Scale-Net: A Hierarchical U-Net Framework for Cross-Scale Generalization in Multi-Task Vehicle Routing Problems

Suyu Liu[1], **Zhiguang Cao**[2], **Nan Yin**[3*], **Yew-Soon Ong**[1,4]
[1]College of Computing and Data Science, Nanyang Technological University
[2]School of Computing and Information Systems, Singapore Management University
[3]Department of Computer Science and Engineering, Hong Kong University of Science and Technology
[4]Centre for Frontier AI Research, Institute of High Performance Computing, Agency for Science, Technology and Research
liusuyu97@gmail.com, zgcao@smu.edu.sg, yinnan8911@gmail.com, asysong@ntu.edu.sg

Abstract

Neural solvers for Vehicle Routing Problems (VRPs) have shown great advantages in solving various kinds of problem types. However, they also face critical challenges in generalizing from small-scale training to large-scale problems and in identifying the most salient topological information for decision-making. To mitigate these gaps, we introduce Scale-Net, a novel hierarchical framework that integrates a U-Net architecture into a unified, multi-task VRP solver. Scale-Net explicitly captures multi-scale structural patterns by processing a nested hierarchy of input graph instances. This enriched, coarse-to-fine representation is extracted by the encoder and fed directly into the decoder, empowering decoder module with superior topological awareness for routing decisions while simultaneously reducing computational overhead in the encoder. We conducted extensive experiments on 16 VRP variants with instances ranging from 50 to 5,000 nodes. The experimental results show that Scale-Net demonstrates significant performance gains over state-of-the-art baselines across in-distribution, zero-shot, and real-world settings. Code is available at https://github.com/lsyysl9711/Scale-Net.

Introduction

The Vehicle Routing Problems (VRPs) and its variants are a group of fundamental challenges in the area of combinatorial optimization with a broad range of applications in related downstream settings such as transportation (Ge, Li, and Tuzhilin 2019; Zhou et al. 2023a), supply chain management (Zhang et al. 2023), and city logistics (Cattaruzza et al. 2017). These problems target at determining the optimal set of routes for a group of vehicles that provide delivery service for customers, while adhering to different kinds of constraints like capacity, time limits, backhaul etc. However, the inherent combinatorial complexity of VRPs makes finding optimal solutions computationally intractable. To mitigate this issue, several sophisticated heuristic algorithms have been proposed with methods like Lin-Kernighan-Helsgaun (LKH) (Lin and Kernighan 1973) and Hybrid Genetic Search (HGS) (Vidal 2022) standing as prominent examples. Although these solvers have achieved remarkable success in finding near-optimal solutions, they are developed on handcrafted rules and problem-specific domain knowledge, which severely limits their ability to generalize to new or unseen constraints that often encountered in real-world scenarios.

In response to these limitations, deep learning has given rise to neural solvers in which cases solvers learn to construct solutions in an automatic manner. Typically, these models utilize an encoder-decoder architecture, where an encoder maps the problem instance into a high-dimensional feature space and an auto-regressive decoder sequentially builds the routes during which an attention mechanism (Vinyals, Fortunato, and Jaitly 2015; Vaswani et al. 2017) guides the decision-making process. Furthermore, there have emerged two frontiers in recent research that have earned great attentions: 1) creating unified foundation models capable of solving multiple VRP variants with a single architecture (Liu et al. 2024; Zhou et al. 2024; Berto et al. 2024; Huang et al. 2025; Li et al. 2025a) and 2) developing specialized large-scale solvers that can handle instances with thousands of nodes for a single task (Luo et al. 2023; Pan et al. 2025; Li et al. 2025b; Luo et al. 2025).

Despite these remarkable progresses, a critical gap remains at the intersection of these frontiers, thereby creating two fundamental challenges. Firstly, there is a *scaling-unification dilemma*: current existing state-of-the-art unified models (Kwon et al. 2020; Zhou et al. 2023a; Berto et al. 2024; Zhou et al. 2024) can handle diverse tasks but are trained on small graphs (typically fewer than 100 nodes) and fail to scale the training into thousands of nodes due to prohibitive computational costs. Conversely, existing large-scale solvers (Luo et al. 2023; Pan et al. 2025; Li et al. 2025b; Luo et al. 2025) are effective on thousand-node instances but are task-specific, lacking the flexibility to generalize across different constraints and tasks. Secondly, most architectures suffer from a *decoder information bottleneck*: decoders typically operate with a narrow view, using only the current node's embedding and dynamic state information to make next step's decision. They are lack of the access into the rich, multi-scale structural context of the graph, which is crucial for making topological-aware routing decisions. Consequently, a single neural solver that is simultaneously unified, scalable with high-fidelity performance remains as an open and under-explored area.

In this paper, we introduce Scale-Net, a novel framework

[*]Corresponding author.

which is designed to directly address these challenges. In specifics, we propose a unified, multi-task neural solver built on a single backbone that achieves both cross-task versatility and large-scale applicability. The core of our innovation is a hierarchical encoder that integrates U-Net modules (Ronneberger, Fischer, and Brox 2015; Gao and Ji 2019) to extract multi-scale structural patterns. By iteratively processing a nested hierarchy of sampled sub-graphs, Scale-Net constructs a coarse-to-fine topological representation. This enriched, multi-level information is then explicitly fed into the decoder, empowering it to make more precise and contextually-aware decisions. Furthermore, by calculating attention scores only on sampled nodes at each layer, our approach significantly reduces computational overhead, making it feasible to train and perform inference on large-scale VRP instances. We summarize our contributions as follows:

- **A Unified and Scalable VRP Solver**: We introduce Scale-Net, the first neural architecture that is designed to function as a single, unified solver for multiple VRP variants while also scaling effectively to large-scale instances involving thousands of nodes. This resolves the critical trade-off between cross-task generalization and large-scale applicability which previous models lack.
- **Hierarchical U-Net for Enriched Decoding**: We propose a novel U-Net-based encoder that constructs a multi-scale feature representation of the input graph. This technique resolves the information bottleneck problem in standard decoders by providing a rich, coarse-to-fine topological-aware context, leading to more precise routing decisions and lower computational overhead.
- **State-of-the-Art Performance Across Scales and Tasks**: We conduct extensive experiments on 16 VRP variants with problem sizes ranging from 50 to 5,000 nodes. The results validate that Scale-Net establishes a new state of the art, outperforming existing baselines in in-distribution, zero-shot, few-shot, large-scale generalization and real-world settings.

Related Works

VRP Solvers

Solvers for Vehicle Routing Problems (VRPs) are typically grouped into three main classes. 1) Traditional Solvers, such as the Lin-Kernighan-Helsgaun (LKH) algorithm (Lin and Kernighan 1973), Hybrid Genetic Search (HGS) (Vidal 2022), and OR-Tools (Perron and Didier 2024), utilize established heuristic search techniques. Their reliance on expert-crafted rules and delicate domain knowledge, however, can restrict their adaptability to unseen problem variants. 2) Neural Solvers, originating from models like Pointer Networks (Vinyals, Fortunato, and Jaitly 2015), use deep learning to build solutions iteratively. The integration of self-attention (Vaswani et al. 2017; Kool, van Hoof, and Welling 2019; Kwon et al. 2020) dramatically boosted performances on both of the in-distribution and out-of distribution settings. Subsequent research has improved generalization through varied training (Kim, Park, and Park 2022; Zhou et al. 2023b), scaled to larger instances (Luo et al. 2023; Pan et al. 2023a; Ye et al. 2024; Cheng et al. 2023) and explored non-autoregressive decoding (Sun and Yang 2023; Xiao et al. 2024). A key limitation is their frequent dependence on auxiliary heuristics for complex instances, which can harm the efficiency. 3) Hybrid Solvers merge both paradigms, often using neural networks to enhance classical heuristics, like generating candidate solutions (Xin et al. 2021) or large-scale neighbour search (Hottung and Tierney 2020; Hottung, Kwon, and Tierney 2021; Hottung, Bhandari, and Tierney 2021). These approaches (Hottung and Tierney 2020; Hottung, Kwon, and Tierney 2021; Xin et al. 2021; Chalumeau et al. 2023; Ma, Cao, and Chee 2024; Chen et al. 2024) show significant success on combinatorial optimization problems, but their requirements for task-specific training often prevents them from generalizing across different VRP types. To address these generalization issues, recent efforts have shifted towards cross-task learning (Liu et al. 2024; Zhou et al. 2024; Berto et al. 2024; Li et al. 2025a). For instance, some methods use attribute composition to tackle diverse VRPs (Liu et al. 2024), while others use Mixture-of-Experts (MoE) models to balance performance and computational cost (Zhou et al. 2024).

However, many of these models are pre-trained on small-to-medium-scale data and they fail to capture fine-grained, multi-scale information from the inputs. Different from these existing works, our approach leverages the power of U-Nets (Ronneberger, Fischer, and Brox 2015) to capture critical information and lower down computational overhead at the same time, making it especially suitable for handling instances containing thousands of nodes.

Large-scale Neural Solvers for VRPs

Generalizing neural solvers into large-scale training or inference setting has attracted great attentions over the past years (Luo et al. 2023; Li et al. 2025b; Pan et al. 2025; Luo et al. 2025). The genre along this direction consists of adaptation, divide-and-conquer and local search methods. 1) For adaptation methods (Drakulic et al. 2023; Zhou et al. 2023a; Luo et al. 2023; Li et al. 2025b), neural solvers are often trained on instances with smaller node numbers (e.g., 100 nodes) and during the testing stage they are broad-casted into the settings with much larger node numbers. During training, different regularization methods (e.g., local reconstruction in (Li et al. 2025b)) or architectures (e.g., heavy decoder in (Luo et al. 2023)) are adopted in order to enhance generalization abilities when these models meet unseen large-scale instances. 2) The divide-and-conquer methods (Pan et al. 2023b; Ye et al. 2024; Zong et al. 2022; Hou et al. 2023; Luo et al. 2025; Pan et al. 2025) often takes a two stage workflow where in the first stage instances are split into smaller sub-graphs and once these sub-problems are solved, these partial solutions will merge into a complete solution. In (Pan et al. 2025), a neural policy module is designed for fabricating more nuanced boundaries between clusters. 3) Different from the above two categories, the family of local search methods (Hottung and Tierney 2020; Hottung, Kwon, and Tierney 2021; Huang et al. 2023) restricts the available choices of next node from the whole graph into a few neighbors. For instance, in (Hottung, Bhandari, and

Tierney 2021) a variational auto-encoder is utilized to map instances into hidden feature space and unconstrained continuous optimization is applied for searching. (Huang et al. 2023) further generalizes large neighbor searching into general integer programming problems.

Different from previous approaches that focus on training separate models for each task, we propose a unified solver. Our model is pre-trained across multiple VRPs with small-scale node numbers and then fine-tuned on instances with thousands of nodes, leading into a single, powerful architecture that handles diverse routing problems simultaneously.

Preliminaries

Definitions of VRPs

The VRPs take the format of fully connected graphs denoted by $G = (V, E)$ where $V = \{v_0, \dots, v_n\}$ is the node set consists of n customer nodes plus one depot node v_0. The $E = \{e_{ij}, i, j = 0, \dots, n\}$ here is the edge set and in our case it consists of edges that link each pair of nodes without self-loops. Each edge e_{ij} is assigned with a cost denoted by c_{ij}. Depending on the type of tasks, we can further acquire two sets of features: *static* features and *dynamic* features. The *static* features of each node v_i consist of coordinates (x_i, y_i), demand d_i and time window $[t_1^i, t_2^i]$. On the other hand, the *dynamic* features for each time step t consist of remaining capacity c_t, current time t_t, current trajectory length l_t and and a binary indicator o_t for whether the current route is open. Note that the *static* features are primarily processed and saved by the encoder module while the *dynamic* features are fed into the decoder at each step to inform the policy.

The Training of Neural Solvers

Following the paradigm of previous works (Kwon et al. 2020; Liu et al. 2024; Zhou et al. 2024), we frame the VRP as a sequential decision-making problem and train our model using reinforcement learning. To be specific, for a generated trajectory τ over the graph G, we modularize the whole process as:

$$p_\theta(\tau|G) = \prod_{t=1}^{T} p_\theta(a_t|a_{t-1}, G), \tag{1}$$

where θ, T and a_t represent model parameter, total time step and actions taken at the t-th time step. The objective function is defined as:

$$\mathcal{L} = E_{\tau \sim p_\theta(\tau|G)}[R(\tau)], \tag{2}$$

where $R(\tau)$ is the length of trajectory τ. By virtue of reinforcement gradient (Williams 1992), the gradient of loss function takes the following format:

$$\nabla_\theta \mathcal{L} = \frac{1}{N}\sum_{i=1}^{N}(R(\tau^i) - b^i(G))\nabla_\theta \log p_\theta(\tau^i|G), \tag{3}$$

where τ_i represents the i-th trajectory and $b^i(G)$ is the introduced baseline term for stabilizing the optimization stage. Note that for some other models like MVMoE(-L) (Zhou et al. 2024), extra regularization loss like expert balancing loss may also exist.

Methods

Our proposed Scale-Net, illustrated in Figure 1, is a unified solver built upon a hierarchical U-Net architecture designed to overcome the scaling and information-bottleneck challenges inherent in previous models. The framework consists of three key stages. First, a hierarchical encoder constructs a multi-scale representation by progressively down-sampling the input graph into a nested series of coarser sub-graphs. Second, a reconstructive decoder up-samples the representation, integrating high-resolution features from the encoder via skip connections. Finally, the policy decoder aggregates these multi-scale features to inform its auto-regressive decision-making process. The following sections provide a detailed description of each component.

Embedding Layer

Given a VRP instance $G = (V, E)$ consists of depot node v_0 and a group of customer nodes $\{v_i; i = 1, \dots n\}$ where each node is assigned with a 2D coordinate (x_i, y_i), demand d_i and time window $[t_1^i, t_2^i]$, we concatenate these inputs and embed them into high-dimensional feature space via a shared linear transform layer:

$$\mathbf{h}_i^{(0)} = \text{Embedding}([x_i, y_i, d_i, t_1^i, t_2^i]). \tag{4}$$

Note that the depot node v_0 does not have demand and time window so for this node we only embed its coordinate-level information. After this, we concatenate features between depot node and customer nodes to formalize a new hidden feature vector:

$$\mathbf{H}^{(0)} = [\mathbf{h}_0^{(0)}; \mathbf{h}_1^{(0)}; \dots; \mathbf{h}_n^{(0)}]. \tag{5}$$

This matrix $\mathbf{H}^{(0)}$ serves as the initial input to the first layer of our hierarchical U-Net encoder.

Pooling Layer of U-Net

Standard neural solvers, such as POMO-MTL (Liu et al. 2024) and MVMoE(-L) (Zhou et al. 2024), feed the initial node embeddings directly into a stack of Transformer layers. This approach, however, presents two significant challenges. Firstly, it creates an *information bottleneck problem*: the decoder module makes decisions based on a single level of feature representation, lacking access to a richer hierarchy of global and local structural information. Secondly, the $O(N^2)$ complexity of the self-attention mechanism makes this design computationally infeasible for large-scale problems with thousands of nodes.

To address both issues, we design a hierarchical encoder based on the U-Net architecture (Ronneberger, Fischer, and Brox 2015). Instead of processing the entire graph at every layer, our encoder iteratively down-samples the graph to build a multi-scale representation. This process is composed of a series of down-sampling blocks followed by a final bottleneck layer. Specifically, the encoder path consists of L down-sampling blocks. Let $\mathbf{H}^{(l)}$ be the matrix of node embeddings and $V^{(l)}$ be the set of active nodes at level l. Initially, at level l, $V^{(0)}$ contains all $n + 1$ nodes from the input graph. Each block performs two sequential operations: *feature extraction* and *graph pooling*.

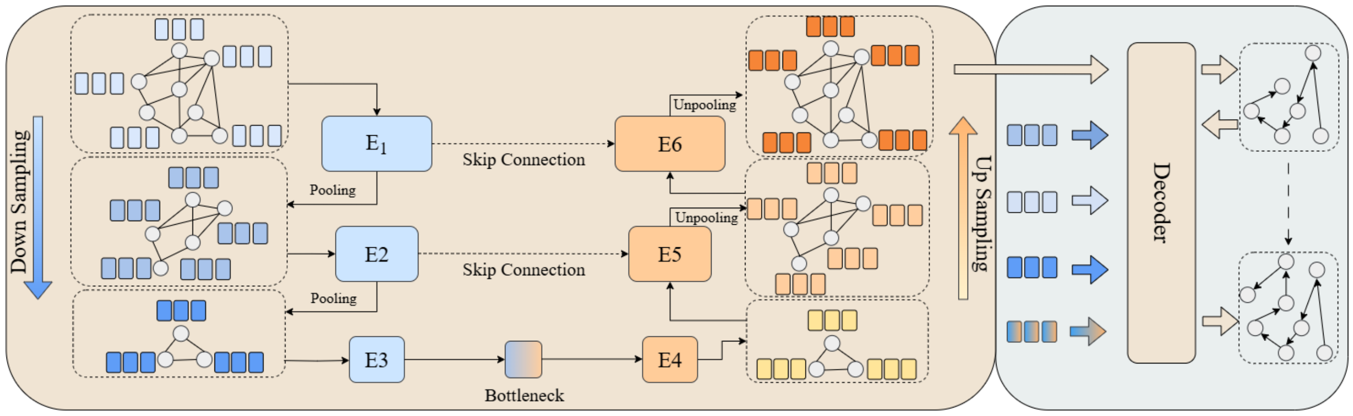

Figure 1: The architecture of our proposed Scale-Net. The encoder path (left) progressively down-samples the input graph using pooling layers to create a nested hierarchy of coarse-to-fine representations. After processing by a bottleneck layer, the decoder path (right) reconstructs the graph features, leveraging skip connections to integrate high-resolution information from the encoder. Finally, the main decoder aggregates these multi-scale features to auto-regressively construct the solution routes. E_i here denotes the i-th layer in the encoder module.

At a given level l, an attention layer operates exclusively on the active nodes $V^{(l)}$ and their features $\mathbf{H}^{(l)}$. This allows nodes within the current sub-graph to exchange information, producing a set of refined, context-aware embeddings $\hat{\mathbf{H}}^{(l)}$:

$$\hat{\mathbf{H}}^{(l)} = \text{Attention}(\mathbf{H}^{(l)}, V^{(l)}). \tag{6}$$

To create the next coarser level of the hierarchy, we perform a pooling operation. We first compute a scalar importance score z_i for each node $v_i \in V^{(l)}$ using a linear projection head on its refined embedding:

$$z_i = \text{Proj}(\hat{\mathbf{h}}_i^{(l)}). \tag{7}$$

We then determine the set of nodes $V^{(l+1)}$ based on the top-k_l scores where k_l is a pre-defined hyperparameter for the number of nodes at level $l+1$. The node features for the next level $\mathbf{H}^{(l+1)}$ are the corresponding embeddings from $\hat{\mathbf{H}}^{(l)}$. This process is repeated L times, yielding a nested hierarchy of graph representations $\{\mathbf{H}^{(0)}, \dots, \mathbf{H}^{(L)}\}$, each capturing the graph structure at a different scale. In addition, since the attention block only operates on the progressively smaller sets of active nodes, the computational overhead is reduced.

After the final down-sampling block, the features of the coarsest graph $\mathbf{H}^{(L)}$ are processed by a bottleneck module to produce a single, dense context vector:

$$\mathbf{H}_{\text{condensed}} = \text{Bottleneck}(\mathbf{H}^{(L)}), \tag{8}$$

which encapsulates a global summary of the problem instance, serves as the initial input to the up-sampling.

Unpooling Layer of U-Net

The up-sampling process begins with the condensed feature vector $\mathbf{H}_{\text{condensed}}$. Then it proceeds iteratively from the coarsest level $l = L$ to the finest level $l = 0$. In details, we propagate features from coarser level $\mathbf{H}^{(l+1)}$ to nodes in $V^{(l)}$ and for nodes in $V^{(l)} \backslash V^{(l+1)}$, we fill in placeholder embeddings (in our case, they are zero-vectors.). To re-integrate high-resolution details, we utilize skip connections to propagate features from corresponding layer of the down-sampling path and then make a concatenation operation:

$$\mathbf{H}_{\text{final}}^{(l)} = \text{Attention}(\text{Concat}(\text{Unpool}(\mathbf{H}^{(l+1)}), \hat{\mathbf{H}}^{(l)})). \tag{9}$$

By repeating this process up to the original graph resolution, we obtain a final set of node embeddings, $\mathbf{H}_{\text{final}}^{(0)}$, where each node's feature vector is enriched with multi-scale context.

Fusion Layer

At each step t of the auto-regressive process, we construct a context vector $\mathbf{c}_t$ that aggregates a comprehensive view of the current state and the problem structure at all scales via a concatenation operation with dynamic feature $\mathbf{h}_{\text{dynamic}}^{(t)}$ at step t:

$$\mathbf{c}_t = \text{Concat}(\mathbf{h}_{\text{current}}^{(t)}, \mathbf{h}_{\text{dynamic}}^{(t)}, \mathbf{H}^{(0)}, \dots, \mathbf{H}^{(L)}), \tag{10}$$

where $\mathbf{h}_{\text{current}}^{(t)}$ is the current node embedding extracted from $\mathbf{H}_{\text{final}}^{(0)}$. This enriched context vector is then used as the query in a final attention block to compute a probability distribution over the available nodes.

Experiments

This section presents a comprehensive empirical evaluation designed to validate the performance, scalability, and generalization capabilities of Scale-Net. To demonstrate its effectiveness as a unified, multi-task solver, we benchmark our framework on a diverse suite of 16 distinct VRP variants spanning a wide range of problem sizes, from small-scale (50-100 nodes) to large-scale (up to 5,000 nodes) instances. All experiments are conducted on a machine equipped with 48 CPUs and four NVIDIA RTX A6000 GPUs, each with 48 GB of CUDA memory.

Type	Model	$n = 1,000$			$n = 2,000$			$n = 3,000$		
		Obj	**Gap**	**Time**	**Obj**	**Gap**	**Time**	**Obj**	**Gap**	**Time**
CVRP	OR-Tools	32.218	0.000%	25m	43.692	0.000%	25m	72.186	0.000%	25m
	ReLD-MVMoE-L	34.965	8.607%	26s	50.628	15.945%	69s	83.588	15.946%	148s
	POMO-MTL-1k	32.443	0.773%	33s	45.764	4.808%	112s	72.431	0.417%	162s
	Scale-Net-1k	**31.834**	**-1.090%**	27s	45.204	3.541%	83s	79.871	10.693%	138s
	Scale-Net-2k	32.793	1.881%	25s	**45.076**	**3.248%**	63s	71.411	-0.989%	133s
	Scale-Net-3k	32.834	2.016%	26s	45.100	3.305%	65s	**71.307**	**-1.136%**	135s
VRPTW	OR-Tools	186.405	0.000%	25m	376.931	0.000%	25m	553.302	0.000%	25m
	ReLD-MVMoE-L	204.298	9.949%	35s	422.736	12.494%	88s	621.070	13.592%	202s
	POMO-MTL-1k	183.805	-1.313%	38s	373.877	-0.987%	147s	559.616	2.043%	215s
	Scale-Net-1k	**182.960**	**-1.774%**	34s	372.698	-1.283%	120s	554.821	1.120%	188s
	Scale-Net-2k	184.609	-0.806%	34s	369.244	-2.196%	82s	541.382	-1.431%	187s
	Scale-Net-3k	184.760	-0.724%	34s	**369.235**	**-2.200%**	85s	**540.206**	**-1.652%**	189s
OVRP	OR-Tools	27.576	0.000%	25m	39.184	0.000%	25m	195.178	0.000%	25m
	ReLD-MVMoE-L	30.036	9.028%	28s	44.998	14.861%	69s	**215.191**	**9.492%**	159s
	POMO-MTL-1k	29.028	5.351%	28s	42.709	9.024%	108s	222.294	13.262%	171s
	Scale-Net-1k	28.229	2.452%	25s	41.889	6.950%	82s	272.987	38.653%	158s
	Scale-Net-2k	29.034	5.377%	24s	41.789	6.694%	63s	238.612	21.457%	150s
	Scale-Net-3k	29.210	6.038%	25s	**41.785**	**6.677%**	64s	236.382	20.265%	150s
VRPL	OR-Tools	34.036	0.000%	25m	52.352	0.000%	25m	369.314	0.000%	25m
	ReLD-MVMoE-L	40.035	17.243%	27s	65.836	25.235%	72s	**383.975**	**3.568%**	172s
	POMO-MTL-1k	35.122	3.158%	30s	53.625	2.290%	115s	405.970	9.153%	186s
	Scale-Net-1k	**33.983**	**-0.058%**	27s	53.158	1.482%	93s	409.195	10.285%	166s
	Scale-Net-2k	35.188	3.472%	26s	**52.455**	**0.192%**	66s	408.520	10.119%	165s
	Scale-Net-3k	35.210	3.571%	27s	52.511	0.304%	68s	405.075	9.150%	165s
VRPB	OR-Tools	32.962	0.000%	25m	48.108	0.000%	25m	268.721	0.000%	25m
	ReLD-MVMoE-L	31.906	-3.155%	27s	47.877	-0.453%	70s	265.551	-1.420%	164s
	POMO-MTL-1k	30.640	-7.003%	29s	44.810	-6.835%	114s	**262.150**	**-2.316%**	176s
	Scale-Net-1k	**29.846**	**-9.396%**	26s	43.980	-8.545%	92s	263.376	-2.015%	153s
	Scale-Net-2k	30.638	-6.999%	25s	**43.597**	**-9.341%**	65s	268.681	-0.085%	154s
	Scale-Net-3k	30.622	-7.041%	26s	43.664	-9.202%	67s	266.544	-0.810%	155s
OVRPTW	OR-Tools	92.051	0.000%	25m	173.529	0.000%	25m	247.902	0.000%	25m
	ReLD-MVMoE-L	104.405	13.745%	33s	199.834	15.165%	84s	287.652	15.640%	195s
	POMO-MTL-1k	93.843	2.060%	36s	174.189	0.161%	139s	268.028	7.701%	205s
	Scale-Net-1k	**93.184**	**1.364%**	33s	173.419	-0.287%	105s	256.657	2.978%	175s
	Scale-Net-2k	95.067	3.422%	33s	172.364	-0.880%	79s	246.641	-1.059%	175s
	Scale-Net-3k	95.105	3.467%	33s	**172.386**	**-0.860%**	78s	**245.782**	**-1.403%**	174s
OVRPB	OR-Tools	28.984	0.000%	25m	42.311	0.000%	25m	65.908	0.000%	25m
	ReLD-MVMoE-L	29.738	2.759%	27s	45.306	7.184%	70s	64.443	-2.434%	157s
	POMO-MTL-1k	28.937	-0.006%	30s	43.138	2.064%	114s	57.616	-12.743%	172s
	Scale-Net-1k	**28.221**	**-2.453%**	26s	42.302	0.115%	94s	59.032	-10.588%	150s
	Scale-Net-MTL-2k	28.834	-0.340%	27s	**41.807**	**-1.066%**	65s	54.156	-17.968%	144s
	Scale-Net-MTL-3k	28.851	-0.286%	27s	41.810	-1.057%	65s	**54.361**	**-17.662%**	144s
OVRPL	OR-Tools	27.204	0.000%	25m	39.328	0.000%	25m	197.772	0.000%	25m
	ReLD-MVMoE-L	30.123	10.737%	27s	46.750	18.886%	71s	**218.923**	**9.567%**	169s
	POMO-MTL-1k	29.336	7.840%	29s	43.617	10.920%	114s	225.764	13.123%	187s
	Scale-Net-1k	**28.409**	**4.453%**	26s	**42.869**	**9.029%**	92s	278.210	38.963%	171s
	Scale-Net-2k	29.380	8.014%	27s	42.893	9.086%	65s	242.583	21.499%	163s
	Scale-Net-3k	29.414	8.137%	26s	42.941	9.210%	66s	239.431	19.825%	163s

Continued on next page

Type	Model	$n = 1,000$			$n = 2,000$			$n = 3,000$		
		Obj	**Gap**	**Time**	**Obj**	**Gap**	**Time**	**Obj**	**Gap**	**Time**
VRPBL	OR-Tools	34.408	0.000%	25m	53.321	0.000%	25m	268.489	0.000%	25m
	ReLD-MVMoE-L	41.137	19.478%	29s	66.097	23.723%	74s	261.721	-3.615%	178s
	POMO-MTL-1k	36.063	5.019%	31s	54.224	1.764%	120s	**258.550**	**-4.453%**	196s
	Scale-Net-1k	**34.947**	**1.910%**	28s	53.882	1.189%	96s	259.703	-4.175%	169s
	Scale-Net-2k	35.800	4.378%	28s	**52.620**	**-1.122%**	69s	264.974	-2.238%	169s
	Scale-Net-3k	35.791	4.374%	29s	52.600	-1.144%	69s	263.463	-2.748%	169s
VRPBTW	OR-Tools	192.325	0.000%	25m	362.567	0.000%	25m	547.855	0.000%	25m
	ReLD-MVMoE-L	216.243	12.847%	36s	415.563	15.013%	91s	627.378	16.401%	215s
	POMO-MTL-1k	194.645	1.340%	39s	366.118	0.849%	154s	558.619	3.156%	233s
	Scale-Net-1k	**193.752**	**0.877%**	37s	364.826	0.526%	113s	553.683	2.220%	202s
	Scale-Net-2k	195.582	1.927%	37s	361.998	-0.240%	84s	541.402	-0.120%	201s
	Scale-Net-3k	195.489	1.877%	36s	**361.995**	**-0.245%**	85s	**540.258**	**-0.341%**	201s
VRPLTW	OR-Tools	198.223	0.000%	25m	379.399	0.000%	25m	543.919	0.000%	25m
	ReLD-MVMoE-L	215.036	8.609%	36s	422.280	12.275%	92s	615.764	13.584%	216s
	POMO-MTL-1k	193.804	-2.429%	40s	373.665	-1.208%	154s	554.089	1.916%	236s
	Scale-Net-1k	**193.042**	**-2.817%**	36s	372.621	-1.463%	107s	549.739	1.070%	204s
	Scale-Net-2k	194.661	-1.901%	36s	**369.078**	**-2.392%**	86s	536.319	-1.484%	204s
	Scale-Net-3k	194.760	-1.840%	36s	369.152	-2.359%	85s	**535.105**	**-1.720%**	202s
OVRPBL	OR-Tools	26.660	0.000%	25m	40.169	0.000%	25m	160.436	0.000%	25m
	ReLD-MVMoE-L	30.413	14.106%	28s	47.580	18.510%	73s	166.424	3.296%	175s
	POMO-MTL-1k	29.552	10.873%	31s	44.543	10.954%	118s	**159.925**	**-0.435%**	192s
	Scale-Net-1k	**28.688**	**7.663%**	28s	43.605	8.646%	97s	179.165	11.448%	173s
	Scale-Net-2k	29.435	10.460%	28s	**43.095**	**7.370%**	68s	162.034	0.759%	168s
	Scale-Net-3k	29.432	10.446%	28s	43.140	7.483%	68s	160.285	-0.319%	167s
OVRPBTW	OR-Tools	95.491	0.000%	25m	170.087	0.000%	25m	250.815	0.000%	25m
	ReLD-MVMoE-L	111.234	16.561%	35s	199.877	17.765%	87s	292.355	16.885%	198s
	POMO-MTL-1k	100.867	5.529%	38s	174.790	2.785%	145s	263.822	5.370%	219s
	Scale-Net-1k	**100.258**	**4.892%**	34s	173.988	2.319%	111s	255.396	1.855%	187s
	Scale-Net-2k	102.357	7.114%	35s	**173.361**	**1.963%**	81s	247.751	-1.216%	187s
	Scale-Net-3k	102.304	7.062%	34s	173.434	2.008%	82s	**246.973**	**-1.532%**	186s
OVRPLTW	OR-Tools	95.031	0.000%	25m	182.703	0.000%	25m	253.948	0.000%	25m
	ReLD-MVMoE-L	105.920	11.559%	34s	199.618	9.700%	87s	292.133	14.876%	197s
	POMO-MTL-1k	95.365	0.315%	38s	173.861	-4.670%	144s	272.543	7.120%	221s
	Scale-Net-1k	**94.810**	**-0.268%**	34s	173.146	-5.071%	116s	261.412	2.583%	187s
	Scale-Net-2k	96.621	1.656%	34s	**172.071**	**-5.644%**	81s	251.330	-1.395%	187s
	Scale-Net-3k	96.668	1.715%	34s	172.080	-5.635%	83s	**250.379**	**-1.772%**	186s
VRPBLTW	OR-Tools	193.039	0.000%	25m	369.980	0.000%	25m	543.943	0.000%	25m
	ReLD-MVMoE-L	215.067	11.618%	38s	422.584	14.676%	15s	629.267	16.359%	228s
	POMO-MTL-1k	193.552	0.227%	41s	372.402	0.608%	160s	560.200	3.094%	249s
	Scale-Net-1k	**192.592**	**-0.236%**	38s	371.053	0.271%	110s	555.088	2.134%	217s
	Scale-Net-2k	194.385	0.774%	38s	**367.860**	**-0.587%**	89s	542.907	-0.184%	217s
	Scale-Net-3k	194.465	0.816%	38s	367.960	-0.545%	91s	**541.626**	**-0.426%**	216s
OVRPBLTW	OR-Tools	93.796	0.000%	25m	179.527	0.000%	25m	253.595	0.000%	25m
	ReLD-MVMoE-L	108.556	15.610%	36s	209.019	16.664%	89s	296.223	16.481%	210s
	POMO-MTL-1k	98.315	4.563%	39s	184.908	3.065%	151s	267.404	5.049%	336s
	Scale-Net-1k	**97.693**	**3.901%**	36s	183.042	1.989%	122s	259.156	1.657%	200s
	Scale-Net-2k	99.847	6.213%	36s	**181.940**	**1.384%**	84s	251.472	-1.382%	199s
	Scale-Net-3k	99.807	6.167%	36s	182.028	1.427%	87s	**250.557**	**-1.756%**	198s

Table 1: In distribution (first 6 tasks) and Out-of-distribution (last 10 tasks) performances for large-scale instances ($n = 1,000, 2,000, 3,000$). The p-values of Scale-Net-1k, Scale-Net-2k and Scale-Net-3k with respect to POMO-MTL-1k are 0.0156, 0.0104 and 0.0249, respectively. These show statistical significance of performances improvements.

Set Ups

Our empirical validation is designed to test Scale-Net's performance across a wide spectrum of tasks and scales. We use a two-stage training pipeline (pre-training and fine-tuning) followed by a series of evaluation scenarios. In specifics, our training process begins with pre-training on small-scale instances (100 nodes) and then fine-tuning on progressively larger-scale ones. We use the Adam optimizer for all stages. During pre-training, a mixture of 6 VRP variants is utilized. After pre-training, we sequentially fine-tune the model on 1,000, 2,000, and 3,000-node instances using the same 6 VRPs variants. The U-Net pooling schedules are $\{75, 50, 25\}$ for 100-node instances and $\{500, 250, 125\}$ for larger-scale instances. On the other hand, we evaluate the pre-trained model on standard benchmarks, including in-distribution (6 tasks) and zero-shot (10 tasks) generalization tests. We also test on real-world datasets: Set-X from CVRPLIB (Uchoa et al. 2017) and the Solomon benchmark (Solomon 1987) for VRPTW instances. For more details regarding to training and testing, please refer to appendix.

Baselines

The baselines used in our study fall into two categories: traditional heuristic solvers and neural solvers. For traditional solvers, we rely on HGS (Vidal 2022), LKH3 (Helsgaun 2017) and OR-Tools (Perron and Didier 2024). For neural solvers, we adopt POMO-MTL (Liu et al. 2024), MVMoE(-L) (Zhou et al. 2024) and ReLD-MVMoE-L (Huang et al. 2025). Due to page limit, we move detailed descriptions into the appendix part.

Results

Results for Large-Size Settings After the pre-training stage on 100 node size, we further fine-tune the pre-trained model on 1,000, 2,000 and 3,000 nodes with 6 VRP tasks. Then, we test its performances from 1,000 to 5,000 nodes on all of the 16 VRP tasks. As shown in Table. 1, the Scale-Net-1k model is dominant, achieving the best performance on all 16 VRP tasks. To ensure a fair comparison, we also fine-tune the original POMO-MTL on 1,000-node instances (denoted by POMO-MTL-1k). The superior results of our model validate that the hierarchical U-Net provides benefits for large-scale problems under both in-distribution and out-of-distribution testing scenarios. Note that fine-tuning baselines like POMO-MTL on 2k/3k nodes will produce out-of-memory problem so we don't include results for them.

For $n = 2,000$ and $n = 3,000$, models continue to exhibit their advantages, achieving best performances on 16 out of 16 and 10 out of 16 tasks, respectively. We also observe that a model fine-tuned on larger-scale instances can sometimes outperform a model tuned on a smaller scale. For example, on several 2,000-node tasks (e.g., OVRP, VRPTW, OVRPTW), the Scale-Net-3k model achieves better solutions and lower gaps than the Scale-Net-2k. This suggests that training on more challenging, larger instances can sometimes acquire a more generalizable capability. Conversely, we notice that a model fine-tuned on a larger-scale setting (e.g., 3,000 nodes) can exhibit degraded performances when tested on smaller-scales (e.g., 1,000 nodes). This is similar to the so called catastrophic forgetting phenomena (Wang et al. 2024) in continual learning where the model adapts its parameters to the new data distribution at the expense of its performance on the original distribution. Addressing this phenomenon remains an important direction for future work.

Besides these, we also include experimental results on large-scale instances in Set-X (Uchoa et al. 2017) which consists of problem instances with node sizes ranging from 500 to 1,000. As shown in Table. 7, Appendix, Scale-Net significantly surpasses all other methods, validating its practical effectiveness on real-world problem sets. Note that in order to produce a fair and comprehensive comparison, we also include the results of POMO-MTL-1k.

Sensitivity Analysis for Pooling Sizes Since the pooling size scheduler is a vital module in the U-Net, we also analyze the model's sensitivity with respect to different choices of pooling schedule. Our findings in Table. 8, Appendix show that Scale-Net is highly robust, as different chosen schedules (in our experiment, we adopt $\{200, 100, 50\}$, $\{300, 150, 75\}$ and $\{500, 250, 125\}$) produce nearly identical results with a low standard deviation of 0.114. Furthermore, we found that performances can be further enhanced with an adaptive schedule tailored to each instance's size. In Table. 9, Appendix, we can observe that after selecting specific pooling size for each node scale, this adaptive approach reduces the overall optimality gap on CVRPLIB from 23.45% to 21.41%, thereby indicating that the current fixed pooling strategy can be further optimized in terms of a more dynamic and fine-grained manner.

Cross-Size Generalization Analysis To further validate the effectiveness of our proposed framework, we also conduct experiments on out-of-distribution testings on node size. In details, we test models on $n = 4,000$ and $n = 5,000$ across 16 VRPs problems. From results in Table. 10, Appendix, our models consistently achieve lower objective values and provide much faster inference speed.

Conclusions

We introduced Scale-Net, a unified neural solver for addressing the critical challenges of scalability and cross-task generalization in Vehicle Routing Problems. By integrating a hierarchical U-Net module into the encoder, our architecture is able to capture rich, multi-scale features from the input graph while alleviating the computational burden of standard attention mechanisms in the encoder module. Our experiments, spanning 16 VRP variants and problem sizes from 50 to 5,000 nodes, validate that Scale-Net achieves state-of-the-art performance. However, this work contains limitations: 1) To manage computational costs, we limited the POMO size, which may harm performances. 2) While our model demonstrates effective training on instances up to 3,000 nodes, extending this capability to the 10,000-node scale and beyond remains a significant challenge. 3) The current U-Net pooling schedule is manually defined. Future works can develop a learnable mechanism that can adaptively choose pooling strategy based on each instance.

Acknowledgements

This research/project is supported by the National Research Foundation, Singapore under its AI Singapore Programme (AISG Award No: AISG3-RP-2022-031), the MTI under its AI Centre of Excellence for Manufacturing (AIMfg) (Award W25MCMF014), and the College of Computing and Data Science, Nanyang Technological University. We also want to express our sincere thanks to all reviewers and area chair for their constructive engagement and valuable suggestions during the rebuttal stage.

References

Berto, F.; Hua, C.; Zepeda, N. G.; Hottung, A.; Wouda, N.; Lan, L.; Tierney, K.; and Park, J. 2024. RouteFinder: Towards Foundation Models for Vehicle Routing Problems. In *ICML 2024 Workshop on Foundation Models in the Wild*.

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

Chalumeau, F.; Surana, S.; Bonnet, C.; Grinsztajn, N.; Pretorius, A.; Laterre, A.; and Barrett, T. 2023. Combinatorial optimization with policy adaptation using latent space search. *Proceedings of the 37th Advances in Neural Information Processing Systems (NeurIPS)*.

Chen, J.; Wang, J.; Zhang, Z.; Cao, Z.; Ye, T.; and Chen, S. 2024. Efficient meta neural heuristic for multi-objective combinatorial optimization. *Proceedings of the 38th Advances in Neural Information Processing Systems (NeurIPS)*.

Cheng, H.; Zheng, H.; Cong, Y.; Jiang, W.; and Pu, S. 2023. Select and optimize: Learning to solve large-scale tsp instances. In *Proceedings of the 26th International Conference on Artificial Intelligence and Statistics (AISTATS)*, 1219–1231.

Drakulic, D.; Michel, S.; Mai, F.; Sors, A.; and Andreoli, J.-M. 2023. Bq-nco: Bisimulation quotienting for efficient neural combinatorial optimization. *Advances in Neural Information Processing Systems*, 36: 77416–77429.

Gao, H.; and Ji, S. 2019. Graph u-nets. In *international conference on machine learning*, 2083–2092. PMLR.

Ge, Y.; Li, H.; and Tuzhilin, A. 2019. Route recommendations for intelligent transportation services. *IEEE Transactions on Knowledge and Data Engineering*, 33(3): 1169–1182.

Helsgaun, K. 2017. An extension of the Lin-Kernighan-Helsgaun TSP solver for constrained traveling salesman and vehicle routing problems. *Roskilde: Roskilde University*, 12: 966–980.

Hottung, A.; Bhandari, B.; and Tierney, K. 2021. Learning a Latent Search Space for Routing Problems using Variational Autoencoders. In *International Conference on Learning Representations*.

Hottung, A.; Kwon, Y.-D.; and Tierney, K. 2021. Efficient Active Search for Combinatorial Optimization Problems. In *Proceedings of the 9th International Conference on Learning Representations (ICLR)*.

Hottung, A.; and Tierney, K. 2020. Neural large neighborhood search for the capacitated vehicle routing problem. In *Proceedings of the 25th European Conference on Artificial Intelligence*, volume 313, 443–450. IOS Press.

Hou, Q.; Yang, J.; Su, Y.; Wang, X.; and Deng, Y. 2023. Generalize learned heuristics to solve large-scale vehicle routing problems in real-time. In *The Eleventh International Conference on Learning Representations*.

Huang, T.; Ferber, A. M.; Tian, Y.; Dilkina, B.; and Steiner, B. 2023. Searching large neighborhoods for integer linear programs with contrastive learning. In *International conference on machine learning*, 13869–13890. PMLR.

Huang, Z.; Zhou, J.; Cao, Z.; and Xu, Y. 2025. Rethinking Light Decoder-based Solvers for Vehicle Routing Problems. In *International Conference on Learning Representations*.

Kim, M.; Park, J.; and Park, J. 2022. Sym-nco: Leveraging symmetricity for neural combinatorial optimization. *Proceedings of the 36th Advances in Neural Information Processing Systems (NeurIPS)*.

Kool, W.; van Hoof, H.; and Welling, M. 2019. Attention, Learn to Solve Routing Problems! In *Proceedings of the 7th International Conference on Learning Representations (ICLR)*.

Kwon, Y.-D.; Choo, J.; Kim, B.; Yoon, I.; Gwon, Y.; and Min, S. 2020. Pomo: Policy optimization with multiple optima for reinforcement learning. *Proceedings of the 34th Advances in Neural Information Processing Systems (NeurIPS)*.

Li, H.; Liu, F.; Zheng, Z.; Zhang, Y.; and Wang, Z. 2025a. CaDA: Cross-Problem Routing Solver with Constraint-Aware Dual-Attention. In *Forty-second International Conference on Machine Learning*.

Li, K.; Liu, F.; Wang, Z.; and Zhang, Q. 2025b. Destroy and Repair Using Hyper-Graphs for Routing. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, 18341–18349.

Lin, S.; and Kernighan, B. W. 1973. An effective heuristic algorithm for the traveling-salesman problem. *Operations research*, 21(2): 498–516.

Liu, F.; Lin, X.; Wang, Z.; Zhang, Q.; Xialiang, T.; and Yuan, M. 2024. Multi-task learning for routing problem with cross-problem zero-shot generalization. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (SIGKDD)*, 1898–1908.

Luo, F.; Lin, X.; Liu, F.; Zhang, Q.; and Wang, Z. 2023. Neural combinatorial optimization with heavy decoder: Toward large scale generalization. *Proceedings of the 37th Advances in Neural Information Processing Systems (NeurIPS)*.

Luo, F.; Lin, X.; Wu, Y.; Wang, Z.; Xialiang, T.; Yuan, M.; and Zhang, Q. 2025. Boosting Neural Combinatorial Optimization for Large-Scale Vehicle Routing Problems. In *The Thirteenth International Conference on Learning Representations*.

Ma, Y.; Cao, Z.; and Chee, Y. M. 2024. Learning to search feasible and infeasible regions of routing problems with flexible neural k-opt. *Proceedings of the 38th Advances in Neural Information Processing Systems (NeurIPS)*.

Pan, X.; Jin, Y.; Ding, Y.; Feng, M.; Zhao, L.; Song, L.; and Bian, J. 2023a. H-tsp: Hierarchically solving the large-scale traveling salesman problem. In *Proceedings of the 37th AAAI Conference on Artificial Intelligence (AAAI)*, volume 37, 9345–9353.

Pan, X.; Jin, Y.; Ding, Y.; Feng, M.; Zhao, L.; Song, L.; and Bian, J. 2023b. H-tsp: Hierarchically solving the large-scale traveling salesman problem. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, 9345–9353.

Pan, Y.; Liu, R.; Chen, Y.; Cao, Z.; and Lin, F. 2025. Hierarchical Learning-based Graph Partition for Large-scale Vehicle Routing Problems. In *Proceedings of the 24th International Conference on Autonomous Agents and Multiagent Systems*, 1604–1612.

Perron, L.; and Didier, F. 2024. CP-SAT.

Ronneberger, O.; Fischer, P.; and Brox, T. 2015. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, 234–241. Springer.

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

Sun, Z.; and Yang, Y. 2023. Difusco: Graph-based diffusion solvers for combinatorial optimization. *Proceedings of the 37th Advances in Neural Information Processing Systems (NeurIPS)*.

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

Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A. N.; Kaiser, Ł.; and Polosukhin, I. 2017. Attention is all you need. *Proceedings of the 31th Advances in Neural Information Processing Systems (NeurIPS)*.

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

Vinyals, O.; Fortunato, M.; and Jaitly, N. 2015. Pointer networks. *Proceedings of the 29th Advances in Neural Information Processing Systems (NeurIPS)*.

Wang, L.; Zhang, X.; Su, H.; and Zhu, J. 2024. A comprehensive survey of continual learning: Theory, method and application. *IEEE transactions on pattern analysis and machine intelligence*, 46(8): 5362–5383.

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

Xiao, Y.; Wang, D.; Li, B.; Wang, M.; Wu, X.; Zhou, C.; and Zhou, Y. 2024. Distilling autoregressive models to obtain high-performance non-autoregressive solvers for vehicle routing problems with faster inference speed. In *Proceedings of the 38th AAAI Conference on Artificial Intelligence (AAAI)*, volume 38, 20274–20283.

Xin, L.; Song, W.; Cao, Z.; and Zhang, J. 2021. Neurolkh: Combining deep learning model with lin-kernighan-helsgaun heuristic for solving the traveling salesman problem. *Proceedings of the 35th Advances in Neural Information Processing Systems (NeurIPS)*.

Ye, H.; Wang, J.; Liang, H.; Cao, Z.; Li, Y.; and Li, F. 2024. Glop: Learning global partition and local construction for solving large-scale routing problems in real-time. In *Proceedings of the 38th AAAI Conference on Artificial Intelligence (AAAI)*, volume 38, 20284–20292.

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

Zhou, J.; Cao, Z.; Wu, Y.; Song, W.; Ma, Y.; Zhang, J.; and Xu, C. 2024. MVMoE: Multi-Task Vehicle Routing Solver with Mixture-of-Experts. In *Proceedings of the 41th International Conference on Machine Learning (ICML)*, 61804–61824.

Zhou, J.; Wu, Y.; Cao, Z.; Song, W.; Zhang, J.; and Chen, Z. 2023a. Learning large neighborhood search for vehicle routing in airport ground handling. *IEEE Transactions on Knowledge and Data Engineering*, 35(9): 9769–9782.

Zhou, J.; Wu, Y.; Song, W.; Cao, Z.; and Zhang, J. 2023b. Towards omni-generalizable neural methods for vehicle routing problems. In *Proceedings of the 40th International Conference on Machine Learning (ICML)*, 42769–42789.

Zong, Z.; Wang, H.; Wang, J.; Zheng, M.; and Li, Y. 2022. Rbg: Hierarchically solving large-scale routing problems in logistic systems via reinforcement learning. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 4648–4658.

Details of Constraints and Problem Types

The complete list of problem types and their corresponding constraints in Table. 1. We utilize 16 VRPs in total. In specifics, the set of in-distribution tasks consists of **CVRP**, **VRPTW**, **OVRP**, **VRPL**, **VRPB** and **OVRPTW**. On the other hand, the out-of-distribution tasks consist of **OVRPB**, **OVRPL**, **VRPBL**, **VRPBTW**, **VRPLTW**, **OVRPBL**, **OVRPBTW**, **OVRPLTW**, **VRPBLTW** and **OVRPBLTW**. For each constraint, we set their hyper-parameters as follows (we primarily focus on settings in (Kwon et al. 2020; Liu et al. 2024; Zhou et al. 2024)):

- **Coordinates:** We focus on uniform distribution setting in which case each node's locations are sampled from $U(0, 1)$ in a unit square.
- **Capacity:** We set capacity to 40 and 50 for $N = 50$ and $N = 100$, respectively. Note that one of the hard constraints involved in each task is that nodes with demands greater than delivery vehicle's current demand are masked.
- **Demand:** We sample the demand of each node from the list $\{1, 2, \ldots, 9\}$. Note that before sending into model, the node demand is normalized by the demand of delivery vehicle.
- **Open Route:** We set it as an indicator vector with all ones. During decoding stage, we need to manually set mask to prevent delivery vehicle from going back to the depot node.
- **Backhauls:** Similar to demand setting, we sample from $\{1, 2, \ldots, 9\}$ as our initial demands. Then, in the same way as that of (Liu et al. 2024), we sample 20% of nodes to be the backhauls nodes.
- **Duration Limit:** We set it to 3, which represents the maximum length of delivery vehicle's route.
- **Time Window:** For the depot node, we assign its time window as $[0, 3]$ and service time for depot is 0 by default. However, service time for customer nodes is set to 0.2 and time window for customer nodes are sampled from uniform distribution.

Configurations with Training

Our main backbone is based on POMO-MTL (Liu et al. 2024). Our training follows a two-stage pipeline: pre-training on small-scale instances and sequential fine-tuning on large-scale instances. We use the Adam optimizer for all training stages. Specifically, the initial pre-training is conducted on a mixture of 6 VRP variants with instance sizes of 50 and 100 nodes. During this stage, we have 5,000 epochs, with 20,000 instances per epoch. The learning rate is initialized as 1e-4, with a weight decay of 1e-6. The learning rate is decayed by a factor of 10 for the final 500 epochs. Batch size is 128 and the pooling sizes in U-Net are set to $\{40, 30, 20\}$ for 50-node instances and $\{75, 50, 25\}$ for 100-node instances. After pre-training, we sequentially fine-tune the models on instances with larger-scale node numbers (1,000, 2,000, and 3,000 nodes) using the same 6 VRP variants. Learning rates for these three periods are set to 1e-5, 1e-6 and 5e-7, respectively. Pooling sizes in this case are $\{500, 250, 125\}$. A list of hyper-parameters is provided in Table. 2. Running time for pre-train and fine-tune take about 5 days and 4 days on a single RTX A6000, respectively. Note that for completeness, we also re-train MVMoE (Zhou et al. 2024) with U-Net from scratch. Results for Scale-POMO-MTL and Scale-MVMoE are presented in Table. 3 and Table. 4. For notations throughout the paper, we use "Scale-Net" and "Scale-POMO-MTL" interchangeably.

Configurations with Small-Scale Testing

For all scenarios, we report the optimality gap relative to the best-known solutions on 1,000 unseen test instances per task. Following standard practice in previous works (Kwon et al. 2020; Zhou et al. 2024), we employ a greedy roll-out policy with 8x instance augmentation (using rotations and clippings) during inference to ensure a fair comparison. In specifics, our evaluation is divided into four settings including in-distribution generalization where model's performances are tested on the 6 VRP variants that were included in training, zero-shot generalization on 10 VRP variants that were not seen during training, few-shot learning on two complex tasks (VRPBLTW and OVRPBLTW) in a low-resourced scenario and real-world benchmarks on the widely-used Set-X from CVRPLIB (Uchoa et al. 2017) for CVRP instances and the Solomon (Solomon 1987) benchmark for VRPTW instances.

Configurations with Large-Scale Testing

To validate the scalability and generalization performance of Scale-Net, we test our fine-tuned models on a comprehensive scenario of large-scale problems. We generate test sets for instances with 1,000, 2,000, 3,000, 4,000 and 5,000 nodes across all 16 VRP variants. For tasks with node sizes in $\{1,000, 2,000, 3,000\}$, we generate 128 instances while for tasks with node sizes in $\{4,000, 5,000\}$, we generate 16 instances. Besides, the vehicle capacity is scaled linearly with the problem size: capacity is set to 500 for 1,000-node instances, 1,000 for 2,000-node instances, and so on, up to 3,000 for 5,000-node instances. We use Google's OR-Tools (Perron and Didier 2024) to compute near-optimal solutions, with a time limit of 200 seconds per instance. Similar to the small-scale testings, we also adopt in-distribution and zero-shot on 6 and 10 VRP tasks, respectively. The evaluation batch sizes for $\{1,000, 2,000, 3,000, 4,000, 5,000\}$ are set to $\{24, 12, 8, 6, 4, 3\}$, respectively.

Baselines

The baselines used in our study fall into two categories: traditional heuristic solvers and neural solvers. Below, we provide specific details for each baseline:

HGS (Vidal 2022): A state-of-the-art metaheuristic that uses a hybrid genetic algorithm to find high-quality solutions for a wide range of VRP variants like CVRP and VRPTW.

LKH3 (Helsgaun 2017): As an extension of the highly effective Lin-Kernighan heuristic (Lin and Kernighan

1973), LKH3 is a state-of-the-art local search algorithm. It iteratively refines a solution by performing sophisticated k-opt moves, where k edges in the current tour are exchanged for a new set to find a better solution.

OR-Tools (Perron and Didier 2024): An open-source software from Google for combinatorial optimization. It provides a unified interface to a wide range of techniques, including constraint programming and specialized local search algorithms for solving VRPs.

POMO-MTL (Liu et al. 2024): A state-of-the-art unified solver that extends the POMO framework (Kwon et al. 2020) to the multi-task setting. It uses a shared encoder-decoder architecture to learn a single policy capable of addressing multiple VRP variants simultaneously.

MVMoE(-L) (Zhou et al. 2024): A recent unified solver that incorporates Mixture-of-Experts (MoE) modules into its encoder and decoder. This allows the model to learn specialized sub-networks for different tasks, improving multi-task performances. We also compare against MVMoE-L, a lightweight variant designed for improved computational efficiency with faster inference.

ReLD-MVMoE-L (Huang et al. 2025): Different from MVMoE-L proposed in (Zhou et al. 2024), ReLD-MVMoE-L enhances the decoder capacity with an extra addition of identity mapping and a feed-forward layer, which demonstrates superior performances on large-scale VRP instances.

Results for Small-Size Settings

The results on 50 and 100 node sizes are provided in Table. 3 and Table. 4. Note that to show the broad availability of utilized U-Net, we also include results of Scale-MVMoE. From the presented results, we can observe that POMO-MTL augmented with U-Net module consistently improves performances on 16 out of 16 VRP tasks. On the other hand, MoE augmented with U-Net architectures boost performances on 5 out of 6 in-distribution tasks and 7 out of 10 zero-shot tasks. In summary, these results show that our hierarchical U-Net is a model-agnostic enhancement. The ability to extract and utilize multi-scale features consistently improves routing decisions across different model architectures and VRP constraints, validating the core hypothesis of our framework in small-scale settings.

Results for Few-Shot Settings

We select VRPBLTW and OVRPBLTW as two tasks to test our proposed framework's ability under few-shot settings. In specifics, for each selected task we tune model for 10 epochs and each epoch contains 10,000 problem instances with 100 nodes. We provide gap of each selected task on each tuning epoch. The gap curves presented in Figure. 1, clearly demonstrate the superiority of our approach. At every stage of the fine-tuning process, Scale-Net maintains a consistently lower optimality gap compared to the baseline

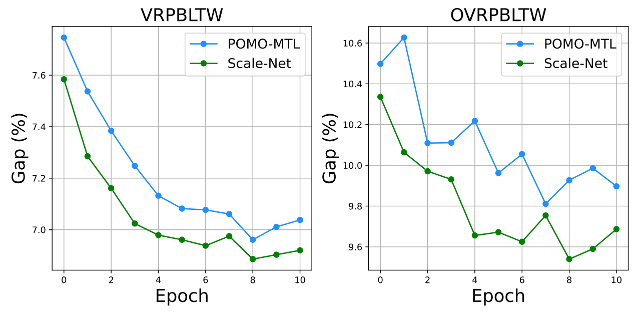

Figure 1: Few-shot testings for VRPBLTW and OVRP-BLTW. The problem size is 100. The compared models are Scale-Net and POMO-MTL.

POMO-MTL. This rapid improvement provides strong evidence that the rich, multi-scale features provided by our U-Net architecture enable faster convergence and better performances in low-resourced scenarios, highlighting the model's strong few-shot learning abilities.

Results for Real-World Scenarios

We conduct real-world testings on two benchmarks: CVR-PLIB (Uchoa et al. 2017) on CVRP instances and Set-Solomon (Solomon 1987) on VRPTW instances. Results are shown in the Table. 5 and Table. 6. As we can see from those two tables, the Scale-POMO-MTL and Scale-MVMoE achieves gaps of 5.82% and 4.58% on CVRPLIB, respectively. Compared with the gaps achieved with original POMO-MTL(9.820%) and MVMoE(6.884%), models augmented with U-Nets consistently produce lower gaps. On the other hand, for real-world VRPTW instances, Scale-POMO-MTL and Scale-MVMoE achieve approximately same performances (13.78% and 13.72%, respectively). Compared with MVMoE-L which achieves 17.84% gap, our model gives better results.

Memory Reduction Analysis

A primary motivation for our U-Net architecture is to mitigate the prohibitive computational and memory costs of standard Transformer-based solvers when applied to large-scale VRPs. Here, we provide a formal analysis to quantify these benefits. Assume that,

- Datas are stored as 32-bit floating-point numbers (FP32), where each number requires 4 bytes.
- The model uses 8 attention heads and 6 encoder layers.
- The calculation is for a single instance in a batch (Batch Size = 1).

Then, we can roughly calculate the memory as follows:

$$\text{Memory (MB)} = \frac{N_{\text{nodes}}^2 \times N_{\text{heads}} \times 4 \text{ bytes}}{1024 \times 1024}. \tag{1}$$

Taking $N_{\text{nodes}} = 5,000$ and $N_{\text{heads}} = 8$ as an example, for a standard transformer, one single encoder layer needs approximately 762MB memory so the total memory for a 6-layer transformer encoder is around 4.6GB. By using the same calculations, we can have that the memory footprint of the encoder in Scale-Net is around 1.6GB which is a 65.2% reduction compared with 4.6 GB. As reflected in Table. 10, our proposed model Scale-Net-3k can achieve superior performances compared to baseline models while keeping speed at a very fast level.

Comparison with Specialized Models

We fine-tune POMO-MTL-100 on two problems (CVRP, VRPTW) with node size 1k and report averaged obj and gaps on all 16 tasks (including 10 zero-shot tasks) in Table. 11. Results show that while specialized models may outperform on single task (e.g., VRPBLTW or VRPTW), unified solver achieves better overall results across all tasks.

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

Table 1: Detailed descriptions of constraints contained in each problem type. We have 16 VRP tasks in total. The first 6 VRP tasks get involved in training stage and the last 10 tasks are used for zero-shot/few-shot testings.

Hyper-Parameters	Value
Training Epochs	5,000
Fine-tuning Epochs	10
Instances in each Training Epoch	20,000
Instances in each Fine-tuning Epoch	10,000
Optimizer	Adam
LR Scheduler	MultiStepLR
LR Milestones	[4,501]
LR Gamma	0.1
Training Learning Rate	1e-4
Fine-tuning Learning Rate in Few-Shot	1e-4
Weight Decay	1e-6
Training Batch Size in Pre-Training	128
Fine-tuning Batch Szie in Few-Shot	128
Evaluation Batch Size in $\{N = 50, 100\}$	64
Problem Scales in Pre-Training	{50, 100}
Node Distribution	U(0,1)
Number of Experts in MoE	4
Auxiliary Loss Weight in MoE	0.001
Gating Mechanism in MoE	node-level, input-choice gating
Embedding Size	128
Hidden Feature Size	512
Number of Encoder Layers	6
QKV Dimension	16
Attention Head Number	8
Logit Clipping	10
Evaluation Type	argmax
Number of Experts in MoE for Routing	2
Pooling Size in Pre-Training	$\{75, 50, 25\}$
Pooling Size in Fine-Tuning	$\{500, 250, 125\}$
Learning Rate in $\{N = 1,000, 2,000, 3,000\}$	$1e-5, 1e-6, 5e-7$
Tuning Epochs in $\{N = 1,000, 2,000, 3,000\}$	$\{500, 100, 100\}$
Tuning Batch Size in $\{N = 1,000, 2,000, 3,000\}$	$\{24, 12, 3\}$
Tuning Epoch Size in $\{N = 1,000, 2,000, 3,000\}$	$\{8,000, 3,000, 1,200\}$
Parameters of POMO-MTL	1.25 (million)
Parameters of Scale-POMO-MTL	2.01 (million)
Parameters of MVMoE	3.69 (million)
Parameters of Scale-MVMoE	4.24 (million)

Table 2: Detailed experiment settings of hyper-parameters. This configuration is consistent with (Zhou et al. 2024).

Type	Model	$n = 50$			$n = 100$			Type	Model	$n = 50$			$n = 100$		
		Obj	**Gap**	**Time**	**Obj**	**Gap**	**Time**			**Obj**	**Gap**	**Time**	**Obj**	**Gap**	**Time**
CVRP	HGS	10.334	0.000%	4.6m	15.504	0.000%	9.1m	VRPTW	HGS	14.509	0.000%	8.4m	24.339	0.000%	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%	20.9m		OR-Tools	14.915	2.694%	10.4m	25.894	6.297%	20.8m
	OR-Tools ($\times$10)	10.418	0.788%	1.7h	15.935	2.751%	3.5h		OR-Tools ($\times$10)	14.665	1.011%	1.7h	25.212	3.482%	3.5h
	POMO-MTL	10.437	0.987%	5s	15.790	1.846%	9s		POMO-MTL	15.032	3.637%	3s	25.610	5.313%	12s
	Scale-POMO-MTL	10.432	0.935%	5s	15.770	1.723%	9s		Scale-POMO-MTL	15.001	3.412%	3s	25.552	5.067%	12s
	MVMoE-L	10.434	0.955%	4s	15.771	1.728%	11s		MVMoE-L	15.013	3.500%	4s	25.519	4.927%	14s
	MVMoE	10.428	0.896%	4s	15.760	1.653%	12s		MVMoE	14.999	3.410%	4s	**25.512**	**4.903%**	15s
	Scale-MVMoE	**10.424**	**0.865%**	4s	**15.744**	**1.554%**	12s		Scale-MVMoE	**14.995**	**3.373%**	4s	25.531	4.986%	15s
OVRP	LKH3	6.511	0.198%	4.5m	9.828	0.000%	5.3m	VRPL	LKH3	10.571	0.790%	7.8m	15.771	0.000%	16.0m
	OR-Tools	6.531	0.495%	10.4m	10.010	1.806%	20.8m		OR-Tools	10.677	1.746%	10.4m	16.496	4.587%	20.8m
	OR-Tools ($\times$10)	6.498	0.000%	1.7h	9.842	0.012%	3.5h		OR-Tools ($\times$10)	10.495	0.000%	1.5h	16.044	1.444%	3.5h
	POMO-MTL	6.671	2.634%	2s	10.169	3.458%	9s		POMO-MTL	10.513	0.201%	3s	15.846	0.479%	10s
	Scale-POMO-MTL	6.667	2.576%	2s	10.148	3.259%	9s		Scale-POMO-MTL	10.506	0.135%	3s	15.822	0.329%	10s
	MVMoE-L	6.665	2.548%	3s	10.145	3.214%	11s		MVMoE-L	10.506	0.131%	3s	15.821	0.323%	12s
	MVMoE	6.655	2.402%	3s	10.138	3.136%	12s		MVMoE	10.501	0.092%	3s	15.812	0.261%	14s
	Scale-MVMoE	**6.651**	**2.236%**	3s	**10.107**	**2.838%**	12s		Scale-MVMoE	**10.499**	**0.071%**	3s	**15.796**	**0.166%**	14s
VRPB	OR-Tools	8.127	0.989%	10.4m	12.185	2.594%	20.8m	OVRPTW	OR-Tools	8.737	0.692%	10.4m	14.635	1.756%	20.8m
	OR-Tools ($\times$10)	8.046	0.000%	1.7h	11.878	0.000%	3.5h		OR-Tools ($\times$10)	8.638	0.000%	1.7h	14.380	0.000%	3.5h
	POMO-MTL	8.282	1.684%	2s	12.072	1.674%	6s		POMO-MTL	8.982	3.470%	3s	15.008	4.411%	12s
	Scale-POMO-MTL	8.171	1.547%	2s	12.039	1.388%	6s		Scale-POMO-MTL	8.961	3.175%	3s	14.954	4.038%	12s
	MVMoE-L	8.176	1.605%	3s	12.063	1.566%	8s		MVMoE-L	8.974	3.322%	6s	14.940	3.941%	14s
	MVMoE	8.170	1.540%	3s	12.071	1.625%	9s		MVMoE	8.964	3.210%	4s	14.927	3.852%	15s
	Scale-MVMoE	**8.161**	**1.419%**	3s	**12.005**	**1.109%**	9s		Scale-MVMoE	**8.943**	**2.981%**	4s	**14.896**	**3.636%**	15s

Table 3: In-distribution performance comparison of solvers on different VRP variants for instance sizes $n = 50$, $n = 100$. Bold values indicate best results while underlined values indicate improved performances with respect to corresponding baselines. The p-values of Scale-POMO-MTL and Scale-MVMoE with respect to their baselines are 0.0002 and 0.0017, which shows statistical significance for improvements.

Type	Model	$n = 50$			$n = 100$			Type	Model	$n = 50$			$n = 100$		
		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 ($\times$10)	5.745	0.000%	1.7h	8.365	0.000%	3.5h		OR-Tools ($\times$10)	6.490	0.000%	1.7h	9.790	0.000%	3.5h
	POMO-MTL	6.116	6.430%	2s	8.979	7.335%	8s		POMO-MTL	6.668	2.734%	2s	10.126	3.441%	9s
	Scale-POMO-MTL	6.102	6.177%	5s	8.964	7.148%	9s		Scale-POMO-MTL	6.663	2.646%	2s	10.102	3.211%	9s
	MVMoE-L	6.122	6.522%	3s	8.972	7.243%	9s		MVMoE-L	6.659	2.590%	3s	10.106	3.244%	9s
	MVMoE	6.092	5.999%	3s	8.959	7.088%	9s		MVMoE	6.650	2.454%	3s	10.097	3.148%	10s
	Scale-MVMoE	**6.067**	**5.568%**	3s	**8.881**	**6.163%**	9s		Scale-MVMoE	**6.647**	**2.409%**	3s	**10.067**	**2.842%**	10s
VRPBL	OR-Tools	8.131	1.254%	10.4m	12.095	2.586%	20.8m	VRPBTW	OR-Tools	15.053	1.857%	10.4m	26.217	2.858%	20.8m
	OR-Tools ($\times$10)	8.029	0.000%	1.7h	11.790	0.000%	3.5h		OR-Tools ($\times$10)	14.771	0.000%	1.7h	25.496	0.000%	3.5h
	POMO-MTL	8.188	1.971%	2s	11.998	1.793%	8s		POMO-MTL	16.055	8.841%	3s	27.319	7.413%	10s
	Scale-POMO-MTL	8.178	1.853%	2s	11.965	1.523%	8s		Scale-POMO-MTL	16.032	8.677%	3s	27.266	7.219%	10s
	MVMoE-L	8.180	1.872%	3s	11.960	1.473%	9s		MVMoE-L	16.041	8.745%	4s	27.265	7.190%	10s
	MVMoE	8.172	1.776%	3s	11.945	1.346%	9s		MVMoE	16.022	8.600%	4s	**27.236**	**7.078%**	11s
	Scale-MVMoE	**8.169**	**1.744%**	3s	**11.934**	**1.256%**	9s		Scale-MVMoE	**15.993**	**8.423%**	4s	27.280	7.258%	11s
VRPLTW	OR-Tools	14.815	1.432%	10.4m	25.823	2.534%	20.8m	OVRPBL	OR-Tools	5.771	0.549%	10.4m	8.555	2.459%	20.8m
	OR-Tools ($\times$10)	14.598	0.000%	1.7h	25.195	0.000%	3.5h		OR-Tools ($\times$10)	5.739	0.000%	1.7h	8.348	0.000%	3.5h
	POMO-MTL	14.961	2.586%	3s	25.619	1.920%	12s		POMO-MTL	6.104	6.306%	2s	8.961	7.343%	8s
	Scale-POMO-MTL	14.937	2.418%	3s	25.576	1.749%	12s		Scale-POMO-MTL	6.085	5.999%	3s	8.954	7.265%	8s
	MVMoE-L	14.953	2.535%	4s	25.529	1.545%	12s		MVMoE-L	6.104	6.310%	3s	8.957	7.300%	9s
	MVMoE	14.937	2.421%	4s	**25.514**	**1.471%**	13s		MVMoE	6.076	5.843%	3s	8.942	7.115%	9s
	Scale-MVMoE	**14.912**	**2.253%**	3s	25.552	1.641%	13s		Scale-MVMoE	**6.049**	**5.361%**	3s	**8.864**	**6.183%**	9s
OVRPBTW	OR-Tools	8.758	0.927%	10.4m	14.713	2.268%	20.8m	OVRPLTW	OR-Tools	8.728	0.656%	10.4m	14.535	1.779%	20.8m
	OR-Tools ($\times$10)	8.675	0.000%	1.7h	14.384	0.000%	3.5h		OR-Tools ($\times$10)	8.669	0.000%	1.7h	14.279	0.000%	3.5h
	POMO-MTL	9.514	9.628%	3s	15.879	10.453%	10s		POMO-MTL	8.987	3.633%	3s	14.896	4.374%	11s
	Scale-POMO-MTL	9.500	9.463%	3s	15.839	10.179%	12s		Scale-POMO-MTL	8.964	3.369%	3s	14.845	4.020%	11s
	MVMoE-L	9.515	9.630%	3s	15.841	10.188%	10s		MVMoE-L	8.974	3.488%	4s	14.839	3.971%	10s
	MVMoE	9.486	9.308%	4s	15.808	9.948%	11s		MVMoE	8.966	3.396%	4s	14.828	3.903%	12s
	Scale-MVMoE	**9.476**	**9.200%**	4s	**15.787**	**9.805%**	15s		Scale-MVMoE	**8.939**	**3.090%**	4s	**14.795**	**3.681%**	12s
VRPBLTW	OR-Tools	14.890	1.402%	10.4m	25.979	2.518%	20.8m	OVRPBLTW	OR-Tools	8.729	0.624%	10.4m	14.496	1.724%	20.8m
	OR-Tools ($\times$10)	14.677	0.000%	1.7h	25.342	0.000%	3.5h		OR-Tools ($\times$10)	8.673	0.000%	1.7h	14.250	0.000%	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
	Scale-POMO-MTL	15.958	8.862%	3s	27.201	7.584%	12s		Scale-POMO-MTL	9.504	9.527%	3s	15.717	10.337%	12s
	MVMoE-L	15.963	8.915%	4s	27.177	7.473%	11s		MVMoE-L	9.518	9.682%	4s	15.706	10.263%	10s
	MVMoE	15.945	8.775%	4s	**27.142**	**7.332%**	12s		MVMoE	9.503	9.516%	4s	15.671	10.009%	11s
	Scale-MVMoE	**15.932**	**8.690%**	4s	27.164	7.436%	15s		Scale-MVMoE	**9.478**	**9.228%**	4s	**15.645**	**9.838%**	15s

Table 4: Out-of-distribution performance comparison for instance sizes $n = 50$ and $n = 100$. Bold values indicate best results while underlined values indicate improved performances with respect to corresponding baselines.

Set-X		POMO		POMO-MTL		MVMoE		MVMoE-L		Scale-Net		Scale-MVMoE	
Instance	Opt	Obj	Gap	Obj	Gap	Obj	Gap	Obj	Gap	Obj	Gap	Obj	Gap
X-n101-k25	27591	30138	9.231%	32482	17.727%	29361	6.415%	29015	5.161%	28895	4.73%	**28754**	**4.22%**
X-n106-k14	26362	39322	49.162%	27369	3.820%	27278	3.475%	27242	3.338%	**26887**	**1.99%**	26930	2.15%
X-n110-k13	14971	15223	1.683%	15151	1.202%	15089	0.788%	15196	1.503%	**15068**	**0.65%**	15087	0.77%
X-n115-k10	12747	16113	26.406%	14785	15.988%	13847	8.629%	13325	4.534%	13369	4.88%	**13228**	**3.77%**
X-n120-k6	13332	14085	5.648%	13931	4.493%	14089	5.678%	13833	3.758%	13687	2.66%	**13583**	**1.88%**
X-n125-k30	55539	58513	5.355%	60687	9.269%	58944	6.131%	58603	5.517%	**57724**	**3.93%**	57919	4.29%
X-n129-k18	28940	**29246**	**1.057%**	30332	4.810%	29802	2.979%	29457	1.786%	29330	1.35%	29327	1.34%
X-n134-k13	10916	11302	3.536%	11581	6.092%	11353	4.003%	11398	4.416%	**11187**	**2.48%**	11382	4.27%
X-n139-k10	13590	14035	3.274%	13911	2.362%	13825	1.729%	13800	1.545%	13862	2.00%	**13801**	**1.55%**
X-n143-k7	15700	16131	2.745%	16660	6.115%	16125	2.707%	16147	2.847%	16194	3.15%	**15981**	**1.79%**
X-n148-k46	43448	49328	13.533%	50782	16.880%	46758	7.618%	45599	4.951%	45962	5.79%	**45387**	**4.46%**
X-n153-k22	21220	32476	53.040%	26237	23.643%	23793	12.125%	23316	9.877%	24917	17.42%	**23240**	**9.52%**
X-n157-k13	16876	17660	4.646%	17510	3.757%	17650	4.586%	17410	3.164%	**17181**	**1.81%**	17272	2.35%
X-n162-k11	14138	14889	5.312%	14720	4.117%	14654	3.650%	14662	3.706%	14731	4.19%	**14483**	**2.44%**
X-n167-k10	20557	21822	6.154%	21399	4.096%	21340	3.809%	21275	3.493%	20953	1.93%	**21138**	**2.83%**
X-n172-k51	45607	49556	8.659%	56385	23.632%	51292	12.465%	49073	7.600%	49582	8.72%	**47182**	**3.45%**
X-n176-k26	47812	54197	13.354%	57637	20.549%	55520	16.121%	52727	10.280%	55326	15.72%	**51982**	**8.72%**
X-n181-k23	25569	37311	45.923%	26219	2.542%	26258	2.695%	26241	2.628%	**26055**	**1.90%**	26132	2.20%
X-n186-k15	24145	25222	4.461%	25000	3.541%	25182	4.295%	**24836**	**2.862%**	24838	2.87%	24856	2.94%
X-n190-k8	16980	18315	7.862%	18113	6.673%	18327	7.933%	18113	6.673%	18097	6.58%	**17895**	**5.39%**
X-n195-k51	44225	49158	11.154%	54090	22.306%	49984	13.022%	48185	8.954%	48318	9.25%	**46335**	**4.77%**
X-n200-k36	58578	64618	10.311%	61654	5.251%	61530	5.039%	61483	4.959%	61194	4.47%	**60953**	**4.05%**
X-n209-k16	30656	32212	5.076%	**32011**	**4.420%**	32033	4.492%	32055	4.564%	31889	5.81%	31842	5.65%
X-n219-k73	117595	133545	13.564%	119887	1.949%	121046	2.935%	120421	2.403%	**119625**	**1.73%**	120251	2.26%
X-n228-k23	25742	48689	89.142%	33091	28.549%	31054	20.636%	28561	10.951%	29988	16.49%	**28025**	**8.87%**
X-n237-k14	27042	29893	10.543%	28472	5.288%	28550	5.577%	28486	5.340%	28506	5.41%	**28294**	**4.63%**
X-n247-k50	37274	56167	50.687%	45065	20.902%	43673	17.167%	**41800**	**12.143%**	45007	20.75%	46299	24.21%
X-n251-k28	38684	**40263**	**4.082%**	40614	4.989%	41022	6.044%	40822	5.527%	40335	4.27%	40072	4.58%
Average Gap		16.629%		9.820%		6.884%		5.160%		5.82%		**4.58%**	

Table 5: Zero-Shot Inference on CVRP benchmark instances from Set-X. Each model is trained on the size n=100, following the settings in (Zhou et al. 2024). The node schedule used here is {75, 50, 25}.

Set-Solomon		POMO		POMO-MTL		MVMoE		MVMoE-L		Scale-Net		Scale-MVMoE	
Instance	Opt	Obj	Gap	Obj	Gap	Obj	Gap	Obj	Gap	Obj	Gap	Obj	Gap
R101	1637.7	1805.6	10.252%	1821.2	11.205%	1798.1	9.794%	1730.1	5.641%	**1705**	**4.11%**	1728	5.51%
R102	1466.6	1556.7	6.143%	1596.0	8.823%	1572.0	7.187%	1574.3	7.345%	1534	4.60%	**1530**	**4.32%**
R103	1208.7	1341.4	10.979%	1327.3	9.812%	1328.2	9.887%	1359.4	12.470%	1325	9.62%	**1319**	**9.13%**
R104	971.5	1118.6	15.142%	1120.7	15.358%	1124.8	15.780%	1098.8	13.100%	1072	10.34%	**1069**	**10.04%**
R105	1355.3	1506.4	11.149%	1514.6	11.754%	1479.4	9.157%	1456.0	7.433%	1461	7.80%	**1453**	**7.21%**
R106	1234.6	1365.2	10.578%	1380.5	11.818%	1362.4	10.352%	1353.5	9.627%	**1331**	**7.81%**	1351	9.43%
R107	1064.6	1214.2	14.052%	1209.3	13.592%	1182.1	11.037%	1196.5	12.391%	**1181**	**10.93%**	1188	11.59%
R108	932.1	1058.9	13.604%	1061.8	13.915%	1023.2	9.774%	1039.1	11.481%	1013	8.68%	**1012**	**8.57%**
R109	1146.9	1249.0	8.902%	1265.7	10.358%	1255.6	9.478%	**1224.3**	**6.750%**	1253	9.25%	1257	9.60%
R110	1068.0	1180.4	10.524%	1171.4	9.682%	1185.7	11.021%	**1160.2**	**8.635%**	1175	10.02%	1177	10.21%
R111	1048.7	1177.2	12.253%	1211.5	15.524%	1176.1	12.148%	1197.8	14.220%	1164	10.99%	**1160**	**10.61%**
R112	948.6	1063.1	12.070%	1057.0	11.427%	1045.2	10.183%	1044.2	10.082%	1025	8.05%	**1016**	**7.11%**
RC101	1619.8	2643.0	63.168%	1833.3	13.181%	1774.4	9.544%	1749.2	7.988%	**1759**	**8.59%**	1807	11.56%
RC102	1457.4	**1534.8**	**5.311%**	1546.1	6.086%	1544.5	5.976%	1556.1	6.771%	1562	7.18%	1552	6.49%
RC103	1258.0	1407.5	11.884%	1396.2	10.986%	1402.5	11.486%	1415.3	12.502%	1411	12.16%	**1392**	**10.65%**
RC104	1132.3	1261.8	11.437%	1271.7	12.311%	1265.4	11.755%	1264.2	11.649%	**1240**	**9.51%**	1256	10.92%
RC105	1513.7	**1612.9**	**6.553%**	1644.9	8.668%	1635.5	8.047%	1619.4	6.980%	1647	8.81%	1633	7.88%
RC106	1372.7	1539.3	12.137%	1552.8	13.120%	1505.0	9.638%	1509.5	9.968%	**1502**	**9.42%**	1513	10.22%
RC107	1207.8	1347.7	11.583%	1384.8	14.655%	1351.6	11.906%	**1324.1**	**9.625%**	1345	11.36%	1338	10.78%
RC108	1114.2	1305.5	17.169%	1274.4	14.378%	1254.2	12.565%	1247.2	11.939%	1242	11.47%	**1227**	**10.12%**
RC201	1261.8	2045.6	62.118%	1761.1	39.570%	1577.3	25.004%	1517.8	20.285%	**1448**	**14.76%**	1506	19.35%
RC202	1092.3	1805.1	65.257%	1486.2	36.062%	1616.5	47.990%	1480.3	35.520%	**1320**	**20.85%**	1376	25.97%
RC203	923.7	1470.4	59.186%	1360.4	47.277%	1473.5	59.521%	1479.6	60.182%	**1215**	**31.54%**	1286	39.22%
RC204	783.5	1323.9	68.973%	1331.7	69.968%	1286.6	64.212%	1232.8	57.342%	1035	32.10%	**1004**	**28.14%**
RC205	1154.0	1568.4	35.910%	1539.2	33.380%	1537.7	33.250%	1440.8	24.850%	**1431**	**24.00%**	1432	24.09%
RC206	1051.1	1707.5	62.449%	1472.6	40.101%	1468.9	39.749%	1394.5	32.671%	1295	23.20%	**1260**	**19.87%**
RC207	962.9	1567.2	62.758%	1375.7	42.870%	1442.0	49.756%	1346.4	39.831%	1217	26.39%	**1180**	**22.55%**
RC208	776.1	1505.4	93.970%	1185.6	52.764%	1107.4	42.688%	1167.5	50.437%	1026	32.20%	**955**	**23.05%**
Average Gap		29.658%		21.380%		20.317%		18.490%		13.78%		**13.72%**	

Table 6: Zero-Shot Inference on VRPTW benchmark instances from Set-Solomon. Each model is trained on the size n=100, following the settings in (Zhou et al. 2024). The node schedule used here is $\{75, 50, 25\}$.

Set-X		POMO		POMO-MTL		MVMoE		MVMoE-L		Scale-Net		POMO-MTL-1k		Scale-Net-1k	
Instance	Opt	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%	73741	6.52%	72575	4.84%	**71305**	**3.00%**
X-n513-k21	24201	30518	26.102%	28510	17.805%	32102	32.647%	31231	29.048%	32489	34.25%	27530	13.76%	**27049**	**11.77%**
X-n524-k153	154593	201877	30.586%	192249	24.358%	186540	20.665%	182392	17.982%	178453	15.43%	183609	18.77%	**170950**	**10.58%**
X-n536-k96	94846	106073	11.837%	106514	12.302%	109581	15.536%	108543	14.441%	104213	9.88%	104934	10.64%	**102419**	**7.98%**
X-n548-k50	86700	103093	18.908%	94562	9.068%	95894	10.604%	95917	10.631%	101440	17.00%	93501	7.84%	**92032**	**6.15%**
X-n561-k42	42717	49370	15.575%	47846	12.007%	56008	31.114%	51810	21.287%	51891	21.48%	49875	16.76%	**47388**	**10.93%**
X-n573-k30	50673	83545	64.871%	60913	20.208%	59473	17.366%	57042	12.569%	59179	16.79%	59429	17.28%	**55628**	**9.78%**
X-n586-k159	190316	229887	20.792%	208893	9.761%	215668	13.321%	214577	12.748%	206453	8.48%	210182	10.44%	**207262**	**8.90%**
X-n599-k92	108451	150572	38.839%	120333	10.956%	128949	18.901%	125279	15.517%	122435	12.89%	121537	12.07%	**118685**	**9.44%**
X-n613-k62	59535	68451	14.976%	67984	14.192%	82586	38.718%	74945	25.884%	69417	16.60%	70646	18.66%	**66190**	**11.18%**
X-n627-k43	62164	84434	35.825%	73060	17.528%	70987	14.193%	70905	14.061%	78058	25.57%	68861	10.77%	**67172**	**8.06%**
X-n641-k35	63682	75573	18.672%	72643	14.071%	75329	18.289%	72655	14.090%	81734	28.35%	71386	12.10%	**68972**	**8.31%**
X-n655-k131	106780	127211	19.134%	116988	9.560%	117678	10.206%	118475	10.952%	112338	5.21%	111700	4.61%	**110591**	**3.57%**
X-n670-k130	146332	208079	42.197%	190118	29.922%	197695	35.100%	183447	25.364%	180840	23.58%	180071	23.06%	**167313**	**14.34%**
X-n685-k75	68205	79482	16.534%	80892	18.601%	97388	42.787%	89441	31.136%	80043	17.36%	79984	17.27%	**76532**	**12.21%**
X-n701-k44	81923	97843	19.433%	92075	12.392%	98469	20.197%	94924	15.870%	106163	29.59%	90909	10.97%	**87985**	**7.40%**
X-n716-k35	43373	51381	18.463%	52709	21.525%	56773	30.895%	52305	20.593%	56327	29.87%	49363	13.81%	**47817**	**10.25%**
X-n733-k159	136187	159098	16.823%	161961	18.925%	178322	30.939%	167477	22.976%	152528	12.00%	157710	15.80%	**148664**	**9.16%**
X-n749-k98	77269	87786	13.611%	90582	17.229%	100438	29.985%	94497	22.296%	91968	19.02%	88701	14.80%	**85097**	**10.13%**
X-n766-k71	114417	135464	18.395%	144041	25.891%	152352	33.155%	136255	19.086%	135681	18.58%	138285	20.86%	**128573**	**12.37%**
X-n783-k48	72386	90289	24.733%	83169	14.897%	100383	38.677%	92960	28.423%	96668	33.55%	82551	14.04%	**80828**	**11.66%**
X-n801-k40	73305	124278	69.536%	85077	16.059%	91560	24.903%	87662	19.585%	106335	45.06%	82251	12.20%	**80516**	**9.84%**
X-n819-k171	158121	193451	22.344%	177157	12.039%	183599	16.113%	185832	17.525%	175251	10.83%	174544	10.39%	**172069**	**8.82%**
X-n837-k142	193737	237884	22.787%	214207	10.566%	229526	18.473%	221286	14.220%	221867	14.52%	212833	9.86%	**208364**	**7.55%**
X-n856-k95	88965	152528	71.447%	101774	14.398%	99129	11.425%	106816	20.065%	106194	19.37%	98597	10.83%	**96856**	**8.87%**
X-n876-k59	99299	119764	20.609%	116617	17.440%	119619	20.463%	114333	15.140%	127378	28.28%	109742	10.52%	**106431**	**7.18%**
X-n895-k37	53860	70245	30.421%	65587	21.773	79018	46.710%	64310	19.402%	85532	58.80%	62581	16.19%	**60910**	**13.09%**
X-n916-k207	329179	399372	21.324%	361719	9.885%	383681	16.557%	374016	13.621%	364443	10.71%	357783	8.69%	**358399**	**8.88%**
X-n936-k151	132715	237625	79.049%	186262	40.347%	220926	66.466%	190407	43.471%	176354	32.88%	170655	28.59%	**165042**	**24.36%**
X-n957-k87	85465	130850	53.104%	98198	14.898%	113882	33.250%	105629	23.593%	129670	51.72%	94740	10.85%	**92802**	**8.58%**
X-n979-k58	118976	147687	24.132%	138092	16.067%	146347	23.005%	139682	17.404%	152185	27.91%	129946	9.22%	**129215**	**8.61%**
X-n1001-k43	72355	100399	38.759%	87660	21.153%	114448	58.176%	94734	30.929%	107299	48.30%	83267	15.08%	**81118**	**12.11%**
Average Gap		29.658%		16.769%		26.048%		19.607%		23.45%		13.49%		**9.85%**	

Table 7: Zero-Shot Inference on large-scale CVRP instances from Set-X. Each model is trained on the size n=100, following the setting in (Zhou et al. 2024). The node schedule used here is {75, 50, 25}. For Scale-POMO-MTL and POMO-MTL, we further fine tune them on problem instances with 1,000 nodes to get Scale-POMO-MTL-1k and POMO-MTL-1k, respectively.

Scheduler	Gap
{75, 50, 25}	9.85%
{200, 100, 50}	**9.65%**
{300, 150, 75}	9.83%
{500, 250, 125}	9.97%
———————	9.825 ± 0.114

Table 8: Average gap on large-scale CVRPLIB datasets tested with different choices of node scheduler. The tested model here is Scale-POMO-MTL-Tuned which is pre-trained on 100 and fine-tuned on 1,000 node size.

Scale	*Scheduler*	Gap
500	{350, 125, 50}	13.44%
600	{350, 150, 50}	15.67%
700	{400, 200, 100}	23.02%
800	{500, 250, 125 }	29.04%
900	{600, 300, 150}	23.55%
1,000	{700, 350, 175 }	61.43%
AVG	—————————	21.41%

Table 9: Average gap on large-scale CVRPLIB datasets tested with node schedulers set for each node scale from 500 to 1,000. The tested model is Scale-POMO-MTL which is pre-trained on 100 node size.

Type	Model	$n = 4,000$		$n = 5,000$		Type	Model	$n = 4,000$		$n = 5,000$	
		Obj	**Time**	**Obj**	**Time**			**Obj**	**Time**	**Obj**	**Time**
CVRP	ReLD-MVMoE-L	74.247	34s	84.785	89s	VRPTW	ReLD-MVMoE-L	809.504	47s	1048.833	131s
	POMO-MTL-1k	67.816	32s	80.047	82s		POMO-MTL-1k	715.142	44s	933.561	119s
	Scale-Net-1k	85.398	29s	159.211	49s		Scale-Net-1k	712.217	59s	928.720	61s
	Scale-Net-2k	65.802	27s	77.311	42s		Scale-Net-2k	692.539	38s	900.838	62s
	Scale-Net-3k	**64.692**	26s	**73.773**	43s		Scale-Net-3k	**690.667**	38s	**897.216**	62s
OVRP	ReLD-MVMoE-L	69.161	33s	79.849	89s	VRPL	ReLD-MVMoE-L	104.549	37s	126.743	101s
	POMO-MTL-1k	65.664	32s	78.099	82s		POMO-MTL-1k	83.409	35s	102.270	92s
	Scale-Net-1k	72.514	29s	101.499	49s		Scale-Net-1k	110.999	32s	256.028	56s
	Scale-Net-2k	62.261	27s	72.193	45s		Scale-Net-2k	79.419	30s	96.290	48s
	Scale-Net-3k	**61.671**	26s	**70.802**	44s		Scale-Net-3k	**79.045**	29s	**94.204**	48s
VRPB	ReLD-MVMoE-L	73.63	36s	85.063	99s	OVRPTW	ReLD-MVMoE-L	357.814	43s	501.333	122s
	POMO-MTL-1k	68.017	35s	80.192	89s		POMO-MTL-1k	311.810	42s	449.129	112s
	Scale-Net-1k	86.716	30s	185.619	52s		Scale-Net-1k	308.090	38s	439.578	56s
	Scale-Net-2k	64.945	29s	76.136	46s		Scale-Net-2k	296.508	35s	419.415	57s
	Scale-Net-3k	**64.267**	29s	**73.884**	46s		Scale-Net-3k	**295.191**	35s	**417.455**	57s
OVRPB	ReLD-MVMoE-L	69.906	36s	80.392	100s	OVRPL	ReLD-MVMoE-L	75.299	36s	83.905	99s
	POMO-MTL-1k	66.095	35s	78.490	90s		POMO-MTL-1k	68.828	35s	81.155	90s
	Scale-Net-1k	72.469	35s	109.889	53s		Scale-Net-1k	77.370	31s	106.438	52s
	Scale-Net-2k	62.401	29s	72.424	50s		Scale-Net-2k	65.237	29s	74.009	49s
	Scale-Net-3k	**62.049**	29s	**71.076**	48s		Scale-Net-3k	**65.077**	29s	**72.995**	47s
VRPBL	ReLD-MVMoE-L	106.472	39s	120.755	110s	VRPBTW	ReLD-MVMoE-L	801.962	49s	970.412	141s
	POMO-MTL-1k	85.407	38s	98.920	101s		POMO-MTL-1k	703.987	47s	854.010	128s
	Scale-Net-1k	105.294	32s	185.731	60s		Scale-Net-1k	702.196	41s	850.545	65s
	Scale-Net-2k	81.215	32s	92.882	52s		Scale-Net-2k	683.363	41s	822.994	66s
	Scale-Net-3k	**80.503**	32s	**90.524**	51s		Scale-Net-3k	**680.547**	41s	**819.825**	66s
VRPLTW	ReLD-MVMoE-L	874.967	50s	1031.964	142s	OVRPBL	ReLD-MVMoE-L	74.803	38s	87.702	108s
	POMO-MTL-1k	778.297	48s	918.949	130s		POMO-MTL-1k	69.632	37s	83.911	98s
	Scale-Net-1k	774.404	42s	914.855	66s		Scale-Net-1k	78.481	34s	109.171	57s
	Scale-Net-2k	754.685	42s	885.484	67s		Scale-Net-2k	65.513	32s	76.232	54s
	Scale-Net-3k	**752.286**	42s	**882.449**	66s		Scale-Net-3k	**65.305**	31s	**75.525**	51s
OVRPBTW	ReLD-MVMoE-L	351.760	45s	490.580	132s	OVRPLTW	ReLD-MVMoE-L	405.967	46s	488.808	129s
	POMO-MTL-1k	304.572	44s	434.622	120s		POMO-MTL-1k	357.916	45s	435.315	120s
	Scale-Net-1k	301.094	38s	427.121	61s		Scale-Net-1k	354.200	38s	427.209	60s
	Scale-Net-2k	290.559	38s	408.863	61s		Scale-Net-2k	341.125	38s	407.471	61s
	Scale-Net-3k	**289.484**	38s	**406.608**	62s		Scale-Net-3k	**339.992**	38s	**405.266**	61s
VRPBLTW	ReLD-MVMoE-L	862.763	54s	942.443	151s	OVRPBLTW	ReLD-MVMoE-L	388.932	49s	457.153	139s
	POMO-MTL-1k	764.625	51s	829.857	139s		POMO-MTL-1k	340.020	47s	401.080	128s
	Scale-Net-1k	761.852	45s	825.791	71s		Scale-Net-1k	336.819	41s	394.747	65s
	Scale-Net-2k	743.103	45s	798.852	71s		Scale-Net-2k	325.377	41s	377.104	65s
	Scale-Net-3k	**740.518**	45s	**795.933**	72s		Scale-Net-3k	**323.938**	40s	**375.455**	65s

Table 10: In-distribution and out-of-distribution performance comparison of solvers on different VRP variants for instance sizes $n = 4,000$, $n = 5,000$. Bold values indicate best results.

Type	Model	$n = 1,000$ Obj	$n = 1,000$ Gap	Type	Model	$n = 1,000$ Obj	$n = 1,000$ Gap
CVRP	CVRP-1K	31.944	-0.771%	VRPTW	CVRP-1k	238.189	28.945%
	VRPTW-1k	39.301	22.073%		VRPTW-1k	**181.696**	**-2.443%**
	Scale-Net-1k	**31.834**	**-1.090%**		Scale-Net-1k	182.960	-1.774%
OVRP	CVRP-1k	30.173	9.659%	VRPL	CVRP-1k	35.472	4.129%
	VRPTW-1k	36.338	32.136%		VRPTW-1k	44.224	29.766%
	Scale-Net-1k	**28.229**	**2.452%**		Scale-Net-1k	**33.983**	**-0.058%**
VRPB	CVRP-1k	30.500	-7.424%	OVRPTW	CVRP-1k	127.613	39.644%
	VRPTW-1k	37.149	12.745%		VRPTW-1k	110.282	19.797%
	Scale-Net-1k	**29.846**	**-9.396%**		Scale-Net-1k	**93.184**	**1.364%**
OVRPB	CVRP-1k	29.744	2.774%	OVRPL	CVRP-1k	30.163	10.879%
	VRPTW-1k	35.471	22.552%		VRPTW-1k	37.448	37.604%
	Scale-Net-1k	**28.221**	**-2.453%**		Scale-Net-1k	**28.409**	**4.453%**
VRPBL	CVRP-1k	36.744	6.918%	VRPBTW	CVRP-1k	257.880	35.557%
	VRPTW-1k	45.012	30.924%		VRPTW-1k	**192.836**	**0.395%**
	Scale-Net-1k	**34.947**	**1.910%**		Scale-Net-1k	193.752	0.877%
VRPLTW	CVRP-1k	249.098	26.744%	OVRPBL	CVRP-1k	29.720	11.506%
	VRPTW-1k	**191.698**	**-3.510%**		VRPTW-1k	36.609	37.310%
	Scale-Net-1k	193.042	-2.817%		Scale-Net-1k	**28.688**	**7.663%**
OVRPBTW	CVRP-1k	139.244	46.698%	OVRPLTW	CVRP-1k	128.954	36.317%
	VRPTW-1k	118.811	24.152%		VRPTW-1k	112.135	17.883%
	Scale-Net-1k	**100.258**	**4.892%**		Scale-Net-1k	**94.810**	**-0.268%**
VRPBLTW	CVRP-1k	255.660	33.671%	OVRPBLTW	CVRP-1k	136.285	45.830%
	VRPTW-1k	**191.515**	**-0.797%**		VRPTW-1k	115.953	23.157%
	Scale-Net-1k	192.592	-0.236%		Scale-Net-1k	**97.693**	**3.901%**

Table 11: In-distribution and out-of-distribution performance comparison of solvers on different VRP variants for instance sizes $n = 1,000$. Bold values indicate best results. The average performances of CVRP-1k and VRPTW-1k are 111.71 (20.69%) and 95.40 (18.98%), respectively. Both lag behind Scale-Net-1k (87.13, 0.80%).

References

Helsgaun, K. 2017. An extension of the Lin-Kernighan-Helsgaun TSP solver for constrained traveling salesman and vehicle routing problems. *Roskilde: Roskilde University*, 12: 966–980.

Huang, Z.; Zhou, J.; Cao, Z.; and Xu, Y. 2025. Rethinking Light Decoder-based Solvers for Vehicle Routing Problems. In *International Conference on Learning Representations*.

Kwon, Y.-D.; Choo, J.; Kim, B.; Yoon, I.; Gwon, Y.; and Min, S. 2020. Pomo: Policy optimization with multiple optima for reinforcement learning. *Proceedings of the 34th Advances in Neural Information Processing Systems (NeurIPS)*.

Lin, S.; and Kernighan, B. W. 1973. An effective heuristic algorithm for the traveling-salesman problem. *Operations research*, 21(2): 498–516.

Liu, F.; Lin, X.; Wang, Z.; Zhang, Q.; Xialiang, T.; and Yuan, M. 2024. Multi-task learning for routing problem with cross-problem zero-shot generalization. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (SIGKDD)*, 1898–1908.

Perron, L.; and Didier, F. 2024. CP-SAT.

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

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

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

Zhou, J.; Cao, Z.; Wu, Y.; Song, W.; Ma, Y.; Zhang, J.; and Xu, C. 2024. MVMoE: Multi-Task Vehicle Routing Solver with Mixture-of-Experts. In *Proceedings of the 41th International Conference on Machine Learning (ICML)*, 61804–61824.