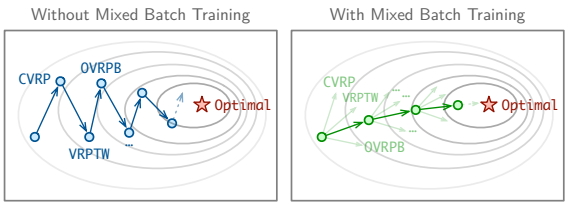

Figure 4.2: **[Left]** Training without MBT may lead to instability since at each step the optimization is biased toward a single task. **[Right]** Training `RouteFinder` with MBT allows for more stable training.

4.3. Global Attribute Embeddings

Global attributes as outlined in Appendix A.1 are essential for modeling VRPs; for instance, given an open (O) attribute, the solver may find optimal routes that do not necessarily loop back to the starting depot. Previous multi-task learning models for VRPs (Liu et al., 2024a; Zhou et al., 2024) project such features on the shallow decoder as dynamic features. However, such a design can be sub-optimal since the deep transformer layers carry out most of the learning and, importantly, can enable effective attribute mixing, which is essential in understanding a (new) problem. We thus design Global Attribute Embeddings for effective problem representation, which incorporate problem variants and help the deep layers understand which problem is being faced. Global attributes $\phi_0, \ldots, \phi_k$ are projected via a projection layer:

$$h_g^0 = f_\theta([\phi_0, \ldots, \phi_k]), \quad f_\theta : \mathbb{R}^k \to \mathbb{R}^d$$

into d-dimensional space. Given our unified VRP representation, some attributes, such as the duration limit l for unconstrained VRPs, might be ∞. Such attributes get transformed to 0, not influencing the calculations. The embedding is then processed in deep transformer layers.

4.4. Efficient Adapter Layers: Finetuning to Unseen Attributes

Wang and Yu (2023) and Lin et al. (2024) propose pretraining a backbone solver, on top of which problem-specific layers can be applied. However, doing so excludes previous knowledge accumulated in the projection layers, complicating optimization. We propose Efficient Adapter Layers (EAL), an effective approach for few-shot learning for VRP foundation models.

Consider a linear projection layer $\mathbf{W} \in \mathbb{R}^{k \times d}$ as the original weight matrix for the projection from the raw attribute to latent space, where k is the number of attributes and d is the hidden dimension. In this work, for simplicity, we consider unbiased linear projections to the latent space. This

can be trivially extended to general affine projections using a bias term. To accommodate l new attributes, EAL augments $\mathbf{W}$ with zeros. The new matrix $\mathbf{W}' \in \mathbb{R}^{(k+l)\times d}$ can be written as $\mathbf{W}' = \begin{bmatrix} \mathbf{W} \\ \mathbf{0} \end{bmatrix}$ where $\mathbf{0} \in \mathbb{R}^{l\times d}$ is a matrix of zeros. The augmented matrix $\mathbf{W}'$ retains the original k attributes and adds l new attributes initialized to zero. Doing so does not affect the model as the new l dimensions are muted until fine-tuning occurs, enabling many new attributes to be included in virtually any part of the model via EAL as shown in Fig. 4.1.

5. Experiments

In this section, we empirically demonstrate the state-of-the-art performance of `RouteFinder` in several experiments.[1]. We employ as traditional baselines the state-of-the-art HGS-PyVRP (Wouda et al., 2024) and Google OR-Tools (Perron and Furnon, 2023). In terms of neural solvers, we consider MTPOMO (Liu et al., 2024a), based on POMO (Kwon et al., 2020), and MVMoE (Zhou et al., 2024), which additionally includes mixture-of-experts (Fedus et al., 2022) to improve the model performance as baselines for multi-task learned models. We consider two versions of `RouteFinder` (also denoted as `RF` in tables for brevity): one version considering POMO as a backbone (`RF`-POMO) and another one with the MVMoE model with four experts and hierarchical gating (`RF`-MoE-L).

5.1. Main Results

Table G.1 compares `RouteFinder` to the previously discussed baselines. We note that the neural baselines perform well on simpler tasks such as the vanilla CVRP, but overall much worse on more complex VRPs, particularly with time windows (TW) and open (O) problems, hinting at the usefulness of our proposed global attribute embeddings. While the training and testing for these results are performed on the same uniform distribution of 50 and 100 nodes, we include results on large-scale CVRPLIB instances in Table G.3 in the Appendix. Remarkably, even though baselines may outperform our method when testing on similar settings as in training, `RouteFinder` can scale better than the neural baselines in real-world settings. Moreover, training on any possible attribute combination (either with our MBT sampling or without) greatly improves the performance of `RouteFinder` and baselines, even for tasks such as CVRP, both in distribution and real-world benchmarks.

[1]We open-source the code at: https://github.com/ai4co/routefinder

5.2. Ablation Studies

We conduct an ablation study to evaluate the impact of the newly introduced components. To this end, we train `RouteFinder` without 1) the mixed batch training mode (Section 4.2), and 2) the new global attribute embeddings (Section 4.3) with the same training strategy. We tested ablation models on the mixed test instances used in Table G.1. The results are shown in Table G.2 and demonstrate that especially the global feature embeddings and the mixed batch training mode have a significant impact on the performance of `RouteFinder`, improving the gap by 0.23 and 0.21 percentage points, respectively.

5.3. Generalization with EAL

We evaluate `RouteFinder` in few-shot learning settings to unseen attributes, namely the mixed (M) backhauls variants. Unlike classical backhauls, this setting allows to pick up items before delivering, but the model needs to keep track of the current number of picked up items as well as a new *global attribute* to effectively learn to plan. We initialize a new EAL that results in a global embedding $\mathbf{W}'_0$ adding $l = 1$ features, i.e., the mixed backhaul flag. Moreover, we additionally encode the available load accounting for the backhaul demand that has been picked up as a dynamic context during decoding, resulting in another EAL $\mathbf{W}'_c$, also adding one dimension.

We compare our EAL against 1) zero-shot performance of the model, 2) fine-tuning only without adding any adapter layer, 3) adding new layers while keeping the backbone fixed (i.e., the proposed method of Lin et al. (2024)) and 4) training from scratch. We train baselines and EAL with the same setup as the full training, but for only 10 epochs in which 10K instances are sampled each. Fig. G.2 shows that EAL can outperform baselines in few-shot learning. Remarkably, keeping the backbone model fixed but changing projection layers significantly drops the performance in the first few epochs, even against zero-shot settings. Conversely, fine-tuning and EAL can preserve the previous information, thus avoiding performance drops, with EAL converging the fastest.

6. Conclusion

We presented `RouteFinder`, a general framework to develop foundation models for VRPs. We introduced a unified VRP environment that represents any combination of attributes. Moreover, we introduced a novel Mixed Batch Training technique that allows learned VRP solvers to learn effective solution strategies for a wide variety of different VRP variants with better robustness, as well as Global Attribute Embeddings that enable deep layers to effectively represent different variants. Finally, we introduced Effi-

cient Adapter Layers that enable the model to efficiently fine-tune to extended variants with any number of new attributes. We evaluate `RouteFinder` with extensive experiments on 24 VRP variants with promising results in advancing foundation models for VRPs. We refer the reader to Appendix E for limitations and, importantly, future works for `RouteFinder`.

Acknowledgements

We would like to thank people in the AI4CO open research community who have contributed to `RouteFinder`. We also gratefully acknowledge the funding of this project by computing time provided by the Paderborn Center for Parallel Computing (PC2). Furthermore, we thank OMELET for supporting us with additional computing.

Funding

This work was supported by the Institute of Information & communications Technology Planning & Evaluation (IITP) grant funded by the Korean government(MSIT)[2022-0-01032, Development of Collective Collaboration Intelligence Framework for Internet of Autonomous Things].

References

L. Accorsi and D. Vigo. A fast and scalable heuristic for the solution of large-scale capacitated vehicle routing problems. *Transportation Science*, 55(4):832–856, 2021.

L. Accorsi and D. Vigo. Routing one million customers in a handful of minutes. *Computers & Operations Research*, 164:106562, 2024.

A. Bdeir, J. K. Falkner, and L. Schmidt-Thieme. Attention, filling in the gaps for generalization in routing problems. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 505–520. Springer, 2022.

I. Bello, H. Pham, Q. V. Le, M. Norouzi, and S. Bengio. Neural combinatorial optimization with reinforcement learning. *arXiv preprint arXiv:1611.09940*, 2016.

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

F. Berto, C. Hua, J. Park, L. Luttmann, Y. Ma, F. Bu, J. Wang, H. Ye, M. Kim, S. Choi, N. G. Zepeda, A. Hottung, J. Zhou, J. Bi, Y. Hu, F. Liu, H. Kim, J. Son, H. Kim, D. Angioni, W. Kool, Z. Cao, J. Zhang, K. Shin, C. Wu, S. Ahn, G. Song, C. Kwon, L. Xie, and J. Park. RL4CO: an Extensive Reinforcement Learning for Combinatorial Optimization Benchmark. *arXiv preprint arXiv:2306.17100*, 2024. URL https://github.com/ai4co/rl4co.

J. Bi, Y. Ma, J. Wang, Z. Cao, J. Chen, Y. Sun, and Y. M. Chee. Learning generalizable models for vehicle routing problems via knowledge distillation. *Advances in Neural Information Processing Systems*, 2022.

A. Bogyrbayeva, M. Meraliyev, T. Mustakhov, and B. Dauletbayev. Learning to solve vehicle routing problems: A survey. *arXiv preprint arXiv:2205.02453*, 2022.

F. Chalumeau, S. Surana, C. Bonnet, N. Grinsztajn, A. Pretorius, A. Laterre, and T. Barrett. Combinatorial optimization with policy adaptation using latent space search. *Advances in Neural Information Processing Systems*, 36, 2024.

X. Chen, Y. Li, Y. Yang, L. Zhang, S. Li, and G. Pan. Extnco: A fine-grained divide-and-conquer approach for extending nco to solve large-scale traveling salesman problem. *Available at SSRN 4679437*, 2024.

J. Choo, Y.-D. Kwon, J. Kim, J. Jae, A. Hottung, K. Tierney, and Y. Gwon. Simulation-guided beam search for neural combinatorial optimization. *Advances in Neural Information Processing Systems*, 35:8760–8772, 2022.

T. Dao. FlashAttention-2: Faster attention with better parallelism and work partitioning. *arXiv preprint arXiv:2307.08691*, 2023.

T. Dao, D. Fu, S. Ermon, A. Rudra, and C. Ré. Flashattention: Fast and memory-efficient exact attention with io-awareness. *Advances in Neural Information Processing Systems*, 35:16344–16359, 2022.

D. Drakulic, S. Michel, F. Mai, A. Sors, and J.-M. Andreoli. BQ-NCO: Bisimulation quotienting for efficient neural combinatorial optimization. *Advances in Neural Information Processing Systems*, 36, 2024.

W. Fedus, J. Dean, and B. Zoph. A review of sparse expert models in deep learning. *arXiv preprint arXiv:2209.01667*, 2022.

Z.-H. Fu, K.-B. Qiu, and H. Zha. Generalize a small pre-trained model to arbitrarily large tsp instances. In *Proceedings of the AAAI conference on artificial intelligence*, volume 35, pages 7474–7482, 2021.

C. Gao, H. Shang, K. Xue, D. Li, and C. Qian. Towards generalizable neural solvers for vehicle routing problems via ensemble with transferrable local policy. *arXiv preprint arXiv:2308.14104*, 2023.

M. Goetschalckx and C. Jacobs-Blecha. The vehicle routing problem with backhauls. *European Journal of Operational Research*, 1989.

N. Grinsztajn, D. Furelos-Blanco, S. Surana, C. Bonnet, and T. Barrett. Winner takes it all: Training performant rl populations for combinatorial optimization. *Advances in Neural Information Processing Systems*, 36, 2024.

K. Helsgaun. An extension of the lin-kernighan-helsgaun tsp solver for constrained traveling salesman and vehicle routing problems. *Roskilde: Roskilde University*, 12: 966–980, 2017.

A. Hottung and K. Tierney. Neural large neighborhood search for the capacitated vehicle routing problem. *arXiv preprint arXiv:1911.09539*, 2019.

A. Hottung, Y.-D. Kwon, and K. Tierney. Efficient active search for combinatorial optimization problems. *arXiv preprint arXiv:2106.05126*, 2021.

A. Hottung, M. Mahajan, and K. Tierney. PolyNet: Learning diverse solution strategies for neural combinatorial optimization. *arXiv preprint arXiv:2402.14048*, 2024.

Q. Hou, J. Yang, Y. Su, X. Wang, and Y. Deng. Generalize learned heuristics to solve large-scale vehicle routing problems in real-time. In *The Eleventh International Conference on Learning Representations*, 2022.

C. K. Joshi, Q. Cappart, L.-M. Rousseau, and T. Laurent. Learning the travelling salesperson problem requires rethinking generalization. *arXiv preprint arXiv:2006.07054*, 2020.

M. Kim, J. Park, et al. Learning collaborative policies to solve NP-hard routing problems. *Advances in Neural Information Processing Systems*, 34:10418–10430, 2021.

M. Kim, J. Park, and J. Park. Sym-NCO: Leveraging symmetricity for neural combinatorial optimization. *Advances in Neural Information Processing Systems*, 35: 1936–1949, 2022.

M. Kim, S. Choi, J. Son, H. Kim, J. Park, and Y. Bengio. Ant colony sampling with gflownets for combinatorial optimization. *arXiv preprint arXiv:2403.07041*, 2024.

D. Kingma and J. Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations (ICLR)*, San Diego, CA, USA, 2015.

Ç. Koç and G. Laporte. Vehicle routing with backhauls: Review and research perspectives. *Computers & Operations Research*, 2018.

V. Konda and J. Tsitsiklis. Actor-critic algorithms. *Advances in neural information processing systems*, 12, 1999.

W. Kool, H. Van Hoof, and M. Welling. Attention, learn to solve routing problems! *International Conference on Learning Representations*, 2019.

W. Kool, H. van Hoof, J. Gromicho, and M. Welling. Deep policy dynamic programming for vehicle routing problems. In *International conference on integration of constraint programming, artificial intelligence, and operations research*, pages 190–213. Springer, 2022.

Y.-D. Kwon, J. Choo, B. Kim, I. Yoon, Y. Gwon, and S. Min. Pomo: Policy optimization with multiple optima for reinforcement learning. *Advances in Neural Information Processing Systems*, 33:21188–21198, 2020.

Y.-D. Kwon, J. Choo, I. Yoon, M. Park, D. Park, and Y. Gwon. Matrix encoding networks for neural combinatorial optimization. *Advances in Neural Information Processing Systems*, 34:5138–5149, 2021.

F. Li, B. Golden, and E. Wasil. The open vehicle routing problem: Algorithms, large-scale test problems, and computational results. *Computers & Operations Research*, 2007.

S. Li, Z. Yan, and C. Wu. Learning to delegate for large-scale vehicle routing. *Advances in Neural Information Processing Systems*, 34:26198–26211, 2021.

I. Lima, E. Uchoa, D. Oliveira, and E. Queiroga. CVRPLIB: Capacitated vehicle routing problem library. *Date accessed*, 8(02):2022, 2014.

Z. Lin, Y. Wu, B. Zhou, Z. Cao, W. Song, Y. Zhang, and S. Jayavelu. Cross-problem learning for solving vehicle routing problems. *IJCAI*, 2024.

F. Liu, X. Lin, Q. Zhang, X. Tong, and M. Yuan. Multi-task learning for routing problem with cross-problem zero-shot generalization. *KDD*, 2024a.

F. Liu, X. Tong, M. Yuan, X. Lin, F. Luo, Z. Wang, Z. Lu, and Q. Zhang. Evolution of heuristics: Towards efficient automatic algorithm design using large language mode. In *ICML*, 2024b. URL https://arxiv.org/abs/2401.02051.

S. Liu, Y. Zhang, K. Tang, and X. Yao. How good is neural combinatorial optimization? A systematic evaluation on the traveling salesman problem. *IEEE Computational Intelligence Magazine*, 18(3):14–28, 2023.

F. Luo, X. Lin, F. Liu, Q. Zhang, and Z. Wang. Neural combinatorial optimization with heavy decoder: Toward

large scale generalization. *Advances in Neural Information Processing Systems*, 36, 2024a.

F. Luo, X. Lin, Z. Wang, T. Xialiang, M. Yuan, and Q. Zhang. Self-improved learning for scalable neural combinatorial optimization. *arXiv preprint arXiv:2403.19561*, 2024b.

Y. Ma, J. Li, Z. Cao, W. Song, L. Zhang, Z. Chen, and J. Tang. Learning to iteratively solve routing problems with dual-aspect collaborative transformer. *Advances in Neural Information Processing Systems*, 34:11096–11107, 2021.

Y. Ma, J. Li, Z. Cao, W. Song, H. Guo, Y. Gong, and Y. M. Chee. Efficient neural neighborhood search for pickup and delivery problems. *arXiv preprint arXiv:2204.11399*, 2022.

Y. Ma, Z. Cao, and Y. M. Chee. Learning to search feasible and infeasible regions of routing problems with flexible neural k-opt. *Advances in Neural Information Processing Systems*, 36, 2024.

N. Mazyavkina, S. Sviridov, S. Ivanov, and E. Burnaev. Reinforcement learning for combinatorial optimization: A survey. *Computers & Operations Research*, 2021.

M. Nazari, A. Oroojlooy, L. Snyder, and M. Takác. Reinforcement learning for solving the vehicle routing problem. *Advances in neural information processing systems*, 31, 2018.

Y. Peng, B. Choi, and J. Xu. Graph learning for combinatorial optimization: a survey of state-of-the-art. *Data Science and Engineering*, 6(2):119–141, 2021.

L. Perron and F. Didier. CP-SAT. URL https://developers.google.com/optimization/cp/cp_solver/.

L. Perron and V. Furnon. OR-Tools. Google, 2023.

S. Ropke and D. Pisinger. A unified heuristic for a large class of vehicle routing problems with backhauls. *European Journal of Operational Research*, 2006.

F. Ruis, G. Burghouts, and D. Bucur. Independent prototype propagation for zero-shot compositionality. *Advances in Neural Information Processing Systems*, 34: 10641–10653, 2021.

Z. Sun and Y. Yang. Difusco: Graph-based diffusion solvers for combinatorial optimization. *Advances in Neural Information Processing Systems*, 36, 2024.

R. S. Sutton, D. McAllester, S. Singh, and Y. Mansour. Policy gradient methods for reinforcement learning with function approximation. *Advances in neural information processing systems*, 12, 1999.

D. Thyssens, T. Dernedde, J. K. Falkner, and L. Schmidt-Thieme. Routing arena: A benchmark suite for neural routing solvers. *arXiv preprint arXiv:2310.04140*, 2023.

A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

T. Vidal. Hybrid genetic search for the cvrp: Open-source implementation and swap* neighborhood. *Computers & Operations Research*, 140:105643, 2022.

T. Vidal, T. G. Crainic, M. Gendreau, and C. Prins. A unified solution framework for multi-attribute vehicle routing problems. *European Journal of Operational Research*, 2014.

O. Vinyals, M. Fortunato, and N. Jaitly. Pointer networks. *Advances in neural information processing systems*, 28, 2015.

C. Wang and T. Yu. Efficient training of multi-task neural solver with multi-armed bandits. *arXiv preprint arXiv:2305.06361*, 2023.

N. A. Wouda, L. Lan, and W. Kool. PyVRP: A high-performance VRP solver package. *INFORMS Journal on Computing*, 2024.

O. Yadan. Hydra - a framework for elegantly configuring complex applications. Github, 2019. URL https://github.com/facebookresearch/hydra.

H. Ye, J. Wang, Z. Cao, F. Berto, C. Hua, H. Kim, J. Park, and G. Song. Large language models as hyper-heuristics for combinatorial optimization, 2024a.

H. Ye, J. Wang, Z. Cao, H. Liang, and Y. Li. DeepACO: Neural-enhanced ant systems for combinatorial optimization. *Advances in Neural Information Processing Systems*, 36, 2024b.

H. Ye, J. Wang, H. Liang, Z. Cao, Y. Li, and F. Li. Glop: Learning global partition and local construction for solving large-scale routing problems in real-time. *AAAI 2024*, 2024c.

J. Zhou, Y. Wu, W. Song, Z. Cao, and J. Zhang. Towards omni-generalizable neural methods for vehicle routing problems. In *International Conference on Machine Learning*, pages 42769–42789. PMLR, 2023.

J. Zhou, Z. Cao, Y. Wu, W. Song, Y. Ma, J. Zhang, and C. Xu. MVMoE: Multi-task vehicle routing solver with mixture-of-experts. In *International Conference on Machine Learning*, 2024.

RouteFinder
Towards Foundation Models for Vehicle Routing Problems
Appendix

A. Unified VRP Environment

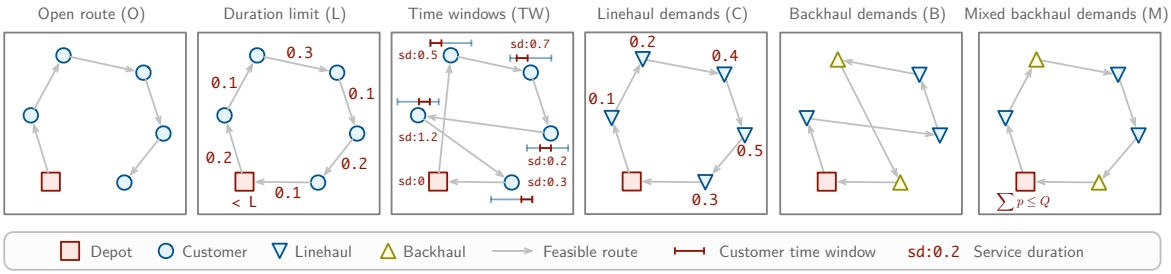

Figure A.1: Different VRP attributes. Open routes (O) and duration limits (L) are *global attributes*, whereas time windows (TW), capacitated vehicles for linehaul demands (C), backhaul demands (B) and mixed (M) backhaul demands are *node attributes*. Attributes may be combined in different ways to define VRP variants.

A.1. Attribute Details

NODE ATTRIBUTES

Demand and Vehicle Capacity (C) $[q \in [0, Q]]$: Every customer $i \in N_c$ has a linehaul demand q_i that needs to be served using vehicles with a homogeneous fixed capacity $Q > 0$. The total customer demand in the vehicle must not exceed its capacity at any point of the route.

Time Windows (TW) $[e, s, l \in [0, T]^3]$: Every customer $i \in N_c$ has a time window $[e_i, l_i]$ during which service must begin. Service takes s_i time. The depot has a time window $[e_0, l_0] = [0, T]$, and a service duration of $s_0 = 0$. Vehicles must reach node i before the end of its time window at l_i, but any early arrivals must wait at the node location until time e_i before service may commence.

Backhauls (B) $[p \in [0, Q]]$: Backhauls generalize demand to also account for return shipments. Customers are either linehaul or backhaul customers. Linehaul customers require delivery of a demand q_i that needs to be transported from the depot to customer i (as in the CVRP), whereas backhaul customers need a pickup of an amount p_i that is transported from the client back to the depot. It is possible for vehicles to serve a combination of linehaul and backhaul customers in a single route, but then any linehaul customers must precede the backhaul customers in the route. An application with returnable bottles is presented in (Ropke and Pisinger, 2006): full bottles need to be delivered from the depot to customers, while empty bottles are returned to the depot via backhaul.

We remark that our definition of backhauls follows the generally accepted definition in the OR community, originally due to (Goetschalckx and Jacobs-Blecha, 1989). This definition differs from the routing problems with backhaul considered in several recent papers in the machine learning (e.g., (Liu et al., 2024a; Zhou et al., 2024)), who define backhaul customers as having a negative demand of the same commodity used for linehaul, and do not consider the precedence constraint that all linehaul must be completed before backhaul may start on the route. The problem setting with a single commodity is not commonly studied in the OR literature since it implies pickups may be used for deliveries at later customers, while the relaxation of the precedence constraint is more properly referred to as a *mixed* backhaul problem (Koç and Laporte, 2018).

GLOBAL ATTRIBUTES

Open Routes (O) $[o \in \{0, 1\}]$: Vehicles are not required to return to the depot after serving all customers. Open routes can be found in applications with third-party drivers, who are often only compensated until they have completed their last delivery (Li et al., 2007).

Duration Limits (L) $[l \in [0, L]]$: Imposes a limit on the total travel duration (or length) of each route, balancing the workload across vehicles. This limit is uniformly applied to all routes.

Mixed (M) Backhauls $[m \in \{0, 1\}]$: Relaxes the strict precedence constraint of linehaul customers before any backhaul customers: with mixed backhauls, linehaul and backhaul customers may be mixed along the route in any configuration. The vehicle's capacity must of course still be respected by the delivery and pickup amounts in the vehicle at any point along the route.

A.2. Data generation process

We consider the six attributes from Section A.1 for instance generation through our environment definition explained in Section 4.1. Leveraging our environment's modular structure, we build the same 16 **VRP variants** as used in MVMoE (Zhou et al., 2024), but by differentiating between *traditional* and *mixed* backhauls, we extend that number to 24, as shown in Table A.1. We now explain the individual steps in the data generation process we use for our modular VRP environment.

Locations We generate $n + 1$ locations randomly with x_i and $y_i \sim U(0, 1), \forall i \in \{0, ..., n\}$, where $[x_0, y_0]$ denotes the depot and $[x_i, y_i], i \in \{1, ..., n\}$, the n customer nodes. Note that this setting can be expanded to consider more realistic distributions as in (Bi et al., 2022; Zhou et al., 2023; Gao et al., 2023), and our implementation is already set up in such a way to allow for different distributions in the future via the `get_sampler` method.

Vehicle capacity (C) The vehicle capacity C is a fixed value applied to all vehicles, and calculated according to:

$$C = \begin{cases} 30 + \left\lfloor \frac{1000}{5} + \frac{n-1000}{33.3} \right\rfloor & \text{if } 1000 < n \\ 30 + \left\lfloor \frac{n}{5} \right\rfloor & \text{if } 20 < n \leq 1000 \\ 30 & \text{otherwise.} \end{cases}$$

which is commonly used in NCO for VRP approaches (Kool et al., 2019; Kwon et al., 2020).

Linehaul and backhaul demands We generate demands according to the following scheme:

1. Generate linehaul demands $q_i \in \{0, ..., Q\}$ for all customers $i \in N_c$. These are needed for both linehaul and backhaul scenarios.
2. Generate backhaul demands $p_i \in \{0, ..., Q\}$ for all customers $i \in N_c$.
3. Generate temporary decision variables $x_i \in \{0, 1\}$ for all customers $i \in N_c$ with probabilities $p(0) = 0.8$ and $p(1) = 0.2$ to determine which customers should be assigned a backhaul demand ($x_i = 1$) and set the backhaul demands to 0 for all remaining customers.

Note that even in a backhaul setting, usually not all customers are backhaul customers, i.e., we need to consider both linehaul and backhaul demands in backhaul problem settings.

Backhaul class (B) For testing the few-shot setting described in Section 5.3, we generate instances with *mixed* backhauls. The instances themselves are actually identical to instances with the *traditional* backhaul and we use a global attribute in the instance to differentiate between them. For this purpose, we allow either setting a fixed value $\in \{1, 2\}$ or sampling from $\{1, 2\}$ for every customer with equal probabilities $p(1) = p(2) = 0.5$, allowing for different backhaul settings within one batch, if needed (see the batching procedure described in Section 4.2). Note that we sample from $\{1, 2\}$ instead of boolean sampling, because we plan to extend the number of backhaul setting in the future.

Open routes (O) For open routes, we generate a boolean vector with all `True` values. During sampling (see Section 4.2) the actual ratio of open route instances is defined, not at the initial instance generation.

Time Windows (TW) We generate the time windows $[e_i, l_i]$ and service times s_i in several steps for all customers $i \in N_c$:

1. Generate service times $s_i \in [0.15, 0.18]$.
2. Generate time window lengths $t_i \in [0.18, 0.2]$.
3. Calculate distances d_{0i} from the depot to the customer.
4. Calculate upper bounds for time window start times $h_i = \frac{t_{max} - s_i - t_i}{d_{0i}} - 1$.
5. Calculate time window start times as $e_i = (1 + (h_i - 1) \cdot u_i) \cdot d_{0i}$ with $u_i \sim U(0, 1)$.
6. Calculate time window end times as $l_i = e_i + t_i$.

Distance limit (L) The distance limit is a fixed value with a default value of 3. We check that $d_{0i} \cdot 2 < L,\ \forall i \in N_c$, where L is the distance limit.

Scaling All demands, both linehauls and backhauls, are scaled to lie in $[0, 1]$ through division by the vehicle capacity.

	Capacity (C)	**Open Route (O)**	**Backhaul (B)**	**Mixed (M)**	**Duration Limit (L)**	**Time Windows (TW)**
CVRP	✓					
OVRP	✓	✓				
VRPB	✓		✓			
VRPL	✓				✓	
VRPTW	✓					✓
OVRPTW	✓	✓				✓
OVRPB	✓	✓	✓			
OVRPL	✓	✓			✓	
VRPBL	✓		✓		✓	
VRPBTW	✓		✓			✓
VRPLTW	✓				✓	✓
OVRPBL	✓	✓	✓		✓	
OVRPBTW	✓	✓	✓			✓
OVRPLTW	✓	✓			✓	✓
VRPBLTW	✓		✓		✓	✓
OVRPBLTW	✓	✓	✓		✓	✓
VRPMB	✓		✓	✓		
OVRPMB	✓	✓	✓	✓		
VRPMBL	✓		✓	✓	✓	
VRPMBTW	✓		✓	✓		✓
OVRPMBL	✓	✓	✓	✓	✓	
OVRPMBTW	✓	✓	✓	✓		✓
VRPMBLTW	✓		✓	✓	✓	✓
OVRPMBLTW	✓	✓	✓	✓	✓	✓

Table A.1: The 24 VRP variants we consider. All variants include the base Capacity (C). The $k = 4$ features O, B, L, and TW can be combined into any subset, including the empty set and itself (i.e., a *power set*) with $2^k = 16$ possible combinations. Finally, we study the additional Mixed (M) global feature that creates new Backhaul (B) variants in generalization studies, adding 8 more variants.

A.3. Determining available actions

To determine available actions for the unified environment formulation, the constraints for the individual problems have to be combined in the action mask. We build a logical test structure, essentially separating the checks in the action mask

according to the individual VRP problem types and then bringing them all together again. The individual **checks** are the following:

a) Can reach in time: depending on the current time and the travel distance to every node not yet visited, can we reach that node before its service time window ends?

b) Can reach depot: if we are handling closed routes, we need to ensure we can reach the depot in time, i.e., the current time plus traveling time to the depot must be smaller than the system end time. For open routes, this will always be set to `True`.

c) Does not exceed distance limit: depending on the current length of the route, if we travel to any available node, will we exceed the total distance limit for the route?

d) Demand constraints for backhaul problems:

- Checks for *all* backhauls problems:
  - Does the linehaul demand exceed vehicle capacity, if we add a node's demand to the current vehicle?
  - Does the backhaul demand exceed vehicle capacity, if we add a node's demand to the current vehicle?
- Checks for *traditional* backhaul settings:
  - Carrying backhaul: if we are already picking up backhaul demands, we cannot service any linehaul demands on this route anymore.
  - If we are not carrying backhaul demands yet, are there any unserved linehaul demands left?
  - If there are no linehaul demands left or we are already carrying backhauls, are there still unserved backhaul demands?
- Checks for *mixed* backhaul settings:
  - Cannot service linehaul demands: depending on the backhaul demands currently loaded in the vehicle, do we have space left for further linehaul demands?

e) Already visited: every customer node needs to be visited exactly one.

We bring together checks a) to e) and introduce an additional check for the depot: if we are currently in the depot and there are still unserved customers, we cannot select the depot as next action. Combining these checks in this way allows us to meticulously check for individual VRP settings while at the same time maintaining the necessary flexibility the unified environment formulation requires.

B. Learning Neural Solvers for VRPs

Solving VRPs using Autoregressive Sequence Generation Autoregressive (AR) methods address CO problems by constructing solutions sequentially. The process begins with encoding the problem instance $\boldsymbol{x}$ (e.g., node and global attributes) using a trainable encoder f_θ which maps $\boldsymbol{x}$ to an embedding $\boldsymbol{h} = f_\theta(\boldsymbol{x})$. The solution $\boldsymbol{a}$ is then decoded based on $\boldsymbol{h}$ through a series of actions, where each action determines the next step in the solution based on the current partial sequence. This is achieved using a decoder g_θ. The encoding and decoding process can be formalized as follows:

$$a_t \sim g_\theta(a_t | a_{t-1}, ..., a_0, \boldsymbol{h}),$$

$$\pi_\theta(\boldsymbol{a}|\boldsymbol{x}) \triangleq \prod_{t=1}^{T-1} g_\theta(a_t | a_{t-1}, ..., a_0, \boldsymbol{h}),$$

where $\boldsymbol{a} = (a_1, ..., a_T)$ represents a feasible solution to the CO problem, T denotes the steps in solution construction, and π_θ is the stochastic solver mapping problem instance $\boldsymbol{x}$ to a solution $\boldsymbol{a}$.

Training VRP Solvers via Reinforcement Learning The solver p_θ can be trained using either supervised learning (SL) or reinforcement learning (RL). This paper focuses on RL due to its ability to train solvers independent of optimal solutions. Under the RL framework, the training objective for neural combinatorial optimization solvers is defined as:

$$\theta^* = \underset{\theta}{\text{argmax}} \left[\mathbb{E}_{\boldsymbol{x} \sim P(\boldsymbol{x})} \left[\mathbb{E}_{a \sim \pi_\theta(\boldsymbol{a}|\boldsymbol{x})} [R(\boldsymbol{a}, \boldsymbol{x})] \right] \right],$$

where $P(\boldsymbol{x})$ is the distribution of problem instances, and $R(\boldsymbol{a}, \boldsymbol{x})$ represents the reward (i.e., the negative cost), associated with the solution $\boldsymbol{a}$ for the given $\boldsymbol{x}$. The above training problem can be tackled using various RL techniques, including value-based, policy gradient (PG) (Sutton et al., 1999), and actor-critic (AC) methods (Konda and Tsitsiklis, 1999). The REINFORCE algorithm is typically employed to train the policy network π_θ, optimizing the policy through estimated gradients of the expected reward as follows:

$$\nabla_\theta \mathcal{L}_a(\theta|\boldsymbol{x}) = \mathbb{E}_{\pi_\theta(\boldsymbol{a}|\boldsymbol{x})}\left[(R(\boldsymbol{a}, \boldsymbol{x}) - b(\boldsymbol{x}))\nabla_\theta \log \pi_\theta(\boldsymbol{a}|\boldsymbol{x})\right],$$

where $b(\cdot)$ is a baseline function used to stabilize training and reduce gradient variance.

C. `RouteFinder` Model

`RouteFinder` follows the encoder-decoder architecture from the Attention Model (Kool et al., 2019), a transformer-like architecture based on the attention mechanism (Vaswani et al., 2017).

C.1. Multi-Head Attention

At the core of `RouteFinder` lies the Multi-Head Attention (MHA) mechanism, proposed by Vaswani et al. (2017). MHA concurrently attends to information from various representation subspaces, facilitating the capture of diverse relationships between input elements. Notably, MHA is capable of handling a variable number of elements.

The MHA operation starts by linearly projecting the input sequences of queries Q, keys K, and values V to H distinct subspaces using learned projection matrices W_i^Q, W_i^K, and W_i^V, respectively, where H denotes the number of attention heads: $Q_i = QW_i^Q$, $K_i = KW_i^K$, $V_i = VW_i^V$ for $i = 1, \ldots, H$. Subsequently, the attention weights for each head are computed by performing a scaled dot product between the projected queries and keys, followed by a softmax operation:

$$A_i = \text{Softmax}\left(\frac{Q_i K_i^T}{\sqrt{d_k}} + M\right) \tag{1}$$

where d_k represents the dimension of the keys, acting as a scaling factor to prevent the dot products from growing too large, $\text{Softmax}(x_i) = \frac{\exp(x_i)}{\sum_{j=1}^N \exp(x_j)}$ and M is an optional attention mask that can be used to prevent attending to certain positions (e.g., infeasible actions), which can be done by setting elements to $-\infty$. The output of each attention head is then calculated as a weighted sum of the projected values, using the attention weights: $Z_i = A_i V_i$.

Lastly, the outputs from all attention heads are concatenated and linearly projected using a learned matrix W^O to yield the final output of the MHA operation:

$$\text{MHA}(Q, K, V) = \text{Concat}(Z_1, \ldots, Z_H)W^O \tag{2}$$

While the MHA grows quadratically, i.e., with sequence length (i.e., number of nodes) N, it grows as $O(N^2)$, several efficient implementations have been proposed over the years, and we use FlashAttention (Dao et al., 2022; Dao, 2023) to speed up the model.

C.2. Encoder

The Encoder transforms an input instance $\boldsymbol{x}$ into a hidden embedding $\boldsymbol{h}$. The Encoder architecture consists of the following main components: 1) Global Embedding 2) Node Embedding, and 3) a series of Encoder Layers [2]. We consider a VRP instance of N location as having $N + 1$ nodes, where n_0 is the depot, and $n_1, \ldots, n_N$ are N customers.

Global Embedding The global embedding f captures problem-level attributes and is projected onto the depot node. These attributes include Open Routes $o \in \{0, 1\}$, Duration Limits $l \in [0, L]$, (with EAL only), Mixed Backhauls flag $m \in \{0, 1\}$, l_0 as the late Time Window for the problem, and the location of the depot node $[x_0, y_0] \in \mathbb{R}^2$. In `RouteFinder`, the global embedding f is a linear projection layer $\mathbf{W}_g \in \mathbb{R}^{k \times d}$ where $k = 5$ features and $d = 128$ is the hidden dimension. The initial projected global hidden embedding can be written as $\boldsymbol{h}_g^{(0)} = \mathbf{W}_g[o, l, m, l_0, x_0, y_0]^\top$.

Node Embedding The node embeddings, on the other hand, capture customer-specific attributes and are projected onto the remaining N nodes. These attributes include for nodes $i \in 1, \ldots, N$: Demand and Vehicle Capacity $q_i \in [0, Q]$], Time Windows parameters $e_i, s_i, l_i \in [0, T]^3$] where e and l are early and late time windows and s is the service time, and

[2] Note that the following description might slightly differ from the code implementation due to minor differences - for instance, we erroneously did not implement l_0 for the depot in the main experiments, where l_0 however was kept constant.

$p_i \in [0, Q]$ are Backhaul demands, and finally locations $[x_i, y_i] \in \mathbb{R}^2$. In `RouteFinder` this a linear projection layer $\mathbf{W}_n \in \mathbb{R}^{k \times d}$ where $k = 7$ features and $d = 128$ is the hidden dimension. The initial projected node hidden embedding can be written for each node n_i as $\boldsymbol{h}_{n_i}^{(0)} = \mathbf{W}_n[q_i, e_i, s_i, l_i, p_i, x_i, y_i]^\top$.

Raw Features to Hidden States The projected global embedding and node embeddings are concatenated to obtain the initial hidden representation $\boldsymbol{h}^{(0)} \in \mathbb{R}^{N \times d}$, where N is the total number of nodes (depot + customers) and d is the hidden dimension:

$$\boldsymbol{h}^{(0)} = \text{Concat}(\boldsymbol{h}_g^{(0)}, \boldsymbol{h}_{n_1}^{(0)}, \ldots, \boldsymbol{h}_{n_N}^{(0)}) \tag{3}$$

The initial hidden representation $\boldsymbol{h}^{(0)}$ is then passed through a series of Encoder Layers to refine and enrich the representation. Each Encoder Layer consists of a Multi-Head Attention (MHA) layer and a Multi-Layer Perceptron (MLP) layer, as described in Eq. (5) and Eq. (6), respectively.

The Encoder can be represented as:

$$\boldsymbol{h} = \text{EncoderBlocks}(\boldsymbol{h}^{(0)}) \tag{4}$$

Each EncoderBlock consists of two sub-layers: a Multi-Head Attention (MHA) layer and a Multi-Layer Perceptron (MLP) layer. The MHA layer allows the model to capture dependencies between different positions in the input sequence, while the MLP layer applies non-linear transformations to the features at each position. The input to each EncoderBlock is first passed through the MHA layer, which computes the self-attention using the input as queries, keys, and values:

$$\hat{\boldsymbol{h}} = \text{Norm}\left(\boldsymbol{h}^{(\ell-1)} + \text{MHA}(\boldsymbol{h}^{(\ell-1)}, \boldsymbol{h}^{(\ell-1)}, \boldsymbol{h}^{(\ell-1)})\right) \tag{5}$$

where $\boldsymbol{h}^{(\ell-1)}$ represents the input to the ℓ-th EncoderBlock, and Norm denotes a normalization operation, in `RouteFinder` we employ Instance Normalization (IN). The output of the MHA layer, $\hat{\boldsymbol{h}}$, is then passed through the MLP layer, which applies a series of linear transformations with non-linear activations:

$$\boldsymbol{h}^{(\ell)} = \text{Norm}\left(\hat{\boldsymbol{h}} + \text{MLP}(\hat{\boldsymbol{h}})\right) \tag{6}$$

The pointwise MLP layer consists of two linear layers with a non-linear activation function as ReLU, between them. In `RouteFinder` we adopt 6 encoder blocks.

C.3. Decoder

The Decoder autoregressively constructs the solution based on the Encoder output $\boldsymbol{h}$ and the state at the current step t, s_t.

Context Embedding The context embedding is used to modify the query embedding of the problem node of the current partial solution. It consists of a linear layer that projects the concatenated current node embedding and state embedding to the embedding space. The state embedding is computed by projecting the following: the current node embedding $_t$ and a set of dynamic features from state s_t, i.e. the available load c_t, current time t_t and current distance traveled d_t.

Attention and Pointer Mechanism The query q_t is then obtained by projecting the concatenated current node embedding and state embedding using a linear layer:

$$q_t = \mathbf{W}_c\text{Concat}([\boldsymbol{h}_t; [c_t, t_t, d_t]])^\top \tag{7}$$

where $\mathbf{W}_c \in \mathbb{R}^{d \times (d+k)}$ is the linear projection matrix, $d = 128$ is the hidden dimension, and $k = 3$ is the number of state features (available load, current time, and remaining distance). Note that with EAL, we additionally add another feature so that $k = 4$, namely the difference between the vehicle capacity and the *used backhaul capacity*. This is necessary because if we pick up items, the deliverable quantity must exceed the remaining capacity after pick up. The query q_t is then passed into a masked MHA layer and final single-head attention to obtain logits $\boldsymbol{z}$:

$$h_t^c = \text{MHA}(q_t, K_t^g, V_t^g, M_t), \tag{8}$$

$$\boldsymbol{z} = \frac{V_t^p h_t^c}{\sqrt{d_k}} \tag{9}$$

where M_t is the set of feasible actions (i.e., the `action_mask`), and projections $K_t^g, V_t^g, V_t^p = W_k^g \boldsymbol{h}, W_v^g \boldsymbol{h}, W_v^p \boldsymbol{h}$ are precomputed once as cache. We note that Eq. (9) is usually referred to as the pointer mechanism (Vinyals et al., 2015).

Logits processing Finally, logits $\boldsymbol{z}$ are transformed into a probability distribution:

$$p = \text{Softmax}\left(C \cdot \tanh(\boldsymbol{z})\right) \tag{10}$$

where logits for infeasible actions can be masked, and C is the *tanh clipping* that serves in improving the exploration, which we set to 10 according to Bello et al. (2016).

Action selection During training, we use the POMO `multistart` sampling which forces the first action to start from all nodes to enhance diversity. During testing, we also employ `multistart` but with greedy selection (i.e., selecting the maximum probability). Prior to the selection, a dihedral augmentation is also performed prior to encoding instance $\boldsymbol{x}$ in the encoder, which enables exploring $8\times$ as many solutions with 4 rotations $\times$ 2 flips. We note that additional augmentations and techniques can be performed during inference, which can further boost evaluation performance (Kim et al., 2022; Ma et al., 2022; Choo et al., 2022; Luo et al., 2024a), which we do not use for fairness of comparison but could greatly boost `RouteFinder` performance.

D. Extended Related Works

Neural combinatorial optimization for VRPs NCO has emerged as a pivotal solution approach for VRPs and other CO problems, leveraging advancements in machine learning and neural network architectures (Bengio et al., 2021; Peng et al., 2021; Mazyavkina et al., 2021; Bogyrbayeva et al., 2022). The seminal work of Vinyals et al. (2015) using pointer networks paved the way to apply these techniques to CO problems, where they now routinely find near-optimal solutions for VRPs through further developments by Bello et al. (2016) and Nazari et al. (2018). Subsequent innovations, including the transformer-based encoder with self-attention of Kool et al. (2019), POMO (Kwon et al., 2020) and Sym-NCO (Kim et al., 2022), have significantly enhanced solution generation and improvement strategies for VRPs. These advancements have been complemented by novel training algorithms, including learning with (partial) problem re-encoding at each step (Bdeir et al., 2022; Drakulic et al., 2024; Luo et al., 2024a;b) and population-based approaches (Grinsztajn et al., 2024; Hottung et al., 2024; Chalumeau et al., 2024).

Despite this progress, challenges remain in the form of requiring manual tuning for inductive bias, the need for problem-specific models, and lack of generalization, which impact deployment and generalizability (Liu et al., 2023; Thyssens et al., 2023). The field has also explored non-autoregressive solution construction methods that allow for better generalization, such as predicting promising edges (Joshi et al., 2020; Fu et al., 2021; Kool et al., 2022; Sun and Yang, 2024), improvement methods iteratively refining solutions through local adjustments or sequential rewriting (Hottung and Tierney, 2019; Ma et al., 2021; 2022; 2024), and test-time adaptation methods (Hottung et al., 2021; Choo et al., 2022) which allow for solution improvement given larger time budgets. Recent works additionally explore alternative ways of solving VRPs, such as learning heuristics for Ant Colony Optimization (Ye et al., 2024b; Kim et al., 2024) and divide-and-conquer methods (Kim et al., 2021; Li et al., 2021; Hou et al., 2022; Ye et al., 2024c; Chen et al., 2024).

E. Limitations & Future Works

Our approach represents an early attempt to learn across problem variants, but it does so at the expense of solution quality compared to techniques trained on specific problem variants. Furthermore, autoregressive models have known scaling issues, and even though they are able to solve real-world-sized problems effectively, the question remains how to best integrate them into decompositions (Ye et al., 2024c) for solving larger problems. Finally, our foundation model ignores asymmetric problem settings as in Kwon et al. (2021), which are highly relevant in real-world VRPs. For future work, we intend to extend `RouteFinder` to support further variants of the vast VRP literature. We also intend to improve the performance of the model with the goal of achieving parity with state-of-the-art, traditional OR solvers.

F. Licenses for used assets

Table F.1 lists the used assets and their licenses. Our code is licensed under the MIT License.

Table F.1: Used assets and their licenses.

Type	Asset	License	Usage
Code	POMO (Kwon et al., 2020)	MIT License	Evaluation
	MTPOMO (Liu et al., 2024a)	MIT License	Evaluation
	MVMoE (Zhou et al., 2024)	MIT License	Evaluation
	RL4CO (Berto et al., 2024)	MIT License	Evaluation
	ORTools (Perron and Didier)	Apache-2.0	Evaluation
	PyVRP (Wouda et al., 2024)	MIT License	Evaluation
Dataset	CVRPLib (Lima et al., 2014)	Available for any non-commercial use	Testing

G. Additional Experiments

G.1. Experimental Setup Details

Hardware All training runs are conducted on NVIDIA A100 GPUs and take between 9 to 48 hours per model. Evaluation runs are conducted on an AMD Ryzen Threadripper 3960X 24-core CPU with a single RTX 3090 GPU.

Classical Baselines Setup We use PyVRP (Wouda et al., 2024), an open-source, state-of-the-art heuristic VRP solver built on top of HGS-CVRP (Vidal, 2022). PyVRP can solve all VRP variants considered in this study. We also use Google's OR-Tools (Perron and Furnon, 2023), an open-source exact and heuristic solver that relies on constraint programming and commonly used in the ML community for its versatility to solve a large number of VRP variants. We use OR-Tools' guided local search procedure in this work. Both baseline methods solve each instance on a single CPU core with a time limit of 10 and 20 seconds for instances with 50 and 100 nodes, respectively. We parallelize traditional solvers across 16 CPU cores as in Kool et al. (2019).

Training We follow the setup in Kwon et al. (2020) and the recent works on MTPOMO (Liu et al., 2024a) and MVMoE (Zhou et al., 2024). Each model is trained over 300 epochs, with each epoch containing 100,000 instances generated on the fly. We use the Adam optimizer (Kingma and Ba, 2015) with a learning rate of 3×10^{-4} and a batch size of 256. At epochs 270 and 295, the learning rate is reduced by a factor of 0.1. We observe that these settings enhance convergence compared to using a learning rate of 1×10^{-4}. Our setup, however, is different from the one in Liu et al. (2024a) and Zhou et al. (2024) in that we do not artificially restrict the variants with single attributes (such as only (B) or (TW)), but train on *all* available data to enhance robustness. This is similar to how LLMs are trained on all available data, which is easily available through our VRP environment.

G.2. Experiment Configuration

We show a non-exhaustive Hydra (Yadan, 2019) configuration template in Listing 1 for `RouteFinder` based on POMO.

Listing 1 Template `experiment.yaml` configuration with the most notable hyperparameters for POMO-trained `RouteFinder`.

Hydra Configuration

```yaml
# Environment
env:
  _target_: routefinder.envs.mtvrp.MTVRPEnv
  generator_params:
    num_loc: 100
    variant_preset: "all"
    use_combinations: True

# RL Algorithm and policy (env passed automatically)
model:
  _target_: rl4co.models.model.RouteFinderBase
  policy:
    _target_: routefinder.models.policy.RouteFinderPolicy
    embed_dim: 128
    feedforward_hidden: 512

    num_heads: 8
    num_encoder_layers: 6
    normalization: "instance"
    tanh_clipping: 10.0
    mask_logits: True
  train_decode_type: "multistart_sampling"
  test_decode_type: "multistart_greedy"
  augment_fn: "dihedral"
  batch_size: 256
  train_data_size: 100_000
  optimizer_kwargs:
    lr: 3e-4
    weight_decay: 1e-6
  lr_scheduler:
    "MultiStepLR"
  lr_scheduler_kwargs:

    milestones: [270, 295]
    gamma: 0.1

seed: 69420

# Trainer
trainer:
  gradient_clip_val: 1.0
  max_epochs: 300

```

Inference For all ML approaches we roll out solutions greedily using multi-starts and $8\times$ symmetric dihedral augmentations of (Kwon et al., 2020), resulting in $n \times 8$ solutions per instance.

G.3. Experimental Results

We present the main empirical results in full in Table G.1, deferred to the Appendix for space.

We report in Fig. G.2 the experiments on few-shot learning (finetuning) of `RouteFinder`. We compare our EAL against 1) zero-shot performance of the model, 2) fine-tuning only without adding any adapter layer, 3) adding new layers while keeping the backbone fixed (i.e., the proposed method of Lin et al. (2024)) and 4) training from scratch. We train baselines and EAL with the same setup as the full training, but for only 10 epochs in which 10K instances are sampled each. Fig. G.2 shows that EAL can outperform baselines in few-shot learning. Remarkably, keeping the backbone model fixed but changing projection layers significantly drops the performance in the first few epochs, even against zero-shot settings.

Table G.1: Performance on 1000 test instances of trained VRPs. * represents the best solutions against which the other gaps are computed. Best neural approach in **bold**; second underlined.

	Solver	$n = 50$			$n = 100$				Solver	$n = 50$			$n = 100$		
		Obj.	Gap	Time	Obj.	Gap	Time			Obj.	Gap	Time	Obj.	Gap	Time
CVRP	HGS-PyVRP	10.287	*	4.6m	15.543	*	9.2m	VRPTW	HGS-PyVRP	16.032	*	4.6m	25.433	*	9.2m
	OR-Tools	10.523	2.294 %	4.6m	16.361	5.263 %	9.2m		OR-Tools	16.124	0.574 %	4.6m	25.923	1.927 %	9.2m
	MTPOMO	10.458	1.662 %	2s	15.796	1.628 %	10s		MTPOMO	16.570	3.356 %	2s	26.403	3.814 %	11s
	MVMoE	**10.414**	**1.235 %**	3s	**15.759**	**1.390 %**	13s		MVMoE	16.455	2.638 %	3s	26.374	3.700 %	14s
	MVMoE-L	10.448	1.565 %	3s	15.777	1.506 %	12s		MVMoE-L	16.521	3.050 %	3s	26.392	3.771 %	13s
	RF-POMO	10.438	1.468 %	2s	15.836	1.885 %	10s		RF-POMO	**16.380**	**2.171 %**	2s	26.294	3.385 %	11s
	RF-MoE-L	10.424	1.332 %	2s	15.818	1.769 %	12s		RF-MoE-L	16.381	2.177 %	3s	**26.256**	**3.236 %**	13s
OVRP	HGS-PyVRP	6.494	*	4.6m	9.730	*	9.2m	VRPL	HGS-PyVRP	10.328	*	4.6m	15.637	*	9.2m
	OR-Tools	6.555	0.939 %	4.6m	10.081	3.607 %	9.2m		OR-Tools	10.570	2.343 %	4.6m	16.466	5.302 %	9.2m
	MTPOMO	6.818	4.989 %	2s	10.239	5.231 %	10s		MTPOMO	10.502	1.685 %	2s	15.905	1.714 %	12s
	MVMoE	6.760	4.096 %	3s	10.195	4.779 %	13s		MVMoE	**10.457**	**1.249 %**	3s	**15.865**	**1.458 %**	13s
	MVMoE-L	6.796	4.650 %	2s	10.224	5.077 %	12s		MVMoE-L	10.488	1.549 %	2s	15.885	1.586 %	10s
	RF-POMO	6.706	3.265 %	2s	10.204	4.872 %	10s		RF-POMO	10.482	1.491 %	2s	15.955	2.034 %	10s
	RF-MoE-L	**6.682**	**2.895 %**	2s	**10.165**	**4.471 %**	12s		RF-MoE-L	10.464	1.317 %	2s	15.924	1.835 %	12s
VRPB	HGS-PyVRP	9.688	*	4.6m	14.386	*	9.2m	OVRPTW	HGS-PyVRP	10.485	*	4.6m	16.900	*	9.2m
	OR-Tools	9.829	1.455 %	4.6m	15.010	4.338 %	9.2m		OR-Tools	10.497	0.114 %	4.6m	17.023	0.728 %	9.2m
	MTPOMO	10.105	4.304 %	2s	15.012	4.351 %	10s		MTPOMO	10.851	3.491 %	2s	17.525	3.698 %	11s
	MVMoE	10.009	3.313 %	3s	**14.948**	**3.907 %**	13s		MVMoE	10.760	2.623 %	3s	17.496	3.527 %	15s
	MVMoE-L	10.069	3.933 %	2s	14.983	4.150 %	11s		MVMoE-L	10.797	2.976 %	2s	17.516	3.645 %	14s
	RF-POMO	10.012	3.344 %	2s	15.020	4.407 %	10s		RF-POMO	**10.651**	**1.583 %**	2s	17.355	2.692 %	11s
	RF-MoE-L	**9.977**	**2.983 %**	2s	14.980	4.129 %	11s		RF-MoE-L	10.656	1.631 %	3s	**17.330**	**2.544 %**	14s
OVRPB	HGS-PyVRP	6.897	*	4.6m	10.304	*	9.2m	OVRPBL	HGS-PyVRP	6.904	*	4.6m	10.310	*	9.2m
	OR-Tools	6.940	0.623 %	4.6m	10.611	2.979 %	9.2m		OR-Tools	6.949	0.652 %	4.6m	10.613	2.939 %	9.2m
	MTPOMO	7.269	5.394 %	2s	10.901	5.794 %	10s		MTPOMO	7.274	5.359 %	2s	10.903	5.752 %	10s
	MVMoE	7.179	4.089 %	3s	10.846	5.260 %	13s		MVMoE	7.191	4.157 %	3s	10.858	5.315 %	13s
	MVMoE-L	7.239	4.959 %	2s	10.874	5.532 %	12s		MVMoE-L	7.239	4.852 %	2s	10.886	5.587 %	12s
	RF-POMO	7.108	3.059 %	2s	10.816	4.969 %	10s		RF-POMO	7.117	3.085 %	2s	10.825	4.995 %	10s
	RF-MoE-L	**7.091**	**2.813 %**	2s	**10.773**	**4.552 %**	12s		RF-MoE-L	**7.099**	**2.824 %**	2s	**10.781**	**4.568 %**	12s
OVRPBLTW	HGS-PyVRP	11.597	*	4.6m	19.005	*	9.2m	OVRPBTW	HGS-PyVRP	11.590	*	4.6m	19.167	*	9.2m
	OR-Tools	11.612	0.129 %	4.6m	19.198	1.016 %	9.2m		OR-Tools	11.610	0.173 %	4.6m	19.314	0.767 %	9.2m
	MTPOMO	11.963	3.156 %	2s	19.626	3.268 %	11s		MTPOMO	11.957	3.167 %	2s	19.780	3.198 %	11s
	MVMoE	11.847	2.156 %	3s	19.588	3.068 %	15s		MVMoE	11.849	2.235 %	3s	19.752	3.052 %	15s
	MVMoE-L	11.883	2.466 %	3s	19.605	3.157 %	14s		MVMoE-L	11.880	2.502 %	2s	19.770	3.146 %	14s
	RF-POMO	**11.735**	**1.190 %**	2s	19.429	2.231 %	11s		RF-POMO	**11.733**	**1.234 %**	2s	19.579	2.150 %	11s
	RF-MoE-L	11.743	1.259 %	3s	**19.402**	**2.089 %**	14s		RF-MoE-L	11.737	1.268 %	3s	**19.558**	**2.040 %**	14s
OVRPL	HGS-PyVRP	6.510	*	4.6m	9.709	*	9.2m	OVRPLTW	HGS-PyVRP	10.455	*	4.6m	16.962	*	9.2m
	OR-Tools	6.571	0.937 %	4.6m	10.047	3.481 %	9.2m		OR-Tools	10.465	0.096 %	4.6m	17.100	0.814 %	9.2m
	MTPOMO	6.839	5.054 %	2s	10.210	5.160 %	10s		MTPOMO	10.803	3.329 %	2s	17.589	3.696 %	11s
	MVMoE	6.777	4.101 %	3s	10.169	4.738 %	13s		MVMoE	10.718	2.516 %	3s	17.553	3.484 %	15s
	MVMoE-L	6.803	4.501 %	2s	10.199	5.047 %	12s		MVMoE-L	10.750	2.822 %	3s	17.577	3.626 %	14s
	RF-POMO	6.727	3.333 %	2s	10.176	4.810 %	10s		RF-POMO	**10.611**	**1.492 %**	2s	17.429	2.753 %	11s
	RF-MoE-L	**6.700**	**2.919 %**	2s	**10.135**	**4.388 %**	12s		RF-MoE-L	10.617	1.549 %	3s	**17.401**	**2.588 %**	14s
VRPBL	HGS-PyVRP	9.688	*	4.6m	14.373	*	9.2m	VRPBLTW	HGS-PyVRP	18.361	*	4.6m	29.026	*	9.2m
	OR-Tools	9.820	1.363 %	4.6m	15.084	4.947 %	9.2m		OR-Tools	18.422	0.332 %	4.6m	29.830	2.770 %	9.2m
	MTPOMO	10.112	4.377 %	2s	15.023	4.522 %	10s		MTPOMO	18.841	2.614 %	2s	30.232	4.155 %	11s
	MVMoE	10.018	3.406 %	3s	**14.951**	**4.021 %**	13s		MVMoE	18.715	1.928 %	3s	30.216	4.100 %	15s
	MVMoE-L	10.080	4.046 %	2s	14.993	4.314 %	11s		MVMoE-L	18.773	2.244 %	3s	30.223	4.124 %	13s
	RF-POMO	10.026	3.489 %	2s	15.030	4.571 %	10s		RF-POMO	18.628	1.454 %	2s	30.094	3.679 %	11s
	RF-MoE-L	**9.992**	**3.138 %**	2s	14.982	4.237 %	12s		RF-MoE-L	**18.622**	**1.421 %**	3s	**30.060**	**3.562 %**	13s
VRPBTW	HGS-PyVRP	18.167	*	4.6m	29.000	*	9.2m	VRPLTW	HGS-PyVRP	15.951	*	4.6m	25.678	*	9.2m
	OR-Tools	18.374	1.139 %	4.6m	29.964	3.324 %	9.2m		OR-Tools	16.036	0.533 %	4.6m	26.156	1.862 %	9.2m
	MTPOMO	18.797	3.468 %	2s	30.325	4.569 %	11s		MTPOMO	16.480	3.316 %	2s	26.684	3.918 %	11s
	MVMoE	18.684	2.846 %	3s	30.319	4.548 %	15s		MVMoE	16.368	2.614 %	3s	26.655	3.805 %	14s
	MVMoE-L	18.760	3.264 %	2s	30.323	4.562 %	14s		MVMoE-L	16.437	3.047 %	3s	26.674	3.879 %	13s
	RF-POMO	18.589	2.323 %	2s	30.181	4.072 %	11s		RF-POMO	16.303	2.207 %	2s	26.572	3.482 %	11s
	RF-MoE-L	**18.585**	**2.301 %**	3s	**30.157**	**3.990 %**	13s		RF-MoE-L	**16.298**	**2.175 %**	3s	**26.533**	**3.330 %**	13s

Conversely, fine-tuning and EAL can preserve the previous information, thus avoiding performance drops, with EAL converging the fastest.

Table G.2 reports an ablation study across tasks of proposed RouteFinder components, where each effectively improves performances.

Table G.3: Results on large-scale CVRPLIB instances from the X set. All models are only trained on the uniformly distributed data with the size $n = 100$ and evaluated via greedy rollouts. Results for methods with † are drawn from Zhou et al. (2024), models trained with single features excluding feature compositions (except for OVRPTW). Training on multiple variants enhances generalization across models.

Set-X		POMO†		MTPOMO †		MVMoE†		MVMoE-L†		MTPOMO		MVMoE-L		MVMoE		RF-POMO	
Instance	Opt.	Obj.	Gap	Obj.	Gap	Obj.	Gap	Obj.	Gap	Obj.	Gap	Obj.	Gap	Obj.	Gap	Obj.	Gap
X-n502-k39	69226	75617	9.232%	77284	11.640%	73533	6.222%	74429	7.516%	77522	11.984%	76687	10.778%	74318	7.356%	76004	9.791%
X-n513-k21	24201	30518	26.102%	28510	17.805%	32102	32.647%	31231	29.048%	28395	17.330%	28255	16.751%	29419	21.561%	28527	17.875%
X-n524-k153	154593	201877	30.586%	192249	24.358%	186540	20.665%	182392	17.982%	170240	10.121%	172270	11.435%	173869	12.469%	174508	12.882%
X-n536-k96	94846	106073	11.837%	106514	12.302%	109581	15.536%	108543	14.441%	104049	9.703%	104205	9.868%	105737	11.483%	104871	10.570%
X-n548-k50	86700	103093	18.908%	94562	9.068%	95894	10.604%	95917	10.631%	105699	21.913%	102221	17.902%	103894	19.832%	101196	16.720%
X-n561-k42	42717	49370	15.575%	47846	12.007%	56008	31.114%	51810	21.287%	48994	14.694%	47908	12.152%	50151	17.403%	49539	15.970%
X-n573-k30	50673	83545	64.871%	60913	20.208%	59473	17.366%	57042	12.569%	58647	15.736%	63498	25.309%	59644	17.704%	56851	12.192%
X-n586-k159	190316	229887	20.792%	208893	9.761%	215668	13.321%	214577	12.748%	213955	12.421%	212127	11.460%	207459	9.008%	210637	10.678%
X-n599-k92	108451	150572	38.839%	120333	10.956%	128949	18.901%	125279	15.517%	121936	12.434%	119793	10.458%	121214	11.768%	121424	11.962%
X-n613-k62	59535	68451	14.976%	67984	14.192%	82586	38.718%	74945	25.884%	69175	16.192%	69281	16.370%	72504	21.784%	69637	16.968%
X-n627-k43	62164	84434	35.825%	73060	17.528%	70987	14.193%	70905	14.061%	71289	14.679%	69520	11.833%	69909	12.459%	76403	22.906%
X-n641-k35	63682	75573	18.672%	72643	14.071%	75329	18.289%	72655	14.090%	74013	16.223%	70250	10.314%	71854	12.833%	70401	10.551%
X-n655-k131	106780	127211	19.134%	116988	9.560%	117678	10.206%	118475	10.952%	117581	10.115%	130266	21.995%	121573	13.854%	125846	17.855%
X-n670-k130	146332	208079	42.197%	190118	29.922%	197695	35.100%	183447	25.364%	169786	16.028%	169561	15.874%	169681	15.956%	169553	15.869%
X-n685-k75	68205	79482	16.534%	80892	18.601%	97388	42.787%	89441	31.136%	78240	14.713%	79620	16.736%	83123	21.872%	79617	16.732%
X-n701-k44	81923	97843	19.433%	92075	12.392%	98469	20.197%	94924	15.870%	93681	14.353%	90621	10.617%	91378	11.541%	89327	9.038%
X-n716-k35	43373	51381	18.463%	52709	21.525%	56773	30.895%	52305	20.593%	49944	15.150%	50120	15.556%	52253	20.474%	48106	10.912%
X-n733-k159	136187	159098	16.823%	161961	18.925%	178322	30.939%	167477	22.976%	155223	13.978%	153501	12.713%	158140	16.120%	158791	16.598%
X-n749-k98	77269	87786	13.611%	90582	17.229%	100438	29.985%	94497	22.296%	87992	13.877%	87148	12.785%	90387	16.977%	86630	12.115%
X-n766-k71	114417	135464	18.395%	144041	25.891%	152352	33.155%	136255	19.086%	131245	14.708%	130967	14.465%	131081	14.564%	127796	11.693%
X-n783-k48	72386	90289	24.733%	83169	14.897%	100383	38.677%	92960	28.423%	84553	16.808%	84058	16.125%	89170	23.187%	81011	11.915%
X-n801-k40	73305	124278	69.536%	85077	16.059%	91560	24.903%	87662	19.585%	93004	26.873%	85231	16.269%	88535	20.776%	85453	16.572%
X-n819-k171	158121	193451	22.344%	177157	12.039%	183599	16.113%	185832	17.525%	180986	14.460%	180550	14.185%	177188	12.058%	181984	15.092%
X-n837-k142	193737	237884	22.787%	214207	10.566%	229526	18.473%	221286	14.220%	217954	12.500%	216751	11.879%	217447	12.238%	215919	11.450%
X-n856-k95	88965	152528	71.447%	101774	14.398%	99129	11.425%	106816	20.065%	100249	12.684%	102714	15.454%	106998	20.270%	111865	25.740%
X-n876-k59	99299	119764	20.609%	116617	17.440%	119619	20.463%	114333	15.140%	114211	15.017%	117081	17.908%	113637	14.439%	111835	12.624%
X-n895-k37	53860	70245	30.421%	65587	21.773%	79018	46.710%	64310	19.402%	68084	26.409%	65014	20.709%	68888	27.902%	63392	17.698%
X-n916-k207	329179	399372	21.324%	361719	9.885%	383681	16.557%	374016	13.621%	369221	12.164%	361855	9.927%	362826	10.221%	359112	9.093%
X-n936-k151	132715	237625	79.049%	186262	40.347%	220926	66.466%	190407	43.471%	161290	21.531%	163037	22.847%	169127	27.436%	167240	26.014%
X-n957-k87	85465	130850	53.104%	98198	14.898%	113882	33.250%	105629	23.593%	107345	25.601%	105951	23.970%	109249	27.829%	110441	29.224%
X-n979-k58	118976	147687	24.132%	138092	16.067%	146347	23.005%	139682	17.404%	143799	20.864%	134406	12.969%	138048	16.030%	130503	9.689%
X-n1001-k43	72355	100399	38.759%	87660	21.153%	114448	58.176%	94734	30.929%	89986	24.367%	87594	21.061%	91289	26.168%	82558	14.101%
Avg. Gap		29.658%		16.796%		26.408%		19.607%		16.114%		15.336%		17.190%		**14.972%**	

In Fig. G.1 we report loss and costs curves for the ablation experiments. Our novel sampling technique leads to much more stable training and to a lower overall loss.

In Fig. G.2 we report additional results for our zero and few shot generalization experiment described in Section 5.3. Across all considered problem variants, EAL shows excellent performance.

Table G.2: Ablation of RouteFinder components.

Method	Gap ↓
w/o global attribute embedding	3.771 ± 0.113%
w/o mixed batch training	3.750 ± 0.121%
RouteFinder	**3.545** ± 0.111%

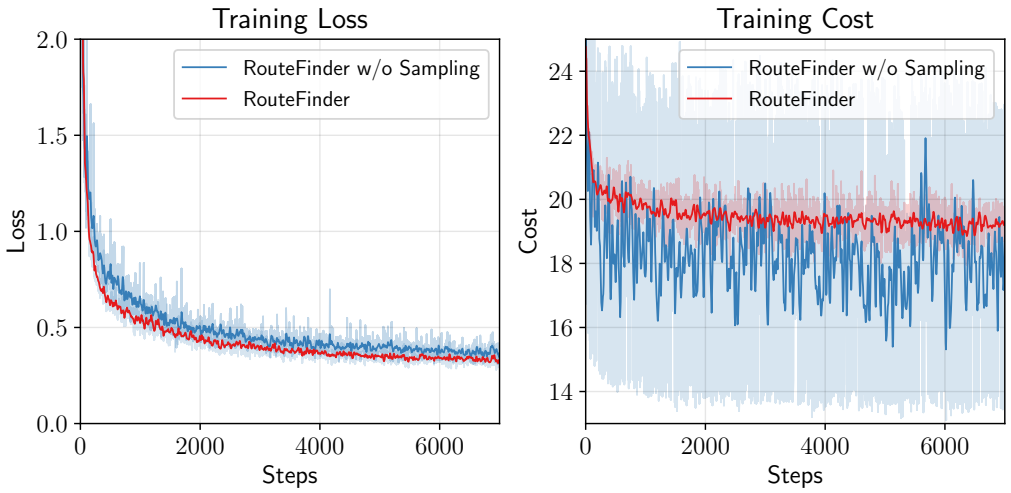

Figure G.1: Training loss and cost curve about the ablation study with and without sampling. We note how training with single-variant attributes biases the optimization toward lower costs - while this is, in fact, not the case in multi-task learning. The proposed sampling technique effectively stabilizes the optimization procedure.

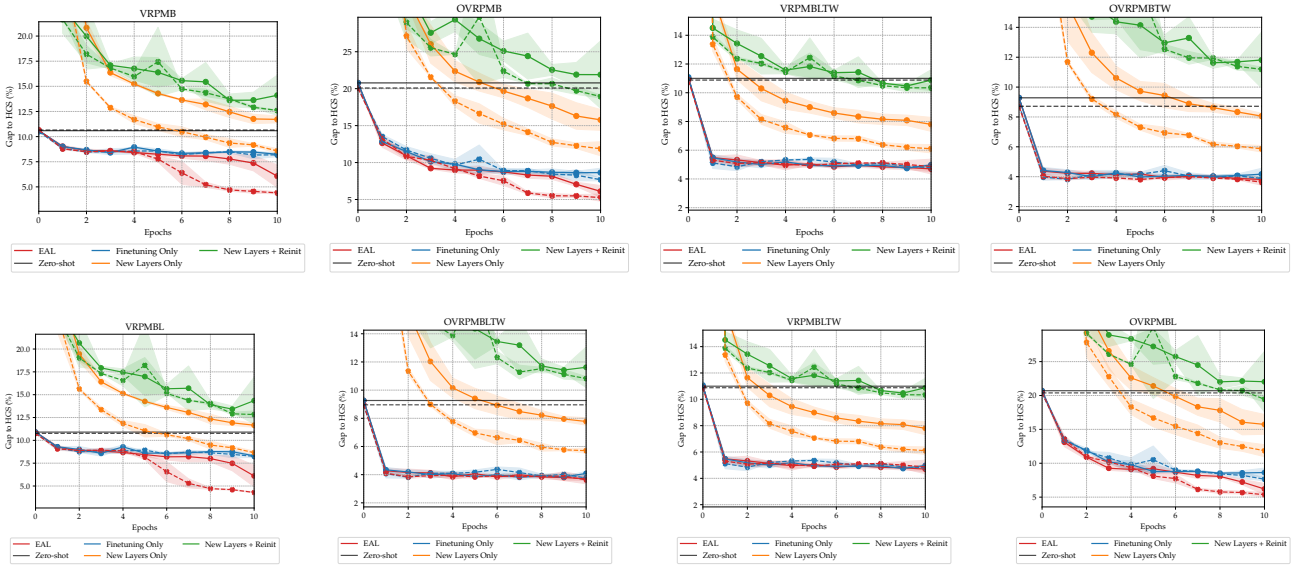

Figure G.2: Zero and few shot generalization to mixed (M) backhaul instances for 10 epochs with 10K instances per epoch. `RF`-POMO with "-" and `RF`-MoE-L is denoted by "- -" linestyles. `RouteFinder`'s Efficient Adapter Layers (EAL) enable fast finetuning to novel VRPs, even with *new attributes* that have never been seen by the model before.