MVMoE: Multi-Task Vehicle Routing Solver with Mixture-of-Experts

Jianan Zhou [1] **Zhiguang Cao** [2] **Yaoxin Wu** [3] **Wen Song** [4] **Yining Ma** [1] **Jie Zhang** [1] **Chi Xu** [1 5]

Abstract

Learning to solve vehicle routing problems (VRPs) has garnered much attention. However, most neural solvers are only structured and trained independently on a specific problem, making them less generic and practical. In this paper, we aim to develop a unified neural solver that can cope with a range of VRP variants simultaneously. Specifically, we propose a multi-task vehicle routing solver with mixture-of-experts (MVMoE), which greatly enhances the model capacity without a proportional increase in computation. We further develop a hierarchical gating mechanism for the MVMoE, delivering a good trade-off between empirical performance and computational complexity. Experimentally, our method significantly promotes zero-shot generalization performance on 10 unseen VRP variants, and showcases decent results on the few-shot setting and real-world benchmark instances. We further conduct extensive studies on the effect of MoE configurations in solving VRPs, and observe the superiority of hierarchical gating when facing out-of-distribution data. The source code is available at: https://github.com/RoyalSkye/Routing-MVMoE.

1. Introduction

Vehicle routing problems (VRPs) are a class of canonical combinatorial optimization problems (COPs) in operation research and computer science, with a wide spectrum of applications in logistics (Cattaruzza et al., 2017), transportation (Wu et al., 2023), and manufacturing (Zhang et al., 2023). The intrinsic NP-hard nature makes VRPs exponentially expensive to be solved by exact solvers. As an alternative, heuristic solvers deliver suboptimal solutions within reasonable time, but need substantial domain expertise to be designed for each problem. Recently, learning to solve VRPs has received much attention (Bengio et al., 2021; Bogyrbayeva et al., 2024), with fruitful neural solvers being developed. Most of them apply deep neural networks to learn solution construction policies via various training paradigms (e.g., reinforcement learning (RL)). Besides gaining decent performance, they are characterized by less computational overhead and domain expertise than conventional solvers. However, prevailing neural solvers still need network structures tailored and trained independently for each specific VRP, instigating prohibitive training overhead and less practicality when facing multiple VRPs.

In this paper, we aim to develop a unified neural solver, which can be trained for solving a range of VRP variants simultaneously, and has decent zero-shot generalization capability on unseen VRPs. A few recent works explore similar problem settings. Wang & Yu (2023) applies multi-armed bandits to solve multiple VRPs, while Lin et al. (2024) adapts the model pretrained on one base VRP to target VRPs by efficient fine-tuning. They fail to achieve zero-shot generalization to unseen VRPs due to the dependence on networks structured for predetermined problem variants. Liu et al. (2024) empowers the neural solver with such generalizability by the compositional zero-shot learning (Ruis et al., 2021), which treats VRP variants as different combinations of a set of underlying attributes and uses a shared network to learn their representations. However, it still leverages existing network structure proposed for simple VRPs, which is limited by its model capacity and empirical performance.

Motivated by the recent advance of large language models (LLMs) (Kaplan et al., 2020; Floridi & Chiriatti, 2020; Touvron et al., 2023), we propose a multi-task VRP solver with mixture-of-experts (MVMoE). Typically, a mixture-of-expert (MoE) layer replaces a feed-forward network (FFN) with several "experts" in a Transformer-based model, which are a group of FFNs with respective trainable parameters. An input to the MoE layer is routed to specific expert(s) by a gating network, and only parameters in selected expert(s)

[1]College of Computing and Data Science, Nanyang Technological University, Singapore [2]School of Computing and Information Systems, Singapore Management University, Singapore [3]Department of Information Systems, Eindhoven University of Technology, The Netherlands [4]Institute of Marine Science and Technology, Shandong University, China [5]Singapore Institute of Manufacturing Technology (SIMTech), Agency for Science, Technology and Research (A*STAR), Singapore. Correspondence to: Yaoxin Wu <y.wu2@tue.nl>.

Proceedings of the 41st International Conference on Machine Learning, Vienna, Austria. PMLR 235, 2024. Copyright 2024 by the author(s).

are activated (i.e., conditional computation (Jacobs et al., 1991; Jordan & Jacobs, 1994)). In this manner, partially activated parameters can effectively enhance the model capacity without a proportional increase in computation, making the training and deployment of LLMs viable. Therefore, towards a more generic and powerful neural solver, we first propose an MoE-based neural VRP solver, and present a hierarchical gating mechanism for a good trade-off between empirical performance and computational complexity. We choose the setting from Liu et al. (2024) as a test bed due to its potential to solve an exponential number of new VRP variants as any combination of the underlying attributes.

Our contributions are summarized as follows. 1) We propose a unified neural solver MVMoE to solve multiple VRPs, which first brings MoEs into the study of COPs. The sole MVMoE can be trained on diverse VRP variants, and facilitate a strong zero-shot generalization capability on unseen VRPs. 2) We develop a hierarchical gating mechanism for MVMoE to attain a favorable balance between empirical performance and computational overhead. Surprisingly, it exhibits much stronger out-of-distribution generalization capability than the base gating. 3) Extensive experiments demonstrate that MVMoE significantly improves the zero-shot generalization against baselines on 10 unseen VRP variants, and achieves decent results on the few-shot setting and real-world instances. We further provide extensive studies on the effect of MoE configurations (such as the position of MoEs, the number of experts, and the gating mechanism) on the zero-shot generalization performance.

2. Related Work

Neural VRP Solvers. Two mainstreams exist in literature on learning to solve VRPs: 1) *Construction-based solvers*, which learn policies to construct solutions in an end-to-end manner. Vinyals et al. (2015) proposes Pointer Network to estimate the optimal solution to the traveling salesman problem (TSP) in an autoregressive way. The follow-up works apply RL to explore better approximate solutions to TSP (Bello et al., 2017) and capacitated vehicle routing problem (CVRP) (Nazari et al., 2018). Kool et al. (2018) proposes an attention-based model (AM) that uses Transformer to solve a series of VRPs independently. By leveraging the symmetry property in solutions, Kwon et al. (2020) proposes the policy optimization with multiple optima (POMO) to further promote the performance in solving TSP and CVRP. Other construction-based solvers are often developed on top of AM and POMO (Kwon et al., 2021; Li et al., 2021a; Kim et al., 2022; Berto et al., 2023; Chen et al., 2023; Grinsztajn et al., 2023; Chalumeau et al., 2023; Hottung et al., 2024). Besides the autoregressive manner, several works construct a heatmap to solve VRPs in a non-autoregressive manner (Joshi et al., 2019; Fu et al., 2021; Kool et al., 2022; Qiu et al., 2022; Sun & Yang, 2023; Min et al., 2023; Ye et al., 2023; Kim et al., 2024). 2) *Improvement-based solvers,* which learn policies to iteratively refine an initial solution until a termination condition is satisfied. The policies are often trained in contexts of classic local search (Croes, 1958; Shaw, 1998) or specialized heuristic solvers (Helsgaun, 2017) for obtaining more efficient or effective search components (Chen & Tian, 2019; Lu et al., 2020; Hottung & Tierney, 2020; d O Costa et al., 2020; Wu et al., 2021; Xin et al., 2021; Hudson et al., 2022; Zhou et al., 2023a; Ma et al., 2023). In general, construction-based solvers can efficiently achieve desired performance, whereas improvement-based solvers have the potential to deliver better solutions given prolonged inference time.

Recent research uncovers the deficient generalization capability of neural solvers, which suffer from drastic performance decrement on unseen data (Joshi et al., 2021). Previous works mainly focus on the cross-size generalization (Fu et al., 2021; Hou et al., 2023; Son et al., 2023; Luo et al., 2023; Drakulic et al., 2023) or cross-distribution generalization (Zhang et al., 2022; Geisler et al., 2022; Bi et al., 2022; Jiang et al., 2023) or both (Manchanda et al., 2022; Zhou et al., 2023b; Wang et al., 2024) on a single problem. In this paper, we step further to explore the generalization across different VRP variants (Wang & Yu, 2023; Liu et al., 2024; Lin et al., 2024).

Mixture-of-Experts. The original idea of MoEs was proposed three decades ago (Jacobs et al., 1991; Jordan & Jacobs, 1994). In early concepts, the expert was defined as an entire neural network, and hence MoEs was similar to an ensemble of neural networks. Eigen et al. (2013) launchs the era when researchers start applying MoEs as components of neural networks. As an early success of MoEs applied in large neural networks, Shazeer et al. (2017) introduces the sparsely-gated MoEs in language modeling and machine translation, achieving state-of-the-art results at the time with only minor losses in computational efficiency. Follow-up works mainly focus on improving the gating mechanism (Lewis et al., 2021; Roller et al., 2021; Zuo et al., 2022; Zhou et al., 2022; Puigcerver et al., 2024; Xue et al., 2024) or applications to other domains (Lepikhin et al., 2020; Riquelme et al., 2021; Fedus et al., 2022b). We refer interested readers to Yuksel et al. (2012); Fedus et al. (2022a) for a comprehensive survey.

3. Preliminaries

In this section, we first present the definition of CVRP, and then introduce its variants featured by additional constraints. Afterwards, we delineate recent construction-based neural solvers for VRPs (Kool et al., 2018; Kwon et al., 2020).

VRP Variants. We define a CVRP instance of size n over

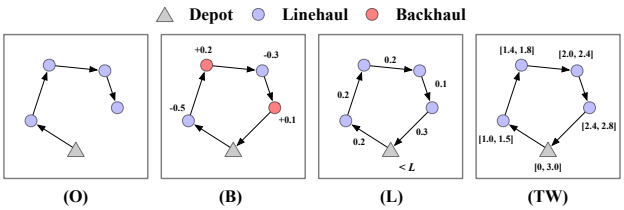

Figure 1. Illustrations of sub-tours with various constraints: open route (O), backhaul (B), duration limit (L), and time window (TW).

a graph $\mathcal{G} = \{\mathcal{V}, \mathcal{E}\}$, where $\mathcal{V}$ includes a depot node v_0 and customer nodes $\{v_i\}_{i=1}^n$, and $\mathcal{E}$ includes edges $e(v_i, v_j)$ between node v_i and $v_j (i \neq j)$. Each customer node is associated with a demand δ_i, and a capacity limit Q is set for each vehicle. The solution (i.e., tour) τ is represented as a sequence of nodes, consisting of multiple sub-tours. Each sub-tour represents that a vehicle starts from the depot, visits a subset of customer nodes and returns to the depot. The solution is feasible if each customer node is visited exactly once, and the total demand in each sub-tour does not exceed the capacity limit Q. We consider the Euclidean space with the cost function $c(\cdot)$ defined as the total length of the tour. The objective is to find the optimal tour τ^* with the minimal cost: $\tau^* = \arg\min_{\tau \in \Phi} c(\tau|\mathcal{G})$, where Φ is the discrete search space that contains all feasible tours.

On top of CVRP (featured by the capacity constraint *(C)*), several VRP variants involve additional practical constraints. 1) *Open Route (O):* The vehicle does not need to return to the depot v_0 after visiting customers; 2) *Backhaul (B):* The demand δ_i is positive in CVRP, representing a vehicle unloads goods at the customer node. In practice, a customer can have a negative demand, requiring a vehicle to load goods. We name the customer nodes with $\delta_i > 0$ as linehauls and the ones with $\delta_i < 0$ as backhauls. Hence, VRP with backhaul allows the vehicle traverses linehauls and backhauls in a mixed manner, without strict precedence between them; 3) *Duration Limit (L):* To maintain a reasonable workload, the cost (i.e., length) of each route is upper bounded by a predefined threshold; 4) *Time Window (TW):* Each node $v_i \in \mathcal{V}$ is associated with a time window $[e_i, l_i]$ and a service time s_i. A vehicle must start serving customer v_i in the time slot from e_i to l_i. If the vehicle arrives earlier than e_i, it has to wait until e_i. All vehicles must return to the depot v_0 no later than l_0. The aforementioned constraints are illustrated in Fig. 1. By combining them, we can obtain 16 typical VRP variants, which are summarized in Table 3. Note that the combination is not a trivial addition of different constraints. For example, when the open route is coupled with the time window, the vehicle does not need to return to the depot, and hence the constraint imposed by l_0 at the depot is relaxed. We present more details of VRP variants and the associated data generation process in Appendix A.

Learning to Solve VRPs. Typical neural solvers (Kool et al., 2018; Kwon et al., 2020) parameterize the solution construction policy by an attention-based neural network π_θ, which is trained to generate a solution in an autoregressive way. The feasibility of the generated solution is guaranteed by the masking mechanism during decoding. Without loss of generality, we consider RL training paradigm, wherein the solution construction process is formulated as a Markov Decision Process (MDP). Given an input instance, the encoder processes it and attains all node embeddings, which, with the context representation of the constructed partial tour, represent the current state. The decoder takes them as inputs and outputs the probabilities of valid nodes (i.e., actions) to be selected. After a complete solution τ is constructed, its probability can be factorized via the chain rule such that $p_\theta(\tau|\mathcal{G}) = \prod_{t=1}^{T} p_\theta(\pi_\theta^{(t)}|\pi_\theta^{(<t)}, \mathcal{G})$, where $\pi_\theta^{(t)}$ and $\pi_\theta^{(<t)}$ denote the selected node and constructed partial tour at step t, and T is the number of total steps. The reward is defined as the negative tour length, i.e., $\mathcal{R} = -c(\tau|\mathcal{G})$. Given a baseline function $b(\cdot)$ for training stability, the policy network π_θ is often trained by REINFORCE (Williams, 1992) algorithm, which applies estimated gradients of the expected reward to optimize the policy as below,

$$\nabla_\theta \mathcal{L}_a(\theta|\mathcal{G}) = \mathbb{E}_{p_\theta(\tau|\mathcal{G})}[(c(\tau) - b(\mathcal{G}))\nabla_\theta \log p_\theta(\tau|\mathcal{G})]. \quad (1)$$

4. Methodology

In this section, we present the multi-task VRP solver with MoEs (MVMoE), and introduce the gating mechanism. Without loss of generality, we aim to learn a construction-based neural solver (Kool et al., 2018; Kwon et al., 2020) for tackling VRP variants with the five constraints introduced in Section 3. The structure of MVMoE is illustrated in Fig. 2.

4.1. Multi-Task VRP Solver with MoEs

Multi-Task VRP Solver. Given an instance of a specific VRP variant, the *static* features of each node v_i are expressed by $\mathcal{S}_i = \{y_i, \delta_i, e_i, l_i\}$, where y_i, δ_i, e_i, l_i denote the coordinate, demand, start and end time of the time window, respectively. The encoder takes these static node features as inputs, and outputs d-dimensional node embeddings h_i. At the t_{th} decoding step, the decoder takes as input the node embeddings and a context representation, including the embedding of the last selected node and *dynamic* features $\mathcal{D}_t = \{c_t, t_t, l_t, o_t\}$, where c_t, t_t, l_t, o_t denote the remaining capacity of the vehicle, the current time, the length of the current partial route, and the presence indicator of the open route, respectively. Thereafter, the decoder outputs the probability distribution of nodes, from which a valid node is selected and appended to the partial solution. A complete solution is constructed in an autoregressive manner by iterating the decoding process.

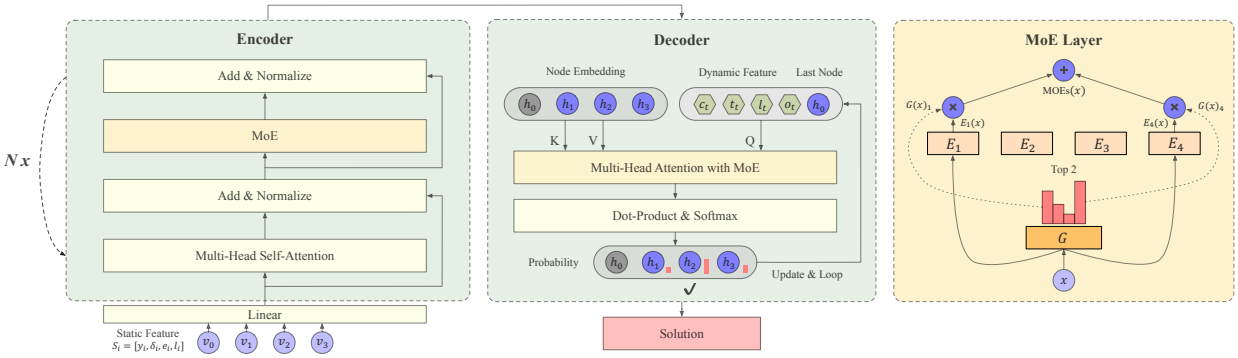

Figure 2. The model structure of MVMoE. [*Green part*]: Given an input instance, the encoder and decoder output node embeddings and probabilities of nodes to be selected, respectively. The gray nodes are masked to satisfy problem-specific constraints for feasibility. The node with a deeper color denote a later node embedding. [*Yellow part*]: In an MoE layer, where we take the (node-level) input-choice Top2 gating as an example, the input x (i.e., node) is routed to two experts that derive the two largest probabilities from the gating network G.

In each training step, we randomly select a VRP variant, and train the neural network to solve associated instances in a batch. In this way, MVMoE is able to learn a unified policy that can tackle different VRP tasks. If only a subset of static or dynamic features are involved in the current selected VRP variant, the other features are padded to the default values (e.g., zeros). For example, given a CVRP instance, the static features of the i_{th} customer node are $\mathcal{S}_i^{(C)} = \{y_i, \delta_i, 0, 0\}$, and the dynamic features at the t_{th} decoding step are $\mathcal{D}_t^{(C)} = \{c_t, 0, l_t, 0\}$. In summary, motivated by the fact that different VRP variants may include some common attributes (e.g., coordinate, demand), we define the static and dynamic features as the union set of attributes that exist in all VRP variants. By training on a few VRP variants with these attributes, the policy network has the potential to solve unseen variants, which are characterized by different combinations of these attributes, i.e., the zero-shot generalization capability (Liu et al., 2024).

Mixture-of-Experts. Typically, an MoE layer consists of 1) m experts $\{E_1, E_2, \ldots, E_m\}$, each of which is a linear layer or FFN with independent trainable parameters, and 2) a gating network G parameterized by W_G, which decides how the inputs are distributed to experts. Given a single input x, $G(x)$ and $E_j(x)$ denote the output of the gating network (i.e., an m-dimensional vector), and the output of the j_{th} expert, respectively. In light of this, the output of an MoE layer is calculated as,

$$\text{MoE}(x) = \sum_{j=1}^{m} G(x)_j E_j(x). \tag{2}$$

Intuitively, a sparse vector $G(x)$ only activates a small subset of experts with partial model parameters, and hence saves the computation. Typically, a Top$K$ operator can achieve such sparsity by only keeping the K-largest values while setting others as the negative infinity. In this case, the gating network calculates the output as $G(x) = \text{Softmax}(\text{Top}K(x \cdot W_G))$. Given the fact that larger sparse models do not always lead to better performance (Zuo et al., 2022), it is crucial yet tricky to *design effective and efficient gating mechanisms to endow each expert being sufficiently trained, given enough training data.* To this effect, some works have been put forward in language and vision domains, such as designing an auxiliary loss (Shazeer et al., 2017) or formulating it as a linear assignment problem (Lewis et al., 2021) in pursuit of the load balancing.

MVMoE. By integrating the above parts, we obtain the multi-task VRP solver with MoEs. The overall model structure is shown in Fig. 2, where we employ MoEs in both the encoder and decoder. In specific, we substitute MoEs for the FFN layer in the encoder, and substitute MoEs for the final linear layer of multi-head attention in the decoder. We refer more details of the structure of MVMoE to Appendix B. We empirically find our design is effective in generating high-quality solutions, and especially employing MoEs in the decoder tends to exert a greater influence on performance (see Section 5.2). We jointly optimize all trainable parameters Θ, with the objective formulated as follows,

$$\min_{\Theta} \mathcal{L} = \mathcal{L}_a + \alpha \mathcal{L}_b, \tag{3}$$

where $\mathcal{L}_a$ denotes the original loss function of the VRP solver (e.g., the REINFORCE loss used to train the policy for solving VRP variants in Eq. (1)), $\mathcal{L}_b$ denotes the loss function associated with MoEs (e.g., the auxiliary loss used to ensure load balancing in Eq. (19) in Appendix B), and α is a hyperparameter to control its strength.

4.2. Gating Mechanism

We mainly consider the node-level (or token-level) gating, by which each node is routed independently to experts.[1] In

[1] In addition, we also investigate another two gating levels, i.e., instance-level and problem-level gating, which are presented in Section 5.2 and Appendix B.

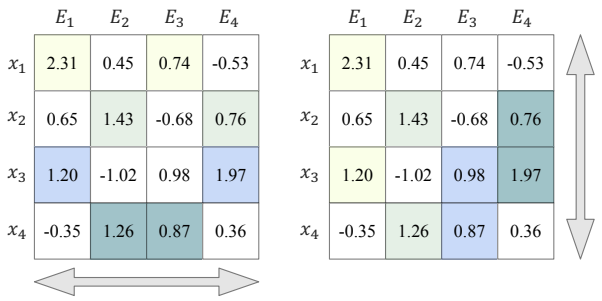

Figure 3. An illustration of the score matrix and gating algorithm. *Left panel:* Input-choice gating. *Right panel:* Expert-choice gating. The selected experts or nodes are in color. The arrow marks the dimension, along which the Top$K$ experts or nodes are selected.

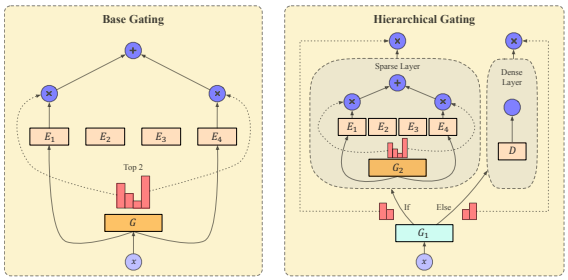

Figure 4. A base gating (i.e., the input-choice gating with $K = 2$) and its hierarchical gating counterpart. In the latter, the gating network G_1 routes inputs to the sparse layer ($\{G_2, E_1, E_2, E_3, E_4\}$) or the dense layer D. If the sparse layer is chosen, the gating network G_2 routes nodes to experts accoring to the base gating.

each MoE layer, the extra computation originates from the forward pass of the gating network and the distribution of nodes to the selected experts. While employing MoEs in the decoder can significantly improve the performance, the number of decoding steps T increases as the problem size n scales up. It suggests that compared to the encoder with a fixed number of gating steps N ($\ll T$), applying MoEs in the decoder may substantially increase the computational complexity. In light of this, we propose a hierarchical gating mechanism to make the better use of MoEs in the decoder for gaining a good trade-off between empirical performance and computational complexity. Next, we detail the node-level and hierarchical gating mechanism.

Node-Level Gating. The node-level gating routes inputs at the granularity of nodes. Let d denote the hidden dimension and $W_G \in \mathbb{R}^{d \times m}$ denote trainable parameters of the gating network in MVMoE. Given a batch of inputs $X \in \mathbb{R}^{I \times d}$, where I is the total number of nodes (i.e., batch size B $\times$ problem scale n), each node is routed to the selected experts based on the score matrix $H = (X \cdot W_G) \in \mathbb{R}^{I \times m}$ predicted by the gating network. We illustrate an example of the score matrix in Fig. 3, where x_i denotes the i_{th} node, and E_j denotes the j_{th} expert in the node-level gating.

In this paper, we mainly consider two popular gating algorithms (Shazeer et al., 2017; Zhou et al., 2022): 1) *Input-choice gating:* Each node selects Top$K$ experts based on H. Typically, K is set to 1 or 2 to retain a reasonable computational complexity. The input-choice gating is illustrated in the left panel of Fig. 3, where each node is routed to two experts with the largest scores (i.e., Top2). However, this method cannot guarantee load balancing. An expert may receive much more nodes than the others, resulting in a dominant expert while leaving others underfitting. To address this issue, most works employ an auxiliary loss to equalize quantities of nodes sent to different experts during training. Here we use the importance & load loss (Shazeer et al., 2017) as $\mathcal{L}_b$ in Eq. (3) to mitigate load imbalance (see Appendix B). 2) *Expert-choice gating:* Each expert selects Top$K$ nodes based on H. Typically, K is set to $\frac{I \times \beta}{m}$, where β is the capacity factor reflecting the average number of experts utilized by a node. The expert-choice gating is illustrated in the right panel of Fig. 3, where each expert selects two nodes with the largest scores given $\beta = 2$. While this gating algorithm explicitly ensures load balancing, some nodes may not be chosen by any expert. We refer more details of the above gating algorithms to Appendix B.

Hierarchical Gating. In the VRP domain, it is computationally expensive to employ MoEs in each decoding step, since 1) the number of decoding steps T increases as the problem size n rises; 2) the problem-specific feasibility constraints must be satisfied during decoding. To tackle the challenges, we propose to employ MoEs only in *partial* decoding steps. Accordingly, we present a hierarchical gating, which learns to effectively and efficiently utilize MoEs during decoding.

We illustrate the proposed hierarchical gating in Fig. 4. An MoE layer with the hierarchical gating includes two gating networks $\{G_1, G_2\}$, m experts $\{E_1, E_2, \ldots, E_m\}$, and a dense layer D (e.g., a linear layer). Given a batch of inputs $X \in \mathbb{R}^{I \times d}$, the hierarchical gating routes them in two stages. In the first stage, G_1 decides to distribute inputs X to either the sparse or dense layer according to the problem-level representation X_1. In specific, we obtain X_1 by applying the mean pooling along the first dimension of X, and process it to obtain the score matrix $H_1 = (X_1 \cdot W_{G_1}) \in \mathbb{R}^{1 \times 2}$. Then, we route the batch of inputs X to the sparse or dense layer by sampling from the probability distribution $G_1(X) = \text{Softmax}(H_1)$. Here we employ the problem-level gating in G_1 for the generality and efficiency of the hierarchical gating (see Appendix D for further discussions). In the second stage, if X is routed to the sparse layer, the gating network G_2 is activated to route nodes to experts on the node-level by using aforementioned gating algorithms (e.g., the input-choice gating). Otherwise, X is routed to the dense layer D and transformed into $D(X) \in \mathbb{R}^{I \times d}$. In summary, the hierarchical gating learns to output $G_1(X)_0 \sum_{j=1}^{m} G_2(X)_j E_j(X)$ or $G_1(X)_1 D(X)$

Table 1. Performance on 1K test instances of trained VRPs. * represents 0.000%, with which the gaps are computed.

	Method	$n = 50$			$n = 100$				Method	$n = 50$			$n = 100$		
		Obj.	Gap	Time	Obj.	Gap	Time			Obj.	Gap	Time	Obj.	Gap	Time
CVRP	HGS	10.334	*	4.6m	15.504	*	9.1m	VRPTW	HGS	14.509	*	8.4m	24.339	*	19.6m
	LKH3	10.346	0.115%	9.9m	15.590	0.556%	18.0m		LKH3	14.607	0.664%	5.5m	24.721	1.584%	7.8m
	OR-Tools	10.540	1.962%	10.4m	16.381	5.652%	20.8m		OR-Tools	14.915	2.694%	10.4m	25.894	6.297%	20.8m
	OR-Tools (x10)	10.418	0.788%	1.7h	15.935	2.751%	3.5h		OR-Tools (x10)	14.665	1.011%	1.7h	25.212	3.482%	3.5h
	POMO	10.418	0.806%	3s	15.734	1.488%	9s		POMO	14.940	2.990%	3s	25.367	4.307%	11s
	POMO-MTL	10.437	0.987%	3s	15.790	1.846%	9s		POMO-MTL	15.032	3.637%	3s	25.610	5.313%	11s
	MVMoE/4E	**10.428**	**0.896%**	4s	**15.760**	**1.653%**	11s		MVMoE/4E	**14.999**	**3.410%**	4s	**25.512**	**4.903%**	12s
	MVMoE/4E-L	10.434	0.955%	4s	15.771	1.728%	10s		MVMoE/4E-L	15.013	3.500%	3s	25.519	4.927%	11s
OVRP	LKH3	6.511	0.198%	4.5m	9.828	*	5.3m	VRPL	LKH3	10.571	0.790%	7.8m	15.771	*	16.0m
	OR-Tools	6.531	0.495%	10.4m	10.010	1.806%	20.8m		OR-Tools	10.677	1.746%	10.4m	16.496	4.587%	20.8m
	OR-Tools (x10)	6.498	*	1.7h	9.842	0.122%	3.5h		OR-Tools (x10)	10.495	*	1.7h	16.004	1.444%	3.5h
	POMO	6.609	1.685%	2s	10.044	2.192%	8s		POMO	10.491	-0.008%	2s	15.785	0.093%	9s
	POMO-MTL	6.671	2.634%	2s	10.169	3.458%	8s		POMO-MTL	10.513	0.201%	2s	15.846	0.479%	9s
	MVMoE/4E	**6.655**	**2.402%**	3s	**10.138**	**3.136%**	10s		MVMoE/4E	**10.501**	**0.092%**	3s	**15.812**	**0.261%**	10s
	MVMoE/4E-L	6.665	2.548%	3s	10.145	3.214%	9s		MVMoE/4E-L	10.506	0.131%	3s	15.821	0.323%	10s
VRPB	OR-Tools	8.127	0.989%	10.4m	12.185	2.594%	20.8m	OVRPTW	OR-Tools	8.737	0.592%	10.4m	14.635	1.756%	20.8m
	OR-Tools (x10)	8.046	*	1.7h	11.878	*	3.5h		OR-Tools (x10)	8.683	*	1.7h	14.380	*	3.5h
	POMO	8.149	1.276%	2s	11.993	0.995%	7s		POMO	8.891	2.377%	3s	14.728	2.467%	10s
	POMO-MTL	8.182	1.684%	2s	12.072	1.674%	7s		POMO-MTL	8.987	3.470%	3s	15.008	4.411%	10s
	MVMoE/4E	**8.170**	**1.540%**	3s	**12.027**	**1.285%**	9s		MVMoE/4E	**8.964**	**3.210%**	4s	**14.927**	**3.852%**	11s
	MVMoE/4E-L	8.176	1.605%	3s	12.036	1.368%	8s		MVMoE/4E-L	8.974	3.322%	4s	14.940	3.941%	10s

based on both problem-level and node-level representations.

Overall, the hierarchical gating improves the computational efficiency with a minor loss on the empirical performance. To balance the efficiency and performance of MVMoE, we use the base gating in the encoder and its hierarchical gating counterpart in the decoder. Note that the hierarchical gating is applicable to different gating algorithms, such as the input-choice gating (Shazeer et al., 2017) and expert-choice gating (Zhou et al., 2022). We also explore a more advanced gating algorithm (Puigcerver et al., 2024) for reducing the number of routed nodes and thus the computational complexity. But its empirical performance is unsatisfactory in the VRP domain (see Section 5.3).

5. Experiments

In this section, we empirically verify the superiority of the proposed MVMoE, and provide insights into the application of MoEs to solve VRPs. We consider 16 VRP variants with five constraints. Due to page limit, we present more experimental results in Appendix C. All experiments are conducted on a machine with NVIDIA Ampere A100-80GB GPU cards and AMD EPYC 7513 CPU at 2.6GHz.

Baselines. *Traditional solvers:* We employ HGS (Vidal, 2022) to solve CVRP and VRPTW instances with default hyperparameters (i.e., the maximum number of iterations without improvement is 20000). We run LKH3 (Helsgaun, 2017) to solve CVRP, OVRP, VRPL and VRPTW instances with 10000 trails and 1 run. OR-Tools (Furnon & Perron, 2023) is an open source solver for complex optimization problems. It is more versatile than LKH and HGS, and can solve all 16 VRP variants considered in this paper. We use the parallel cheapest insertion as the first solution strategy, and use the guided local search as the local search strategy in OR-Tools. For $n = 50/100$, we set the search time limit as 20s/40s to solve an instance, and also provide its results given 200s/400s (i.e., OR-Tools (x10)). For all traditional solvers, we use them to solve 32 instances in parallel on 32 CPU cores following Kool et al. (2018).

Neural solvers: We compare our method to POMO (Kwon et al., 2020) and POMO-MTL (Liu et al., 2024). While POMO is trained on each single VRP, POMO-MTL is trained on multiple VRPs by multi-task learning. Note that POMO-MTL is the dense model counterpart of MVMoE, which is structured by dense layers (e.g., FFNs) rather than sparse MoEs. In specific, POMO-MTL and MVMoE/4E possess 1.25M and 3.68M parameters, but they activate a similar number of parameters for each single input.

Training. We follow most setups in (Kwon et al., 2020). 1) *For all neural solvers:* Adam optimizer is used with the learning rate of $1e-4$, the weight decay of $1e-6$, and the batch size of 128. The model is trained for 5000 epochs, with each containing 20000 training instances (i.e., 100M training instances in total). The learning rate is decayed by 10 for the last 10% training instances. We consider two problem scales $n \in \{50, 100\}$ during training, according to (Liu et al., 2024). 2) *For multi-task solvers:* The training problem set includes CVRP, OVRP, VRPB, VRPL, VRPTW, and OVRPTW (see Appendix C.1 for further discussions). In each batch of training, we randomly sample a problem from the set and generate its instances. Please refer to Appendix A for details of the generation procedure. 3) *For our method:* We employ $m = 4$ experts with $K = \beta = 2$ in each MoE layer, and set the the weight α of the auxiliary loss $\mathcal{L}_b$ as 0.01. The default gating mechanism of MVMOE/4E is the node-level input-choice gating in both the encoder

Table 2. Zero-shot generalization on 1K test instances of unseen VRPs. * represents 0.000%, with which the gaps are computed.

	Method	$n = 50$			$n = 100$				Method	$n = 50$			$n = 100$		
		Obj.	Gap	Time	Obj.	Gap	Time			Obj.	Gap	Time	Obj.	Gap	Time
OVRPB	OR-Tools	5.764	0.332%	10.4m	8.522	1.852%	20.8m	OVRPL	OR-Tools	6.522	0.480%	10.4m	9.966	1.783%	20.8m
	OR-Tools (x10)	5.745	*	1.7h	8.365	*	3.5h		OR-Tools (x10)	6.490	*	1.7h	9.790	*	3.5h
	POMO-MTL	6.116	6.430%	2s	8.979	7.335%	8s		POMO-MTL	6.668	2.734%	2s	10.126	3.441%	9s
	MVMoE/4E	**6.092**	**5.999%**	3s	**8.959**	**7.088%**	9s		MVMoE/4E	**6.650**	**2.454%**	3s	**10.097**	**3.148%**	10s
	MVMoE/4E-L	6.122	6.522%	3s	8.972	7.243%	9s		MVMoE/4E-L	6.659	2.597%	3s	10.106	3.244%	9s
VRPBL	OR-Tools	8.131	1.254%	10.4m	12.095	2.586%	20.8m	VRPBTW	OR-Tools	15.053	1.857%	10.4m	26.217	2.858%	20.8m
	OR-Tools (x10)	8.029	*	1.7h	11.790	*	3.5h		OR-Tools (x10)	14.771	*	1.7h	25.496	*	3.5h
	POMO-MTL	8.188	1.971%	2s	11.998	1.793%	8s		POMO-MTL	16.055	8.841%	3s	27.319	7.413%	10s
	MVMoE/4E	**8.172**	**1.776%**	3s	**11.945**	**1.346%**	9s		MVMoE/4E	**16.022**	**8.600%**	4s	**27.236**	**7.078%**	11s
	MVMoE/4E-L	8.180	1.872%	3s	11.960	1.473%	9s		MVMoE/4E-L	16.041	8.745%	4s	27.265	7.190%	10s
VRPLTW	OR-Tools	14.815	1.432%	10.4m	25.823	2.534%	20.8m	OVRPBL	OR-Tools	5.771	0.549%	10.4m	8.555	2.459%	20.8m
	OR-Tools (x10)	14.598	*	1.7h	25.195	*	3.5h		OR-Tools (x10)	5.739	*	1.7h	8.348	*	3.5h
	POMO-MTL	14.961	2.586%	3s	25.619	1.920%	12s		POMO-MTL	6.104	6.306%	2s	8.961	7.343%	8s
	MVMoE/4E	**14.937**	**2.421%**	4s	**25.514**	**1.471%**	13s		MVMoE/4E	**6.076**	**5.843%**	3s	**8.942**	**7.115%**	9s
	MVMoE/4E-L	14.953	2.535%	4s	25.529	1.545%	12s		MVMoE/4E-L	6.104	6.310%	3s	8.957	7.300%	9s
OVRPBTW	OR-Tools	8.758	0.927%	10.4m	14.713	2.268%	20.8m	OVRPLTW	OR-Tools	8.728	0.656%	10.4m	14.535	1.779%	20.8m
	OR-Tools (x10)	8.675	*	1.7h	14.384	*	3.5h		OR-Tools (x10)	8.669	*	1.7h	14.279	*	3.5h
	POMO-MTL	9.514	9.628%	3s	15.879	10.453%	10s		POMO-MTL	8.987	3.633%	3s	14.896	4.374%	11s
	MVMoE/4E	**9.486**	**9.308%**	4s	**15.808**	**9.948%**	11s		MVMoE/4E	**8.966**	**3.396%**	4s	**14.828**	**3.903%**	12s
	MVMoE/4E-L	9.515	9.630%	3s	15.841	10.188%	10s		MVMoE/4E-L	8.974	3.488%	4s	14.839	3.971%	10s
VRPBLTW	OR-Tools	14.890	1.402%	10.4m	25.979	2.518%	20.8m	OVRPBLTW	OR-Tools	8.729	0.624%	10.4m	14.496	1.724%	20.8m
	OR-Tools (x10)	14.677	*	1.7h	25.342	*	3.5h		OR-Tools (x10)	8.673	*	1.7h	14.250	*	3.5h
	POMO-MTL	15.980	9.035%	3s	27.247	7.746%	11s		POMO-MTL	9.532	9.851%	3s	15.738	10.498%	10s
	MVMoE/4E	**15.945**	**8.775%**	4s	**27.142**	**7.332%**	12s		MVMoE/4E	**9.503**	**9.516%**	4s	**15.671**	**10.009%**	11s
	MVMoE/4E-L	15.963	8.915%	4s	27.177	7.473%	11s		MVMoE/4E-L	9.518	9.682%	4s	15.706	10.263%	10s

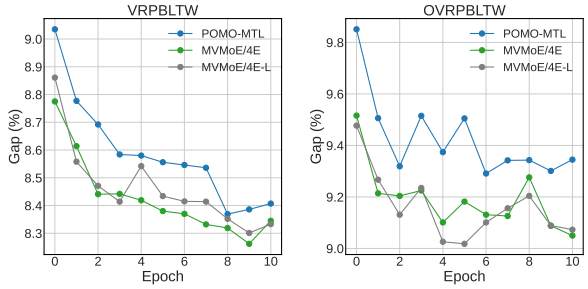

Figure 5. Few-shot generalization on unseen VRPs.

and decoder layers. MVMoE/4E-L is a computationally light version that replaces the input-choice gating with its hierarchical gating counterpart in the decoder.

Inference. For all neural solvers, we use greedy rollout with x8 instance augmentation following Kwon et al. (2020). We report the average results (i.e., objective values and gaps) over the test dataset that contains 1K instances, and the total time to solve the entire test dataset. The gaps are computed with respect to the results of the best-performing traditional VRP solvers (i.e., * in Tables 1 and 2).

5.1. Empirical Results

Performance on Trained VRPs. We evaluate all methods on 6 trained VRPs and gather all results in Table 1. The single-task neural solver (i.e., POMO) achieves better performance than multi-task neural solvers on each single problem, since it is restructured and retrained on each VRP independently. However, its average performance over all trained VRPs is quite inferior as shown in Table 4 in Appendix C, since each trained POMO is overfitted to a specific VRP. For example, the average performance of POMO solely trained on CVRP is 16.815%, while POMO-MTL and MVMoE/4E achieve 2.102% and 1.925%, respectively. Notably, our neural solvers consistently outperform POMO-MTL. MVMoE/4E performs slightly better than MVMoE/4E-L at the expense of more computation. Despite that, MVMoE/4E-L exhibits stronger out-of-distribution generalization capability than MVMoE/4E (see Tables 7 and 8 in Appendix C).

Generalization on Unseen VRPs. We evaluate multi-task solvers on 10 unseen VRP variants. 1) *Zero-shot generalization:* We directly test the trained solvers on unseen VRPs. The results in Table 2 reveal that the proposed MVMoE significantly outperforms POMO-MTL across all VRP variants. 2) *Few-shot generalization:* We also consider the few-shot setting on $n = 50$, where a trained solver is fine-tuned on the target VRP using 10K instances (0.01% of total training instances) in each epoch. Without loss of generality, we conduct experiments on VRPBLTW and OVRPBLTW following the training setups. The results in Fig. 5 showcase MVMoE generalizes more favorably than POMO-MTL.

5.2. Ablation on MoEs

Here we explore the effect of different MoE settings on the zero-shot generalization of neural solvers, and provide insights on how to effectively apply MoEs to solve VRPs. Due to the fast convergence, we reduce the number of epochs to 2500 on VRPs of the size $n = 50$, while leaving other setups unchanged. We set MVMoE/4E as the default baseline,

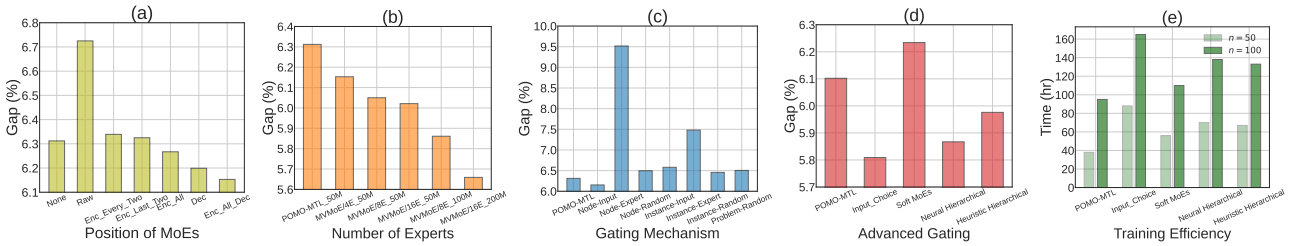

Figure 6. Left three panels: The effect of MoE settings on the average zero-shot generalization performance - **(a)** the position of MoEs; **(b)** the number of experts; **(c)** the gating mechanism. *Right two panels:* Further analyses - **(d)** average zero-shot generalization performance of each method employing various gating algorithms in the decoder; **(e)** training efficiency of each gating algorithm.

and ablate on different components of MoEs below.

Position of MoEs. We consider three positions to apply MoEs in neural solvers: 1) *Raw feature processing (Raw):* The linear layer, which projects raw features into initial embeddings, are replaced by MoEs. 2) *Encoder (Enc):* The FFN in an encoder layer is replaced by MoEs. Typically, MoEs are widely used in *every-two* or *last-two* layers (i.e., every or last two layers with even indices $\ell \in [0, N-1]$) (Riquelme et al., 2021). Besides, we further attempt to use MoEs in all encoder layers. 3) *Decoder (Dec):* The final linear layer of the multi-head attention is replaced by MoEs in the decoder. We show the average performance over 10 unseen VRPs in Fig. 6(a). The results reveal that applying MoEs at the shallow layer (e.g., Raw) may worsen the model performance, while using MoEs in all encoder layers (Enc_All) or decoder (Dec) can benefit the zero-shot generalization. Therefore, in this paper, we employ MoEs in both encoder and decoder to pursue a strong unified model architecture to solve various VRPs.

Number of Experts. We increase the number of experts in each MoE layer to 8 and 16, and compare the derived MVMoE/8E/16E models to MVMoE/4E. We first train all models using the same number (50M) of instances. After that, we also train MVMoE/8E/16E with more data and computation to explore potential better results, based on the scaling laws (Kaplan et al., 2020). In specific, we provide MVMoE/8E/16E with more data by using larger batch sizes, which linearly scale up against the number of experts (i.e., MVMoE/4E/8E/16E are trained on 50M/100M/200M instances with batch sizes 128/256/512, respectively). The results in Fig. 6(b) show that increasing the number of experts with more training data further unleashes the power of MVMoE, indicating the efficacy of MoEs in solving VRPs.

Gating Mechanism. We investigate the effect of different gating levels and algorithms, including three levels (i.e., node-level, instance-level and problem-level) and three algorithms (i.e., input-choice, expert-choice and random gatings), with their details presented in Appendix B. As shown in Fig. 6(c), the node-level input-choice gating performs the best, while the node-level expert-choice gating performs the worst. Interestingly, we observe that the expert-choice gating in the decoder makes MVMoE hard to be optimized. It may suggest that each gating algorithm could have its most suitable position to serve MoEs. However, after an attempt to tune this configuration (i.e., by using MoEs only in the encoder), its performance is still inferior to the baseline, with an average gap of 7.190% on unseen VRPs.

5.3. Additional Results

We further provide experiments and discussions on more advanced gating algorithms, training efficiency, benchmark performance, and scalability. We refer readers to more empirical results (e.g., sensitivity analyses) in Appendix C.

Advanced Gating. Besides the input-choice and expert-choice gating algorithms evaluated above, we further consider soft MoEs (Puigcerver et al., 2024), which is a recent advanced gating algorithm. Specifically, it performs an implicit soft assignment by distributing K slots (i.e., convex combinations of all inputs) to each expert, rather than a hard assignment between inputs and experts as done by the conventional sparse and discrete gating networks. Since only K (e.g., 1 or 2) slots are distributed to each expert, it can save much computation. We train MVMoE on $n = 50$ by using node-level soft MoEs in the decoder, following training setups. We also show the result of employing heuristic (random) hierarchical gating in the decoder. However, their results are unsatisfactory as shown in Fig. 6(d).

Training Efficiency. Fig. 6(e) shows the training time of employing each gating algorithm in the decoder, combining with their results reported in Fig. 6(d), demonstrating the efficacy of the proposed hierarchical gating in reducing the training overhead with only minor losses in performance.

Benchmark Performance. We further evaluate the out-of-distribution (OOD) generalization performance of all neural solvers on CVRPLIB benchmark instances. Detailed results can be found in Tables 7 and 8 in Appendix C. Surprisingly, we observe that MVMoE/4E performs poorly on large-scale

instances (e.g., $n > 500$). It may be caused by the generalization issue of sparse MoEs when transferring to new distributions or domains, which is still an open question in the MoE literature (Fedus et al., 2022a). In contrast, MVMoE/4E-L mostly outperforms MVMoE/4E, demonstrating more favourable potential of the hierarchical gating in promoting the OOD generalization capability. It is worth noting that all neural solvers are only trained on the simple uniformly distributed instances with the size $n = 100$. Embracing more varied problem sizes (cross-size) and attribute distributions (cross-distribution) into the multi-task training (cross-problem) may further consolidate their performance.

Scalability. Given that supervised learning based approaches appear to be more scalable than RL-based approaches in the current literature, we try to build upon a more scalable method, i.e., LEHD (Luo et al., 2023). Concretely, we train a dense model LEHD and a light sparse model with 4 experts LEHD/4E-L on CVRP. The training setups are kept the same as Luo et al. (2023), except that we train all models for only 20 epochs for the training efficiency. We use the hierarchical MoE in each decoder layer of LEHD/4E-L. The results are shown in Table 8, which demonstrates the potential of MoE as a general idea that can further benefit recent scalable methods. Moreover, during the solution construction process, recent works (Drakulic et al., 2023; Gao et al., 2023) typically constrain the search space within a neighborhood of the currently selected node, which is shown to be effective in handling large-scale instances. Integrating MVMoE with these simple yet effective techniques may further improve large-scale performance.

6. Conclusion

Targeting a more generic and powerful neural solver for solving VRPs, we propose a multi-task vehicle routing solver with MoEs (MVMoE), which can solve a range of VRPs concurrently, even in a zero-shot manner. We provide valuable insights on how to apply MoEs in neural VRP solvers, and propose an effective and efficient hierarchical gating mechanism. Empirically, MVMoE demonstrates strong generalization capability on zero-shot, few-shot settings, and real-world benchmark. Despite this paper presents the first attempt towards a large VRP model, the scale of parameters is still far less than LLMs. We leave 1) the development of *scalable* MoE-based models in solving *large-scale VRPs*, 2) the venture of *generic* representations for different problems, 3) the exploration of *interpretability* of gating mechanisms (Nguyen et al., 2023; 2024), and 4) the investigation of *scaling laws* in MoEs (Krajewski et al., 2024) to the future work. We hope our work benefit the COP community in developing large optimization (or foundation) models[2].

[2] https://github.com/ai4co/awesome-fm4co

Acknowledgements

This research is supported by the National Research Foundation, Singapore under its AI Singapore Programme (AISG Award No: AISG3-RP-2022-031), the Singapore Ministry of Education (MOE) Academic Research Fund (AcRF) Tier 1 grant, the National Natural Science Foundation of China (Grant 62102228), and the Natural Science Foundation of Shandong Province (Grant ZR2021QF063). We would like to thank the anonymous reviewers and (S)ACs of ICML 2024 for their constructive comments and dedicated service to the community. Jianan Zhou would like to personally express deep gratitude to his grandfather, Jinlong Hu, for his meticulous care and love during last 26 years. Eternal easy rest in sweet slumber.

Impact Statement

This paper presents work whose goal is to advance the field of Machine Learning. There are many potential societal consequences of our work, none which we feel must be specifically highlighted here.

References

Bello, I., Pham, H., Le, Q. V., Norouzi, M., and Bengio, S. Neural combinatorial optimization with reinforcement learning. In *ICLR Workshop Track*, 2017.

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

Berto, F., Hua, C., Park, J., Kim, M., Kim, H., Son, J., Kim, H., Kim, J., and Park, J. RL4CO: a unified reinforcement learning for combinatorial optimization library. In *NeurIPS 2023 Workshop: New Frontiers in Graph Learning*, 2023.

Bi, J., Ma, Y., Wang, J., Cao, Z., Chen, J., Sun, Y., and Chee, Y. M. Learning generalizable models for vehicle routing problems via knowledge distillation. In *NeurIPS*, 2022.

Bogyrbayeva, A., Meraliyev, M., Mustakhov, T., and Dauletbayev, B. Machine learning to solve vehicle routing problems: A survey. *IEEE Transactions on Intelligent Transportation Systems*, 2024.

Boisvert, L., Verhaeghe, H., and Cappart, Q. Towards a generic representation of cominatorial problems for learning-based approaches. *arXiv preprint arXiv:2403.06026*, 2024.

Cattaruzza, D., Absi, N., Feillet, D., and González-Feliu, J. Vehicle routing problems for city logistics. *EURO Journal on Transportation and Logistics*, 6(1):51–79, 2017.

Chalumeau, F., Surana, S., Bonnet, C., Grinsztajn, N., Pretorius, A., Laterre, A., and Barrett, T. D. Combinatorial optimization with policy adaptation using latent space search. In *NeurIPS*, 2023.

Chen, J., Zhang, Z., Cao, Z., Wu, Y., Ma, Y., Ye, T., and Wang, J. Neural multi-objective combinatorial optimization with diversity enhancement. In *NeurIPS*, 2023.

Chen, X. and Tian, Y. Learning to perform local rewriting for combinatorial optimization. In *NeurIPS*, volume 32, 2019.

Chen, Z., Deng, Y., Wu, Y., Gu, Q., and Li, Y. Towards understanding the mixture-of-experts layer in deep learning. In *NeurIPS*, volume 35, pp. 23049–23062, 2022.

Croes, G. A. A method for solving traveling-salesman problems. *Operations research*, 6(6):791–812, 1958.

d O Costa, P. R., Rhuggenaath, J., Zhang, Y., and Akcay, A. Learning 2-opt heuristics for the traveling salesman problem via deep reinforcement learning. In *Asian Conference on Machine Learning*, pp. 465–480. PMLR, 2020.

Drakulic, D., Michel, S., Mai, F., Sors, A., and Andreoli, J.-M. BQ-NCO: Bisimulation quotienting for generalizable neural combinatorial optimization. In *NeurIPS*, 2023.

Eigen, D., Ranzato, M., and Sutskever, I. Learning factored representations in a deep mixture of experts. *arXiv preprint arXiv:1312.4314*, 2013.

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

Fedus, W., Zoph, B., and Shazeer, N. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *The Journal of Machine Learning Research*, 23(1):5232–5270, 2022b.

Floridi, L. and Chiriatti, M. Gpt-3: Its nature, scope, limits, and consequences. *Minds and Machines*, 30:681–694, 2020.

Fu, Z.-H., Qiu, K.-B., and Zha, H. Generalize a small pre-trained model to arbitrarily large tsp instances. In *AAAI*, volume 35, pp. 7474–7482, 2021.

Furnon, V. and Perron, L. Or-tools routing library, 2023. URL https://developers.google.com/optimization/routing.

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

Geisler, S., Sommer, J., Schuchardt, J., Bojchevski, A., and Günnemann, S. Generalization of neural combinatorial solvers through the lens of adversarial robustness. In *ICLR*, 2022.

Grinsztajn, N., Furelos-Blanco, D., Surana, S., Bonnet, C., and Barrett, T. D. Winner takes it all: Training performant RL populations for combinatorial optimization. In *NeurIPS*, 2023.

Helsgaun, K. An extension of the lin-kernighan-helsgaun tsp solver for constrained traveling salesman and vehicle routing problems. *Roskilde: Roskilde University*, pp. 24–50, 2017.

Hottung, A. and Tierney, K. Neural large neighborhood search for the capacitated vehicle routing problem. In *European Conference on Artificial Intelligence*, pp. 443–450, 2020.

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

Hou, Q., Yang, J., Su, Y., Wang, X., and Deng, Y. Generalize learned heuristics to solve large-scale vehicle routing problems in real-time. In *ICLR*, 2023.

Hu, E. J., yelong shen, Wallis, P., Allen-Zhu, Z., Li, Y., Wang, S., Wang, L., and Chen, W. LoRA: Low-rank adaptation of large language models. In *ICLR*, 2022.

Hudson, B., Li, Q., Malencia, M., and Prorok, A. Graph neural network guided local search for the traveling salesperson problem. In *ICLR*, 2022.

Ismail, A. A., Arik, S. O., Yoon, J., Taly, A., Feizi, S., and Pfister, T. Interpretable mixture of experts. *Transactions on Machine Learning Research*, 2023. ISSN 2835-8856.

Jacobs, R. A., Jordan, M. I., Nowlan, S. J., and Hinton, G. E. Adaptive mixtures of local experts. *Neural computation*, 3(1):79–87, 1991.

Jiang, Y., Cao, Z., Wu, Y., Song, W., and Zhang, J. Ensemble-based deep reinforcement learning for vehicle routing problems under distribution shift. In *NeurIPS*, 2023.

Jordan, M. I. and Jacobs, R. A. Hierarchical mixtures of experts and the em algorithm. *Neural computation*, 6(2): 181–214, 1994.

Joshi, C. K., Laurent, T., and Bresson, X. An efficient graph convolutional network technique for the travelling salesman problem. *arXiv preprint arXiv:1906.01227*, 2019.

Joshi, C. K., Cappart, Q., Rousseau, L.-M., and Laurent, T. Learning tsp requires rethinking generalization. In *International Conference on Principles and Practice of Constraint Programming*, 2021.

Kaplan, J., McCandlish, S., Henighan, T., Brown, T. B., Chess, B., Child, R., Gray, S., Radford, A., Wu, J., and Amodei, D. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*, 2020.

Kim, M., Park, J., and Park, J. Sym-NCO: Leveraging symmetricity for neural combinatorial optimization. In *NeurIPS*, 2022.

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

Kool, W., van Hoof, H., and Welling, M. Attention, learn to solve routing problems! In *ICLR*, 2018.

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

Krajewski, J., Ludziejewski, J., Adamczewski, K., Pióro, M., Krutul, M., Antoniak, S., Ciebiera, K., Król, K., Odrzygóźdź, T., Sankowski, P., et al. Scaling laws for fine-grained mixture of experts. *arXiv preprint arXiv:2402.07871*, 2024.

Kwon, Y.-D., Choo, J., Kim, B., Yoon, I., Gwon, Y., and Min, S. POMO: Policy optimization with multiple optima for reinforcement learning. In *NeurIPS*, volume 33, pp. 21188–21198, 2020.

Kwon, Y.-D., Choo, J., Yoon, I., Park, M., Park, D., and Gwon, Y. Matrix encoding networks for neural combinatorial optimization. In *NeurIPS*, volume 34, pp. 5138–5149, 2021.

Lepikhin, D., Lee, H., Xu, Y., Chen, D., Firat, O., Huang, Y., Krikun, M., Shazeer, N., and Chen, Z. Gshard: Scaling giant models with conditional computation and automatic sharding. *arXiv preprint arXiv:2006.16668*, 2020.

Lewis, M., Bhosale, S., Dettmers, T., Goyal, N., and Zettlemoyer, L. Base layers: Simplifying training of large, sparse models. In *ICML*, pp. 6265–6274. PMLR, 2021.

Li, J., Ma, Y., Gao, R., Cao, Z., Lim, A., Song, W., and Zhang, J. Deep reinforcement learning for solving the heterogeneous capacitated vehicle routing problem. *IEEE Transactions on Cybernetics*, 52(12):13572–13585, 2021a.

Li, S., Yan, Z., and Wu, C. Learning to delegate for large-scale vehicle routing. In *NeurIPS*, volume 34, pp. 26198–26211, 2021b.

Li, X. L. and Liang, P. Prefix-tuning: Optimizing continuous prompts for generation. In *ACL*, pp. 4582–4597, 2021.

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

Liu, F., Lin, X., Zhang, Q., Tong, X., and Yuan, M. Multi-task learning for routing problem with cross-problem zero-shot generalization. *arXiv preprint arXiv:2402.16891*, 2024.

Lu, H., Zhang, X., and Yang, S. A learning-based iterative method for solving vehicle routing problems. In *ICLR*, 2020.

Luo, F., Lin, X., Liu, F., Zhang, Q., and Wang, Z. Neural combinatorial optimization with heavy decoder: Toward large scale generalization. In *NeurIPS*, 2023.

Ma, Y., Cao, Z., and Chee, Y. M. Learning to search feasible and infeasible regions of routing problems with flexible neural k-opt. In *NeurIPS*, 2023.

Manchanda, S., Michel, S., Drakulic, D., and Andreoli, J.-M. On the generalization of neural combinatorial optimization heuristics. In *ECML PKDD*, 2022.

Min, Y., Bai, Y., and Gomes, C. P. Unsupervised learning for solving the travelling salesman problem. In *NeurIPS*, 2023.

Nazari, M., Oroojlooy, A., Snyder, L., and Takác, M. Reinforcement learning for solving the vehicle routing problem. In *NeurIPS*, volume 31, 2018.

Nguyen, H., Nguyen, T., and Ho, N. Demystifying softmax gating function in gaussian mixture of experts. In *NeurIPS*, 2023.

Nguyen, H., Akbarian, P., Yan, F., and Ho, N. Statistical perspective of top-k sparse softmax gating mixture of experts. In *ICLR*, 2024.

Puigcerver, J., Riquelme, C., Mustafa, B., and Houlsby, N. From sparse to soft mixtures of experts. In *ICLR*, 2024.

Qiu, R., Sun, Z., and Yang, Y. DIMES: A differentiable meta solver for combinatorial optimization problems. In *NeurIPS*, 2022.

Riquelme, C., Puigcerver, J., Mustafa, B., Neumann, M., Jenatton, R., Susano Pinto, A., Keysers, D., and Houlsby, N. Scaling vision with sparse mixture of experts. In *NeurIPS*, volume 34, pp. 8583–8595, 2021.

Roller, S., Sukhbaatar, S., Weston, J., et al. Hash layers for large sparse models. In *NeurIPS*, volume 34, pp. 17555–17566, 2021.

Ruis, F., Burghouts, G., and Bucur, D. Independent prototype propagation for zero-shot compositionality. *NeurIPS*, 34:10641–10653, 2021.

Shaw, P. Using constraint programming and local search methods to solve vehicle routing problems. In *International conference on principles and practice of constraint programming*, pp. 417–431. Springer, 1998.

Shazeer, N., Mirhoseini, A., Maziarz, K., Davis, A., Le, Q., Hinton, G., and Dean, J. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. In *ICLR*, 2017.

Solomon, M. M. Algorithms for the vehicle routing and scheduling problems with time window constraints. *Operations research*, 35(2):254–265, 1987.

Son, J., Kim, M., Kim, H., and Park, J. Meta-SAGE: Scale meta-learning scheduled adaptation with guided exploration for mitigating scale shift on combinatorial optimization. In *ICML*, 2023.

Sun, Z. and Yang, Y. DIFUSCO: Graph-based diffusion solvers for combinatorial optimization. In *NeurIPS*, 2023.

Touvron, H., Martin, L., Stone, K., Albert, P., Almahairi, A., Babaei, Y., Bashlykov, N., Batra, S., Bhargava, P., Bhosale, S., et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023.

Uchoa, E., Pecin, D., Pessoa, A., Poggi, M., Vidal, T., and Subramanian, A. New benchmark instances for the capacitated vehicle routing problem. *European Journal of Operational Research*, 257(3):845–858, 2017.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. Attention is all you need. In *NeurIPS*, volume 30, 2017.

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

Vinyals, O., Fortunato, M., and Jaitly, N. Pointer networks. In *NeurIPS*, volume 28, 2015.

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

Wang, C., Yu, Z., McAleer, S., Yu, T., and Yang, Y. ASP: Learn a universal neural solver! *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2024.

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

Wu, Y., Song, W., Cao, Z., Zhang, J., and Lim, A. Learning improvement heuristics for solving routing problems. *IEEE transactions on neural networks and learning systems*, 33(9):5057–5069, 2021.

Wu, Y., Zhou, J., Xia, Y., Zhang, X., Cao, Z., and Zhang, J. Neural airport ground handling. *IEEE Transactions on Intelligent Transportation Systems*, 2023.

Xin, L., Song, W., Cao, Z., and Zhang, J. NeuroLKH: Combining deep learning model with lin-kernighan-helsgaun heuristic for solving the traveling salesman problem. In *NeurIPS*, volume 34, pp. 7472–7483, 2021.

Xue, F., Zheng, Z., Fu, Y., Ni, J., Zheng, Z., Zhou, W., and You, Y. OpenMoE: An early effort on open mixture-of-experts language models. *arXiv preprint arXiv:2402.01739*, 2024.

Ye, H., Wang, J., Cao, Z., Liang, H., and Li, Y. DeepACO: Neural-enhanced ant systems for combinatorial optimization. In *NeurIPS*, 2023.

Yuksel, S. E., Wilson, J. N., and Gader, P. D. Twenty years of mixture of experts. *IEEE transactions on neural networks and learning systems*, 23(8):1177–1193, 2012.

Zhang, C., Wu, Y., Ma, Y., Song, W., Le, Z., Cao, Z., and Zhang, J. A review on learning to solve combinatorial optimisation problems in manufacturing. *IET Collaborative Intelligent Manufacturing*, 5(1):e12072, 2023.

Zhang, Z., Zhang, Z., Wang, X., and Zhu, W. Learning to solve travelling salesman problem with hardness-adaptive curriculum. In *AAAI*, 2022.

Zhou, J., Wu, Y., Cao, Z., Song, W., Zhang, J., and Chen, Z. Learning large neighborhood search for vehicle routing in airport ground handling. *IEEE Transactions on Knowledge and Data Engineering*, 2023a.

Zhou, J., Wu, Y., Song, W., Cao, Z., and Zhang, J. Towards omni-generalizable neural methods for vehicle routing problems. In *ICML*, pp. 42769–42789. PMLR, 2023b.

Zhou, Y., Lei, T., Liu, H., Du, N., Huang, Y., Zhao, V., Dai, A. M., Le, Q. V., Laudon, J., et al. Mixture-of-experts with expert choice routing. In *NeurIPS*, volume 35, pp. 7103–7114, 2022.

Zuo, S., Liu, X., Jiao, J., Kim, Y. J., Hassan, H., Zhang, R., Gao, J., and Zhao, T. Taming sparsely activated transformer with stochastic experts. In *ICLR*, 2022.

A. Setups of VRP Variants

We mainly consider five constraints as presented in Section 3. Our base VRP variant is CVRP, from which we derive 16 VRP variants by adding additional constraints (as listed in Table 3). The node coordinates are randomly sampled from the unit square $U(0, 1)$. Below, we provide a comprehensive description of the data generation process for each constraint.

Capacity (C). We follow the common settings in literature (Kool et al., 2018; Kwon et al., 2020). The node demand δ_i is randomly sampled from a discrete uniform distribution $\{1, 2, \ldots, 9\}$. The vehicle capacity Q is set to 40 and 50 for $n = 50$ and $n = 100$, respectively. Before feeding into the network, the node demand is further normalized by $\delta'_i = \delta_i/Q$. During the decoding process, we mask all nodes whose demands are larger than the remaining capacity of the current vehicle.

Open Route (O). The open route constraint does not involve extra data generation. During the decoding process, we set the open route indicator $o_t = 1$ in the dynamic feature set $\mathcal{D}_t$. As mentioned in Section 3, the integration of the open route constraint into others is not a simple addition. Concretely, 1) *O w. L:* Since the vehicle does not return to the depot node, during the decoding process, we only need to mask the node v_j satisfying $l_t + d_{ij} > \mathbb{L}$, where l_t is the length of the current route; d_{ij} is the distance between the current node v_i and unmasked node v_j; $\mathbb{L}$ is the predefined threshold of the route length. Without the open route constraint, we should mask the node v_j satisfying $l_t + d_{ij} + d_{j0} > \mathbb{L}$; 2) *O w. TW:* During the decoding process, we mask the node v_j satisfying $t_t + d_{ij} > l_j$, where t_t is the current time (i.e., the completed time of serving the current node v_i); d_{ij} is the distance (i.e., travel time since the vehicle speed is 1) between the current node v_i and unmasked node v_j; l_j is the end time window of v_j. Without the open route constraint, we need to *further* mask the node v_j satisfying $\max(t_t + d_{ij}, e_j) + s_j + d_{j0} > l_0$, where e_j is the start time window of v_j; s_j is the service time; l_0 is the end time window of the depot node v_0. 3) *O w. C or B:* Since the demand of the depot node is 0, it does not affect the constraint satisfaction during decoding, except that the vehicle does not need to return to the depot node at the end of each route.

Backhaul (B). Similar to CVRP, we first generate node demands by randomly sampling from a discrete uniform distribution $\{1, 2, \ldots, 9\}$. Then, following Liu et al. (2024), we randomly select 20% of customers as backhauls. In this paper, we consider routes that involve a mixture of linehauls and backhauls, without the presence of strict precedence constraints between them. For neural VRP solvers, to ensure the feasibility of constructed solutions, the initial customer node chosen for each vehicle should be a linehaul, with an exception being that all remaining (unvisited) nodes are backhauls.

Duration Limit (L). Following Liu et al. (2024), we set the duration limit (i.e., the maximum length of each vehicle route) as $\mathbb{L} = 3$, which ensures the neural solver can find feasible solutions within the unit square space $U(0, 1)$.

Time Window (TW). Following Li et al. (2021b) whose data generation is similar to the procedure as in Solomon (Solomon, 1987), we set the time window for the depot node v_0 as $[e_0, l_0] = [0, 3]$, and the service time at each customer node v_i as $s_i = 0.2$. The depot node does not have service time. The time window for the customer node v_i is then obtained by 1) sampling the time window center $\gamma_i \sim U(e_0 + d_{0i}, l_0 - d_{i0} - s_i)$, where $d_{0i} = d_{i0}$ denotes the distance or travel time between v_0 and v_i; 2) sampling the time window half-width h_i uniformly at random from $[s_i/2, l_0/3] = [0.1, 1]$; 3) setting the time window $[e_i, l_i]$ as $[\max(e_0, \gamma_i - h_i), \min(l_0, \gamma_i + h_i)]$.

Table 3. 16 VRP variants with five constraints.

	Capacity (C)	Open Route (O)	Backhaul (B)	Duration Limit (L)	Time Window (TW)
CVRP	✓				
OVRP	✓	✓			
VRPB	✓		✓		
VRPL	✓			✓	
VRPTW	✓				✓
OVRPTW	✓	✓			✓
OVRPB	✓	✓	✓		
OVRPL	✓	✓		✓	
VRPBL	✓		✓	✓	
VRPBTW	✓		✓		✓
VRPLTW	✓			✓	✓
OVRPBL	✓	✓	✓	✓	
OVRPBTW	✓	✓	✓		✓
OVRPLTW	✓	✓		✓	✓
VRPBLTW	✓		✓	✓	✓
OVRPBLTW	✓	✓	✓	✓	✓

B. Multi-Task VRP Solver with MoEs

We now present more details of the multi-task VRP solver with MoEs (MVMoE). We consider POMO (Kwon et al., 2020) as the backbone model, which is one of the most prevalent autoregressive construction-based neural solvers in VRPs.

B.1. Encoder

Given a VRP instance with n nodes, the static features of the $i_{\text{th}}(i \in [0, n])$ node are $\mathcal{S}_i = \{y_i, \delta_i, e_i, l_i\}$, where we use y_i, δ_i, e_i, l_i to denote the coordinate, node demand, start and end time of the time window, respectively. A linear layer is first used to project the raw features to d-dimensional (i.e., $d = 128$) node embeddings $h_i^0 = \text{Linear}([y_i, \delta_i, e_i, l_i])$. Then, a stack of $N = 6$ encoder layers is applied to extract the refined node embeddings h_i^N. Specifically, as shown in Fig. 2, each encoder layer consists of a multi-head attention (MHA) layer and a feed forward network (FFN). Following the Transformer architecture (Vaswani et al., 2017), their outputs are further processed by a skip-connection and instance normalization (IN),

$$\tilde{h}_i = \text{IN}(h_i^\ell + \text{MHA}(h_i^\ell)), \tag{4}$$

$$h_i^{\ell+1} = \text{IN}(\tilde{h}_i + \text{FFN}(\tilde{h}_i)). \tag{5}$$

Below, we detail the MHA and FFN layers. 1) *MHA:* the MHA layer in the encoder employs a multi-head self-attention mechanism (i.e., with $A = 8$ heads) to compute attention weights between each two nodes. Formally, we define dimensions d_k and d_v (i.e., $d_k = d_v = d/A = 16$), and compute query $q_i^{\ell,a} \in \mathbb{R}^{d_k}$, key $k_i^{\ell,a} \in \mathbb{R}^{d_k}$, and value $v_i^{\ell,a} \in \mathbb{R}^{d_v}$ as follows,

$$q_i^{\ell,a} = W_Q^{\ell,a} h_i^\ell, \quad k_i^{\ell,a} = W_K^{\ell,a} h_i^\ell, \quad v_i^{\ell,a} = W_V^{\ell,a} h_i^\ell, \tag{6}$$

where $W_Q^{\ell,a}, W_K^{\ell,a}, W_V^{\ell,a}$ are the parameter matrices of the $a_{\text{th}}(a \in [1, A])$ attention head in the $\ell_{\text{th}}(\ell \in [0, N-1])$ encoder layer. Then, the attention weight $u_{ij}^{\ell,a}$ is computed using the Softmax function to represent the correlation between the i_{th} node and the j_{th} node as follows,

$$u_{ij}^{\ell,a} = \text{Softmax}\left(\frac{(q_i^{\ell,a})^T k_j^{\ell,a}}{\sqrt{d_k}}\right). \tag{7}$$

By weighted message passing among nodes and aggregating outputs from multiple heads, the final MHA output for the i_{th} node at the ℓ_{th} encoder layer is obtained as follows,

$$h_i^{\ell,a} = \sum_{j=0}^{n} u_{ij}^{\ell,a} v_j^{\ell,a}, \tag{8}$$

$$\text{MHA}(h_i^\ell) = [h_i^{\ell,1}, h_i^{\ell,2}, \ldots, h_i^{\ell,A}] W_O^\ell, \tag{9}$$

where W_O^ℓ is the learnable parameter and $[,]$ is the concatenate operator. 2) *FFN:* the FFN layer computes node-wise projections using a hidden sublayer with the dimension $d_f = 512$ and a ReLU activation as follows,

$$\text{FFN}(\tilde{h}_i) = W_F^{\ell,1} \cdot \text{ReLU}(W_F^{\ell,0} \tilde{h}_i + b_F^{\ell,0}) + b_F^{\ell,1}, \tag{10}$$

where $W_F^{\ell,0}, W_F^{\ell,1}, b_F^{\ell,0}, b_F^{\ell,1}$ are learnable parameters of the FFN in the ℓ_{th} layer. In this paper, we replace the FFN in each encoder layer with an MoE layer, which consists of m FFNs $\{\text{FFN}_1, \ldots, \text{FFN}_m\}$ and a gating network G, such that Eq. (5) is rewritten as,

$$h_i^{\ell+1} = \text{IN}(\tilde{h}_i + \sum_{j=1}^{m} G(\tilde{h}_i)_j \text{FFN}_j(\tilde{h}_i)). \tag{11}$$

B.2. Decoder

The decoder takes all node embeddings $\{h_i^N\}_{i=0}^n$ and a context embedding h_c as inputs. At the t_{th} decoding step, the context embedding $h_c = [h_{\tau_{t-1}}^N, \mathcal{D}_t] \in \mathbb{R}^{d+4}$ includes the embedding of the last selected node $h_{\tau_{t-1}}^N$ (initialized as the depot node h_0^N) and dynamic features $\mathcal{D}_t = \{c_t, t_t, l_t, o_t\}$, where c_t, t_t, l_t, o_t represent the remaining capacity of the vehicle, the current time, the length of the current partial route, and the presence indicator of the open route, respectively. Next, we compute an updated context embedding h_c' through a MHA layer. Note that the MHA layer in the decoder does not employ

the self-attention. Concretely, the context embedding h_c is used for computing query, while the node embeddings h_i^N are used for computing the key and value,

$$q_c^{D,a} = W_Q^{D,a} h_c, \quad k_i^{D,a} = W_K^{D,a} h_i^N, \quad v_i^{D,a} = W_V^{D,a} h_i^N, \tag{12}$$

where $W_Q^{D,a}, W_K^{D,a}, W_V^{D,a}$ are the parameter matrices of the a_{th} attention head in the decoder. Then, we follow Eqs. (7)-(9) to obtain the updated context embedding h'_c. Note that invalid nodes (i.e., appending them to the current partial solution may violate the problem-specific feasibility constraints) are masked such that their attention weights are set to 0 in Eq. (7). Finally, the probabilities p_i of valid nodes to be selected is calculated by a single-head attention as follows,

$$p_i = \text{Softmax}\left(C \cdot \tanh\left(\frac{(h'_c)^T h_i^N}{\sqrt{d_k}}\right)\right), \tag{13}$$

where C is typically set to 10 to clip the logits so as to benefit the policy exploration (Bello et al., 2017; Kool et al., 2018). In this paper, we replace the final linear layer of MHA (i.e., W_O^ℓ in Eq. (9)) in the decoder with an MoE layer, which consists of m linear layers with parameters $\{W_O^{\ell,1}, \ldots, W_O^{\ell,m}\}$ and a gating network G. In this case, Eq. (9) is rewritten as:

$$\text{MHA}(h_i^\ell) = \sum_{j=1}^{m} G([h_i^{\ell,1}, h_i^{\ell,2}, \ldots, h_i^{\ell,A}])_j [h_i^{\ell,1}, h_i^{\ell,2}, \ldots, h_i^{\ell,A}] W_O^{\ell,j}. \tag{14}$$

B.3. Gating Algorithms

We present more details of the popular gating algorithms (Shazeer et al., 2017; Zhou et al., 2022) in MoEs. Without loss of generality, we take the node-level gating as an example. Suppose that we have m experts in an MoE layer, and its input has a shape of $X \in \mathbb{R}^{I \times d}$, where I is the total number of nodes in a batch of inputs. The gating network G takes X as inputs and outputs the score matrix $H = (X \cdot W_G) \in \mathbb{R}^{I \times m}$, where W_G are trainable parameters of G.

Input-Choice Gating. Each node selects Top$K$ experts based on the score matrix H in the input-choice gating. The Softmax function is applied along the last dimension of H to obtain the probability matrix P, where $P[i, j]$ denotes the probability of the j_{th} expert being selected by the i_{th} node. However, this method does not inherently ensure load balancing. For example, an expert may receive significantly more nodes than others, resulting in a dominant expert while leaving others underfitting. Following Shazeer et al. (2017), we use the noisy Top$K$ gating with the importance & load loss to enforce a similar number of nodes received by each expert. Concretely, a learnable Gaussian noise is added to H before taking the Softmax,

$$H' = H + \text{StandardNormal}() \cdot \text{Softplus}(X \cdot W_{noise}), \tag{15}$$

$$P = \text{Softmax}\left(\text{Top}K(H')\right), \tag{16}$$

where $W_{noise} \in \mathbb{R}^{d \times m}$ is a learnable matrix. An example of Python code of Eq. (15) is shown below.

```
# An example of Python Code of Eq. (15)
H = x @ w_gate  # w_gate is the parameter of the gating network G
noise_stddev = torch.nn.functional.softplus(x @ w_noise) + 1e-2
noisy_scores = H + (torch.randn_like(H) * noise_stddev)  # noisy_scores: H'
```

For load-balancing purposes, the goal is to ensure that each expert receives a roughly equal number of nodes. However, the number of received nodes is discrete, and hence cannot be used as the loss information to update the network through backpropagation. Shazeer et al. (2017) proposes to use a smooth estimator to approximate the number of nodes assigned to each expert. The above noise term helps with load balancing by enabling the gradient backpropagation through the smooth estimator. Then, an importance & load loss is used as the auxiliary loss $\mathcal{L}_b$ to enforce load balancing as follows,

$$Importance(X) = \sum_{x \in X} G(x), \tag{17}$$

$$Load(X)_j = \sum_{x \in X} \Phi\left(\frac{(x \cdot W_G)_j - \varphi(H'_x, k, j)}{\text{Softplus}((x \cdot W_{noise})_j)}\right), \tag{18}$$

$$\mathcal{L}_b = CV(Importance(X))^2 + CV(Load(X))^2, \tag{19}$$

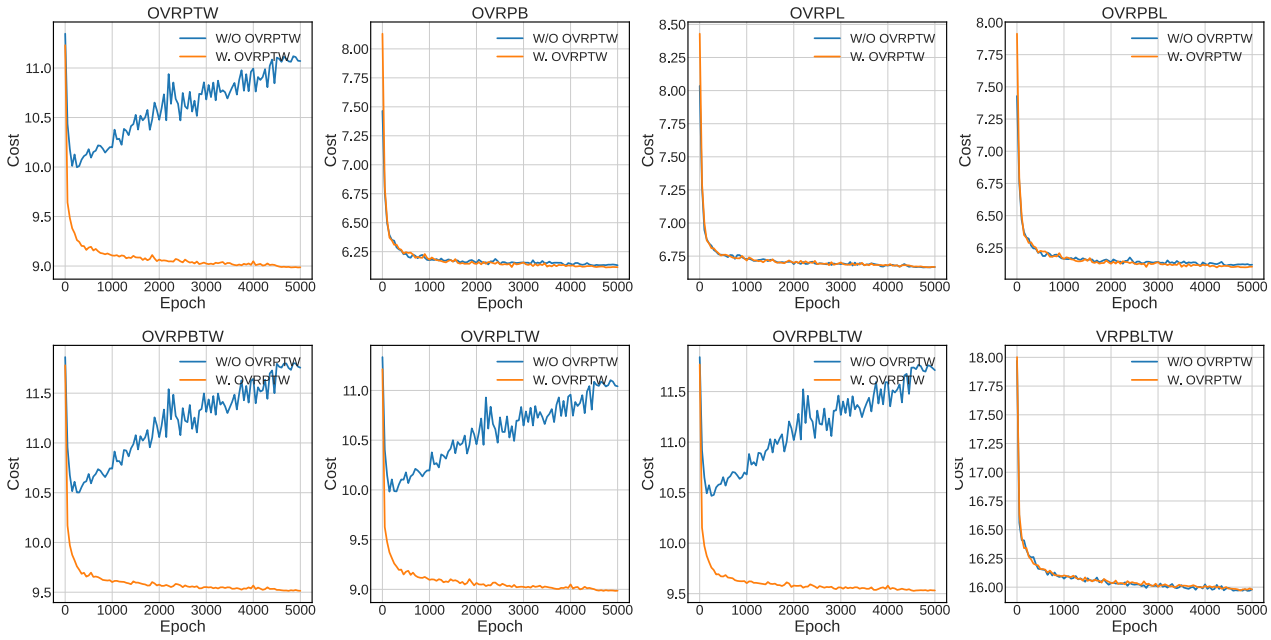

Figure 7. The validation curves of POMO-MTL trained w/o OVRPTW and w. OVRPTW on $n = 50$.

where $Importance(X)$ and $Load(X)$ are vectors with the shape of $\mathbb{R}^m$; Φ denotes the cumulative distribution function of the standard normal distribution; $\varphi(a, b, c)$ denotes the b_{th} largest component of a excluding component c; H'_x denotes the vector corresponding to the node x in H'; CV denotes the coefficient of variation. Specifically, Eq. (17) encourages all experts to have equal importance (i.e., the batchwise sum of gate values), and Eq. (18) ensures balanced loads among experts. We set the weight of the auxiliary loss (i.e., α in Eq. (3)) as 0.01. We refer more details to Shazeer et al. (2017).

Expert-Choice Gating. In contrast to the input-choice gating, each expert independently selects Top$K$ nodes based on the score matrix H in the expert-choice gating. The Softmax function is applied along the first dimension of H to obtain P, where $P[i, j]$ denotes the probability of the i_{th} node being selected by the j_{th} expert. It explicitly guarantees load balancing and enables a flexible allocation of computation (e.g., a node can be distributed to various number of experts). In general, K is set to $\frac{I \times \beta}{m}$, where β is the capacity factor, representing on average how many experts are utilized by a node. We set $\beta = 2$ in our experiments. Note that we do not limit the maximum number of experts for each node, since it needs to solve an extra linear programming problem (Zhou et al., 2022) and hence increases the computational complexity.

Random Gating. It is a simple heuristic that randomly route inputs to experts. For example, in the problem-level gating, we randomly route a batch of inputs to K experts, similar to Zuo et al. (2022).

B.4. Gating Levels

In addition to the node-level gating presented in Section 4, we further investigate another two gating levels in VRPs. The detailed empirical results and analyses can be found in Section 5.

Instance-Level Gating. The instance-level gating routes inputs at the granularity of instances. Given a batch of inputs $X \in \mathbb{R}^{B \times n \times d}$, all n nodes belong to the same instance are routed to the same expert(s). Before forward passing X to the gating network, we apply the mean pooling operator along the second dimension of X, such that the score matrix has the shape of $H \in \mathbb{R}^{B \times m}$. Similar to the node-level gating, the input-choice and expert-choice gating algorithms are applicable to the instance-level gating as well. In contrast to the node-level gating, x_i denotes the i_{th} instance rather than a single node in Fig. 3. Typically, the instance-level gating can save the gating computation since $B \ll I$ as the problem scale n increases. However, its empirical performance may be slightly inferior to that of the node-level gating based on our empirical results.

Problem-Level Gating. The problem-level gating views a batch of instances from the same VRP variant independently, and hence we route a batch of inputs X to the same expert(s). Different from the node-level or instance-level gating, the

input-choice and expert-choice gating algorithms are not applicable to the problem-level gating. Potential gating algorithms include randomly routing X to K (e.g., 1 or 2) experts (Zuo et al., 2022) or adapting the input-choice gating without extra auxiliary losses. Although the problem-level gating is simple and efficient, it cannot guarantee all experts being sufficiently trained, since only K experts are optimized in each training step.

C. Experiments

C.1. Training Problem Set

As mentioned in Section 5, our training problem set includes CVRP, OVRP, VRPB, VRPL, VRPTW and OVRPTW. In contrast to our setting, the training problem set in Liu et al. (2024) does not include OVRPTW. We would like to explain this difference from two perspectives: 1) *different generation of time window:* we follow Li et al. (2021b) whose data generation is similar to the procedure as in Solomon (Solomon, 1987), while Liu et al. (2024) follows an artificial generation process. Since Solomon dataset is based on practical constraints and real-life data, we believe its generation of time window is more realistic and challenging; 2) *O meets with TW:* we empirically observe that the zero-shot generalization performance of POMO-MTL trained without OVRPTW becomes worse on some VRP variants, which have open route and time window constraints concurrently. Therefore, we further include OVRPTW into our training problem set to solve the challenge. The comprehensive validation curves are shown in Fig. 7. We assume this empirical observation is attributed to the different data generation of time window, since it is the only difference between our settings and Liu et al. (2024). Note that we retrain all neural methods following our training setups for a fair experimental comparison.

C.2. Effect of Constraints

We empirically find the time window constraint may have a bigger effect on the generalization performance. We show the result of POMO trained on each problem with $n = 100$ in Table 4, where we observe the average performance of POMO-VRPTW is the worst, demonstrating that the model is easier to overfit to the time window. Note that the data generation process may also affect the generalization. As shown in Appendix C.1, modifying the data generation of time window may significantly affect the zero-shot generalization performance of the trained model on specific VRP variants.

Table 4. The results of POMO trained on each problem with $n = 100$.

	CVRP	OVRP	VRPB	VRPL	VRPTW	OVRPTW	Average
POMO-CVRP	**1.488%**	20.124%	5.130%	0.105%	31.593%	42.452%	16.815%
POMO-OVRP	11.698%	**2.192%**	17.877%	10.152%	38.001%	27.734%	17.942%
POMO-VRPB	2.422%	20.902%	**0.995%**	1.050%	33.739%	42.849%	16.993%
POMO-VRPL	1.490%	19.606%	5.849%	**0.093%**	30.382%	41.164%	16.431%
POMO-VRPTW	43.850%	80.905%	79.753%	37.869%	**4.307%**	20.832%	44.586%
POMO-OVRPTW	33.839%	58.329%	56.245%	30.257%	23.518%	**2.467%**	34.109%

C.3. Learned Gating Policy

Hierarchical Gating. Here, we give a simple illustration of the learned policy of the first gating network G_1 in MVMoE/4E-L. In specific, in Fig. 8, we show the gating decision of G_1, and the percentage of decoding steps which route inputs to the dense or sparse layer, on all 16 VRP variants during inference. Since we use the problem-level gating in the first stage, it is expected that G_1 explores various gating policies for different VRPs, which can be justified by the statistics in Fig. 8, in order to obtain diverse solution construction policies. Note that the interpretability of the learned gating policy is still a challenging task in the literature of MoEs (Chen et al., 2022; Ismail et al., 2023). Since this is an early work of applying MoEs in VRPs, we concentrate on the empirical performance and leave their interpretability to the future work.

Expert Load. We expect the learned gating policy can ensure load balancing, such that each expert is sufficiently trained. Below, we take MVMoE/4E as an example, and show the load of each expert on 16 VRP variants during inference. Concretely, the load is calculated as the average percentage of nodes being routed to each expert. Based on the results in Fig. 9, we find that 1) the loads are well-balanced among 4 experts on 6 trained VRPs; 2) the learned gating policy can generalize to unseen 10 VRPs, with only slight unbalance (e.g., maximum 27% nodes are routed to one expert).

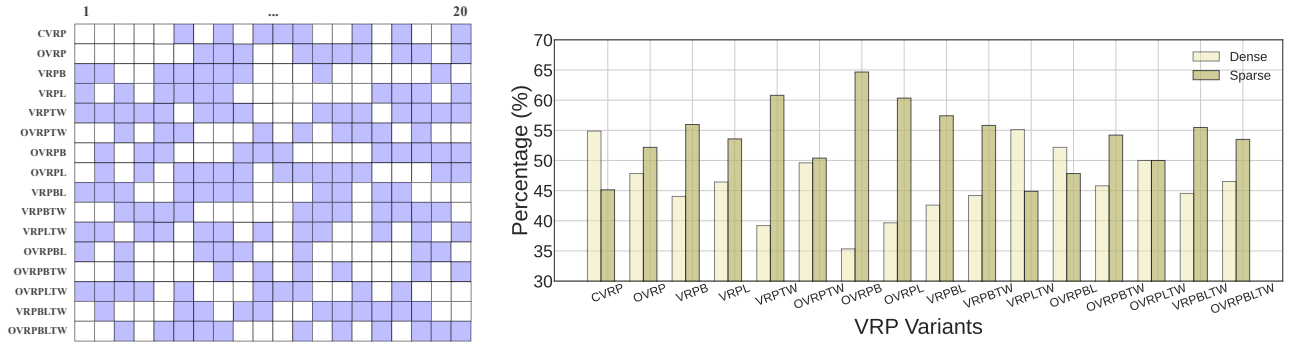

Figure 8. Left panel: An illustration of the first gating network's decision in MVMoE/4E-L. For simplicity, we only show the first 20 decoding steps. The colored square represents the selection of the sparse layer, while the blank square represents the selection of the dense layer. *Right panel:* The percentage of decoding steps which route inputs to the dense or sparse layer by the first gating network.

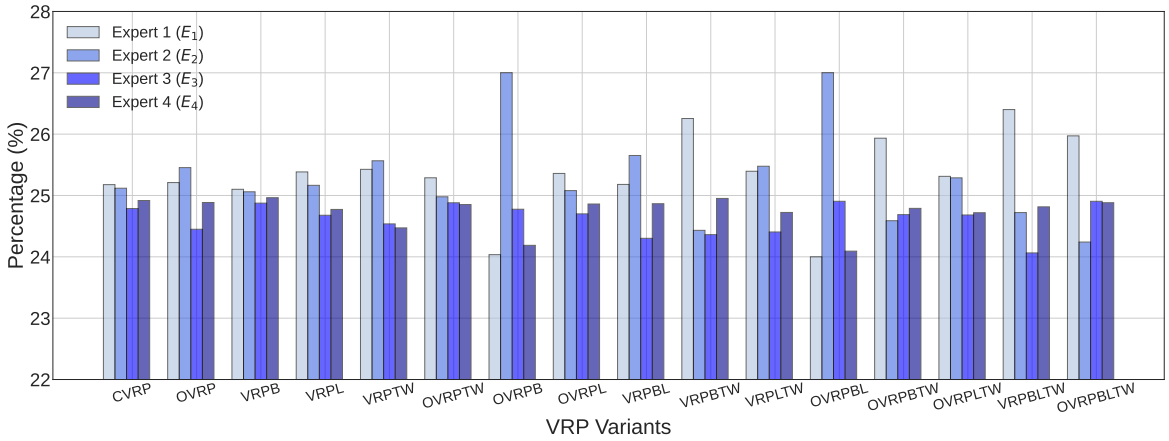

Figure 9. The average percentage of nodes being routed to each expert on 16 VRP Variants.

C.4. Sensitivity Analyses

In addition to the extensive studies on MoE settings (see Section 5.2), we further conduct sensitivity analyses on hyperparameters, including the auxiliary loss weight α, normalization layer, and K (representing how many experts are leveraged to process each input). For the training efficiency, we follow the training setups used in Section 5.2, where the number of training epochs is halved on $n = 50$. The detailed results can be found in Table 5, where we show the average performance of MVMoE/4E with different hyperparameters on the 6 trained VRPs and 10 unseen VRPs, respectively. Based on the empirical results, we find that further tuning hyperparameters (e.g., normalization layer) may boost the final performance. However, for a fair comparison with baselines, we follow their original setups (Kwon et al., 2020; Liu et al., 2024) by taking the first line as the default hyperparameter setting.

Table 5. Sensitivity Analyses on hyperparameters.

α	Normalization Layer	K	Trained VRPs	Unseen VRPs
0.01	Instance Norm	2	2.171%	6.153%
0.001	Instance Norm	2	2.156%	6.091%
0.1	Instance Norm	2	2.338%	6.438%
1.0	Instance Norm	2	2.308%	6.406%
0.01	Batch Norm	2	11.560%	18.821%
0.01	Layer Norm	2	2.048%	5.973%
0.01	None	2	**2.033%**	**5.842%**
0.01	Instance Norm	1	2.610%	6.751%
0.01	Instance Norm	3	2.126%	6.109%

C.5. Benchmark Performance

We evaluate all neural solvers on CVRPLIB benchmark dataset, including CVRP and VRPTW instances with various problem sizes and attribute distributions. We mainly consider the classic Set-X (Uchoa et al., 2017) and Set-Solomon (Solomon, 1987). Note that all neural solvers are only trained on the simple uniformly distributed instances with the size $n = 100$ (i.e., the customer nodes' locations follow the uniform distribution). The comprehensive results are shown in Tables 7 and 8, where we observe 1) the single task training method (i.e., POMO) may overfit to the uniformly distributed data, and hence its out-of-distribution (OOD) generalization performance is worst; 2) the OOD generalization capability of our sparse models is generally stronger than the dense model counterpart POMO-MTL, but the superiority tends to vanish on large-scale instances; 3) surprisingly, MVMoE/4E-L performs better than MVMoE/4E, demonstrating more favourable potential of the proposed hierarchical gating in promoting the OOD generalization capability of sparse MoE-based models.

D. Discussions

The motivation of multi-task VRP solver. 1) From the perspective of operation research (OR): We believe generality is a favorable objective in the community. Both exact solvers (e.g., CPLEX, Gurobi) and heuristic solvers (e.g., LKH3) can solve a wide range of problems, making them a popular choice for solving VRPs (or COPs). Despite the recent trend of developing neural solvers, they still need to be tailored for each specific problem, which may consume much computation resources. For example, if training a simple model for each problem, the total training cost would be 16 models x 3 days per model, while the training cost of a unified model is only 3 days. This resource consumption cannot be neglected from a global and practical view, since we often have not much computational resources and time in practice to train many individual (and large already) models. 2) From the perspective of ML: The recent success of LLMs has demonstrated the power of foundation models. By only using text (and vision) prompts, they can solve a wide range of tasks, including code generation, text summary, or even solving optimization problems. It is also a research trend to develop a foundation model that can unify multiple tasks in CV and NLP communities. With that said, developing a general neural solver (or foundation model) in combinatorial optimization is important, and we believe it should receive attention in the NCO community, and will be a trend in future research. Moreover, given a pretrained multi-task solver, we can also adapt it to the specific downstream task through in-context learning or efficient fine-tuning, making it an attractive choice versus training from scratch.

Why using the problem-level gating in the first stage of the hierarchical gating? We explain this question from two perspectives: 1) *generality:* we expect the proposed hierarchical gating can be applicable to any existing gating algorithms. If we use the node-level gating in the first stage, the instance-level gating algorithms will not be applicable afterwards since we cannot guarantee all nodes belong to the same instance are routed to the same (sparse or dense) layer. Similarly, if we use the instance-level gating in the first stage, the problem-level gating algorithms (e.g., THOR (Zuo et al., 2022)) will not be applicable in the second stage; 2) *efficiency:* it is possible to route a subset of inputs to the sparse layer, while distributing others to the dense layer. However, based on the Cannikin Law (i.e., Wooden Bucket Theory), the dense layer may need to wait the (relatively expensive) sparse layer before finishing the forward pass of the entire layer. The distribution of inputs at the first stage may also require extra computation if leveraging other gating levels. Therefore, for the generality and efficiency of the proposed hierarchical gating, we use the problem-level gating in the first stage.

The effect of the hierarchical gating mechanism. The proposed hierarchical gating mechanism is anticipated to reduce the training complexity, and significantly improve the out-of-distribution (OOD) generalization as we empirically observed. On the one hand, it is intuitive that the hierarchical gating can improve training efficiency. The solution construction (i.e., decoding) process is autoregressive in VRPs. If using the base gating, in each decoding step, the model needs to (a) handle the problem-specific feasibility constraints, (b) forward pass the inputs to the gating network, (c) distribute and forward pass each input to the selected k experts. It will induce more computation overheads as the problem size scales up. Therefore, the hierarchical gating is proposed to adaptively replace the time-consuming computation of (c) with a light forward pass to a dense layer in some decoding steps. As shown in Fig. 6(e), the training time can be reduced by around 23%. On the other hand, as shown in Tables 1 and 7, though MVMoE/4E-L (i.e., the model with the hierarchical gating) is slightly worse than MVMoE/4E (i.e., the model with the base gating) on the synthetic datasets, MVMoE/4E-L has a much stronger OOD generalization capability than MVMoE/4E on the real-world CVRPLIB dataset. We conduct further experiments and observe that the decision of the gating network in MVMoE/4E is much worse when facing OOD data. Specifically, we evaluate MVMoE/4E on uniformly distributed data with the size $n = 500$, while the training instance only has a size of $n = 100$. In some decoding steps, the gating network of MVMoE/4E distributes all nodes to 3 experts, while assigning few nodes to the remaining expert (e.g., the load of each expert is around [24.1%, 49.9%, 0.1%, 25.9%]). This extreme load

unbalancing may be one of the main reasons for its poor OOD generalization performance. In contrast, the hierarchical gating can mitigate this issue, which may be attributed to the learned robust representation due to its regularization effect. Note that the generalization issue of sparse MoEs when transferring to new distributions or domains also exists in the MoE literature (Fedus et al., 2022a), and may be worthy of future exploration.

Given the computation burden of large models, how much time does it take to solve large-scale VRPs? We set the inference batch size as 1, and record the average time for solving a large-scale VRP instance with the size $n = 1000$. The results are shown in Table 6, where we observe 1) large models take more time to solve a VRP instance, but they are still quite efficient compared to classical solvers; 2) the inference time of MVMoE/4E-L is generally smaller than MVMoE/4E, since MVMoE/4E-L is a light version of MVMoE/4E. Moreover, though sparse MoE models are designed to be more parameter-efficient, they still require significant computational resources, especially in terms of GPU memory (since all experts are needed to be loaded during inference). More efficient development (e.g., leveraging parallelism techniques) may be needed for future research.

Extending zero-shot generalization to novel problems. It is non-trivial to enable zero-shot generalization to a novel problem, whose attribute is outside the union of predefined attributes. There may be two potential ways: 1) one can design a general representation or problem formulation for VRP tasks, such as the one leverages MILP formulation (Boisvert et al., 2024), and then train a unified model for MILP; 2) prompt tuning (Li & Liang, 2021) or efficient fine-tuning (Hu et al., 2022) may be potential techniques to efficiently extend the pretrained multi-task solver to a novel problem. They have already achieved superior results in other domains, and we believe this is a good research direction worthy of future exploration.

Table 6. Average time to solve a large-scale VRP instance with the size $n = 1000$.

	CVRP	OVRP	VRPB	VRPTW	VRPL	OVRPTW	OVRPB	OVRPL	VRPBL	VRPBTW	VRPLTW	OVRPBL	OVRPBTW	OVRPLTW	VRPBLTW	OVRPBLTW
POMO-MTL	1.28s	1.35s	1.35s	1.90s	1.45s	1.75s	1.46s	1.47s	1.50s	1.93s	2.20s	1.45s	1.74s	1.78s	2.01s	1.86s
MVMoE/4E	2.14s	2.13s	2.21s	2.92s	2.37s	3.24s	2.28s	2.18s	2.34s	2.96s	2.89s	2.30s	3.08s	2.98s	3.10s	3.17s
MVMoE/4E-L	2.08s	2.06s	2.12s	2.83s	2.19s	2.87s	2.17s	2.17s	2.23s	2.92s	2.97s	2.26s	2.93s	2.96s	3.04s	3.02s

Table 7. Results on CVRPLIB instances, including Set-X (Uchoa et al., 2017) on CVRP and Set-Solomon (Solomon, 1987) on VRPTW. All models are only trained on the uniformly distributed data with the size $n = 100$. Greedy rollout is used by default.

Set-X		POMO		POMO-MTL		MVMoE/4E		MVMoE/4E-L		Set-Solomon		POMO		POMO-MTL		MVMoE/4E		MVMoE/4E-L	
Instance	Opt.	Obj.	Gap	Obj.	Gap	Obj.	Gap	Obj.	Gap	Instance	Opt.	Obj.	Gap	Obj.	Gap	Obj.	Gap	Obj.	Gap
X-n101-k25	27591	30138	9.231%	32482	17.727%	29361	6.415%	**29015**	**5.161%**	R101	1637.7	1805.6	10.252%	1821.2	11.205%	1798.1	9.794%	**1730.1**	**5.641%**
X-n106-k14	26362	39322	49.162%	27369	3.820%	27278	3.475%	**27242**	**3.338%**	R102	1466.6	**1556.7**	**6.143%**	1596.0	8.823%	1572.0	7.187%	1574.3	7.345%
X-n110-k13	14971	15223	1.683%	15151	1.202%	**15089**	**0.788%**	15196	1.503%	R103	1208.7	1341.4	10.979%	**1327.3**	**9.812%**	1328.2	9.887%	1359.4	12.470%
X-n115-k10	12747	16113	26.406%	14785	15.988%	13847	8.629%	**13325**	**4.534%**	R104	971.5	1118.6	15.142%	1120.7	15.358%	1124.8	15.780%	**1098.8**	**13.100%**
X-n120-k6	13332	14085	5.648%	13931	4.493%	14089	5.678%	**13833**	**3.758%**	R105	1355.3	1506.4	11.149%	1514.6	11.754%	1479.4	9.157%	**1456.0**	**7.433%**
X-n125-k30	55539	**58513**	**5.355%**	60687	9.269%	58944	6.131%	58603	5.517%	R106	1234.6	1365.2	10.578%	1380.5	11.818%	1362.4	10.352%	**1353.5**	**9.627%**
X-n129-k18	28940	**29246**	**1.057%**	30332	4.810%	29802	2.979%	29457	1.786%	R107	1064.6	1214.2	14.052%	1209.3	13.592%	**1182.1**	**11.037%**	1196.5	12.391%
X-n134-k13	10916	**11302**	**3.536%**	11581	6.092%	11353	4.003%	11398	4.416%	R108	932.1	1058.9	13.604%	1061.8	13.915%	**1023.2**	**9.774%**	1039.1	11.481%
X-n139-k10	13590	14035	3.274%	13911	2.362%	13825	1.729%	**13800**	**1.545%**	R109	1146.9	1249.0	8.902%	1265.7	10.358%	1255.6	9.478%	**1224.3**	**6.750%**
X-n143-k7	15700	16131	2.745%	16660	6.115%	**16125**	**2.707%**	16147	2.847%	R110	1068.0	1180.4	10.524%	1171.4	9.682%	1185.7	11.021%	**1160.2**	**8.635%**
X-n148-k46	43448	49328	13.533%	50782	16.880%	46758	7.618%	**45599**	**4.951%**	R111	1048.7	1177.2	12.253%	1211.5	15.524%	**1176.1**	**12.148%**	1197.8	14.220%
X-n153-k22	21220	32476	53.040%	26237	23.643%	23793	12.125%	**23316**	**9.877%**	R112	948.6	1063.1	12.070%	1057.0	11.427%	1045.2	10.183%	**1044.2**	**10.082%**
X-n157-k13	16876	17660	4.646%	17510	3.757%	17650	4.586%	**17410**	**3.164%**	RC101	1619.8	2643.0	63.168%	1833.3	13.181%	1774.4	9.544%	**1749.2**	**7.988%**
X-n162-k11	14138	14889	5.312%	14720	4.117%	**14654**	**3.650%**	14662	3.706%	RC102	1457.4	**1534.8**	**5.311%**	1546.1	6.086%	1544.5	5.976%	1556.1	6.771%
X-n167-k10	20557	21822	6.154%	21399	4.096%	21340	3.809%	**21275**	**3.493%**	RC103	1258.0	1407.5	11.884%	**1396.2**	**10.986%**	1402.5	11.486%	1415.3	12.502%
X-n172-k51	45607	49556	8.659%	56385	23.632%	51292	12.465%	**49073**	**7.600%**	RC104	1132.3	**1261.8**	**11.437%**	1271.7	12.311%	1265.4	11.755%	1264.2	11.649%
X-n176-k26	47812	54197	13.354%	57637	20.549%	55520	16.121%	**52727**	**10.280%**	RC105	1513.7	**1612.9**	**6.553%**	1644.9	8.668%	1635.5	8.047%	1619.4	6.980%
X-n181-k23	25569	37311	45.923%	**26219**	**2.542%**	26258	2.695%	26241	2.628%	RC106	1372.7	1539.3	12.137%	1552.8	13.120%	**1505.0**	**9.638%**	1509.5	9.968%
X-n186-k15	24145	25222	4.461%	25000	3.541%	25182	4.295%	**24836**	**2.862%**	RC107	1207.8	1347.7	11.583%	1384.8	14.655%	1351.6	11.906%	**1324.1**	**9.625%**
X-n190-k8	16980	18315	7.862%	**18113**	**6.673%**	18327	7.933%	**18113**	**6.673%**	RC108	1114.2	1305.5	17.169%	1274.4	14.378%	1254.2	12.565%	**1247.2**	**11.939%**
X-n195-k51	44225	49158	11.154%	54090	22.306%	49984	13.022%	**48185**	**8.954%**	RC201	1261.8	2045.6	62.118%	1761.1	39.570%	1577.3	25.004%	**1517.8**	**20.285%**
X-n200-k36	58578	64618	10.311%	61654	5.251%	61530	5.039%	**61483**	**4.959%**	RC202	1092.3	1805.1	65.257%	1486.2	36.062%	1616.5	47.990%	**1480.3**	**35.520%**
X-n209-k16	30656	32212	5.076%	**32011**	**4.420%**	32033	4.492%	32055	4.564%	RC203	923.7	1470.4	59.186%	**1360.4**	**47.277%**	1473.5	59.521%	1479.6	60.182%
X-n219-k73	117595	133545	13.564%	**119887**	**1.949%**	121046	2.935%	120421	2.403%	RC204	783.5	1323.9	68.973%	1331.7	69.968%	1286.6	64.212%	**1232.8**	**57.342%**
X-n228-k23	25742	48689	89.142%	33091	28.549%	31054	20.636%	**28561**	**10.951%**	RC205	1154.0	1568.4	35.910%	1539.2	33.380%	1537.7	33.250%	**1440.8**	**24.850%**
X-n237-k14	27042	29893	10.543%	**28472**	**5.288%**	28550	5.577%	28486	5.340%	RC206	1051.1	1707.5	62.449%	1472.6	40.101%	1468.9	39.749%	**1394.5**	**32.671%**
X-n247-k50	37274	56167	50.687%	45065	20.902%	43673	17.167%	**41800**	**12.143%**	RC207	962.9	1567.2	62.758%	1375.7	42.870%	1442.0	49.756%	**1346.4**	**39.831%**
X-n251-k28	38684	**40263**	**4.082%**	40614	4.989%	41022	6.044%	40822	5.527%	RC208	776.1	1505.4	93.970%	1185.6	52.764%	**1107.4**	**42.688%**	1167.5	50.437%
Avg. Gap		16.629%		9.820%		6.884%		**5.160%**		Avg. Gap		28.054%		21.380%		20.317%		**18.490%**	

Table 8. Results on large-scale CVRPLIB instances. All models are only trained on the uniformly distributed data with the size $n = 100$. Greedy rollout is used by default, except for the last two columns (w. 100 Random Re-Construct (RRC) iterations).

Set-X		POMO		POMO-MTL		MVMoE/4E		MVMoE/4E-L		LEHD		LEHD/4E-L		LEHD RRC100		LEHD/4E-L RRC100	
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%	71438	3.195%	71984	3.984%	70678	2.097%	70304	1.557%
X-n513-k21	24201	30518	26.102%	28510	17.805%	32102	32.647%	31231	29.048%	25624	5.880%	25810	6.648%	24808	2.508%	25026	3.409%
X-n524-k153	154593	201877	30.586%	192249	24.358%	186540	20.665%	182392	17.982%	280556	81.480%	198498	28.400%	194569	25.859%	188040	21.636%
X-n536-k96	94846	106073	11.837%	106514	12.302%	109581	15.536%	108543	14.441%	103785	9.425%	104750	10.442%	102098	7.646%	102414	7.979%
X-n548-k50	86700	103093	18.908%	94562	9.068%	95894	10.604%	95917	10.631%	90644	4.549%	88779	2.398%	88161	1.685%	88383	1.941%
X-n561-k42	42717	49370	15.575%	47846	12.007%	56008	31.114%	51810	21.287%	44728	4.708%	44509	4.195%	43890	2.746%	43847	2.645%
X-n573-k30	50673	83545	64.871%	60913	20.208%	59473	17.366%	57042	12.569%	53482	5.543%	54981	8.502%	53222	5.030%	53309	5.202%
X-n586-k159	190316	229887	20.792%	208893	9.761%	215668	13.321%	214577	12.748%	232867	22.358%	217229	14.141%	211415	11.086%	209941	10.312%
X-n599-k92	108451	150572	38.839%	120333	10.956%	128949	18.901%	125279	15.517%	115377	6.386%	117192	8.060%	112742	3.957%	113914	5.037%
X-n613-k62	59535	68451	14.976%	67984	14.192%	82586	38.718%	74945	25.884%	62484	4.953%	62548	5.061%	61843	3.877%	62191	4.461%
X-n627-k43	62164	84434	35.825%	73060	17.528%	70987	14.193%	70905	14.061%	67568	8.693%	66779	7.424%	65509	5.381%	65117	4.750%
X-n641-k35	63682	75573	18.672%	72643	14.071%	75329	18.289%	72655	14.090%	68249	7.172%	67355	5.768%	66053	3.723%	66196	3.948%
X-n655-k131	106780	127211	19.134%	116988	9.560%	117678	10.206%	118475	10.952%	117532	10.069%	113165	5.980%	112228	5.102%	110707	3.678%
X-n670-k130	146332	208079	42.197%	190118	29.922%	197695	35.100%	183447	25.364%	220927	50.977%	183681	25.523%	184934	26.380%	179615	22.745%
X-n685-k75	68205	79482	16.534%	80892	18.601%	97388	42.787%	89441	31.136%	72946	6.951%	73783	8.178%	71635	5.029%	71723	5.158%
X-n701-k44	81923	97843	19.433%	92075	12.392%	98469	20.197%	94924	15.870%	86327	5.376%	85860	4.806%	84330	2.938%	83965	2.493%
X-n716-k35	43373	51381	18.463%	52709	21.525%	56773	30.895%	52305	20.593%	46502	7.214%	47517	9.554%	45655	5.261%	46028	6.121%
X-n733-k159	136187	159098	16.823%	161961	18.925%	178322	30.939%	167477	22.976%	149115	9.493%	150814	10.740%	144934	6.423%	146338	7.454%
X-n749-k98	77269	87786	13.611%	90582	17.229%	100438	29.985%	94497	22.296%	83439	7.985%	83951	8.648%	81801	5.865%	82346	6.571%
X-n766-k71	114417	135464	18.395%	144041	25.891%	152352	33.155%	136255	19.086%	131487	14.919%	130899	14.405%	126293	10.380%	130511	14.066%
X-n783-k48	72386	90289	24.733%	83169	14.897%	100383	38.677%	92960	28.423%	76766	6.051%	77698	7.338%	75611	4.455%	76014	5.012%
X-n801-k40	73305	124278	69.536%	85077	16.059%	91560	24.903%	87662	19.585%	77546	5.785%	78187	6.660%	75453	2.930%	75318	2.746%
X-n819-k171	158121	193451	22.344%	177157	12.039%	183599	16.113%	185832	17.525%	178558	12.925%	181075	14.517%	171025	8.161%	172779	9.270%
X-n837-k142	193737	237884	22.787%	214207	10.566%	229526	18.473%	221286	14.220%	207709	7.212%	208760	7.754%	203657	5.120%	203254	4.912%
X-n856-k95	88965	152528	71.447%	101774	14.398%	99129	11.425%	106816	20.065%	92936	4.464%	93542	5.145%	91221	2.536%	90878	2.150%
X-n876-k59	99299	119764	20.609%	116617	17.440%	119619	20.463%	114333	15.140%	104183	4.918%	106866	7.620%	103465	4.195%	104399	5.136%
X-n895-k37	53860	70245	30.421%	65587	21.773%	79018	46.710%	64310	19.402%	58028	7.739%	58157	7.978%	56481	4.866%	56749	5.364%
X-n916-k207	329179	399372	21.324%	361719	9.885%	383681	16.557%	374016	13.621%	385208	17.021%	363744	10.500%	355168	7.895%	355635	8.037%
X-n936-k151	132715	237625	79.049%	186262	40.347%	220926	66.466%	190407	43.471%	196547	48.097%	172568	30.029%	169696	27.865%	163087	22.885%
X-n957-k87	85465	130850	53.104%	98198	14.898%	113882	33.250%	105629	23.593%	90295	5.651%	89334	4.527%	88187	3.185%	88384	3.415%
X-n979-k58	118976	147687	24.132%	138092	16.067%	146347	23.005%	139682	17.404%	127972	7.561%	130662	9.822%	125137	5.178%	126113	5.999%
X-n1001-k43	72355	100399	38.759%	87660	21.153%	114448	58.176%	94734	30.929%	76689	5.990%	75933	4.945%	75742	4.681%	75403	4.213%
Avg. Gap		29.658%		**16.796%**		26.408%		19.607%		12.836%		**9.678%**		7.001%		**6.884%**	